Quora-1.png

Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

Problem Statement

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

1.2 Sources/Useful Links

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

  Useful Links
- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments (https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments)
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0 (https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0)
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning (https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning)
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30 (https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30)

1.3 Real world/Business Objectives and Constraints

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

2.3 Train and Test Construction

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

3. Exploratory Data Analysis

In [0]:

```
!pip install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.org/packages/5c/1a/883e47df323437ae
fa0d0a92ccfb38895d9416bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz (https://
files.pythonhosted.org/packages/5c/1a/883e47df323437aefa0d0a92ccfb38895d9416
bd0b56262c2e46a47767b8/Distance-0.1.3.tar.gz) (180kB)
     |████████████████████████████████| 184kB 2.8MB/s eta 0:00:01
Building wheels for collected packages: distance
  Building wheel for distance (setup.py) ... done
  Created wheel for distance: filename=Distance-0.1.3-cp36-none-any.whl size
=16261 sha256=111bd016c006cf8aac3d3cffa1e1fdad627ca1781b7e8fdc2510121582d3ee
39
  Stored in directory: /root/.cache/pip/wheels/d5/aa/e1/dbba9e7b6d397d645d0f
12db1c66dbae9c5442b39b001db18e
Successfully built distance
Installing collected packages: distance
Successfully installed distance-0.1.3
```

In [0]:

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

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

In [0]:

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



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

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

In [0]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clien
t_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.co
m&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=
email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2f
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%
2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleap
i.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803
-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=ur
n%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%
2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.co
m%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.re
adonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

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

3.1 Reading data and basic stats

In [0]:

```
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/train.csv")

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

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

In [0]:

```
df.head()
```

Out[6]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [0]:

```
df.info()
```

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

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

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

3.2.1 Distribution of data points among output classes

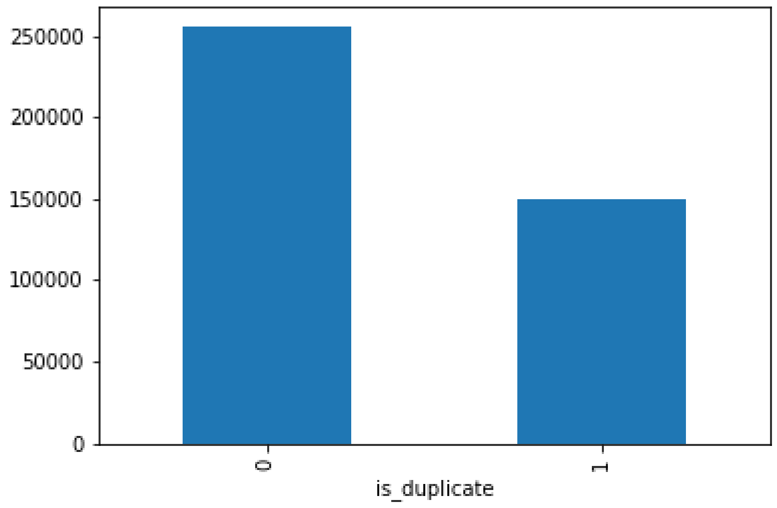
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [0]:

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

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f45716bc8d0>
```



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
print('\n~> Question pairs are Similar (is_duplicate = 1):\n   {}%'.format(round(df['is_dup
```

```
~> 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

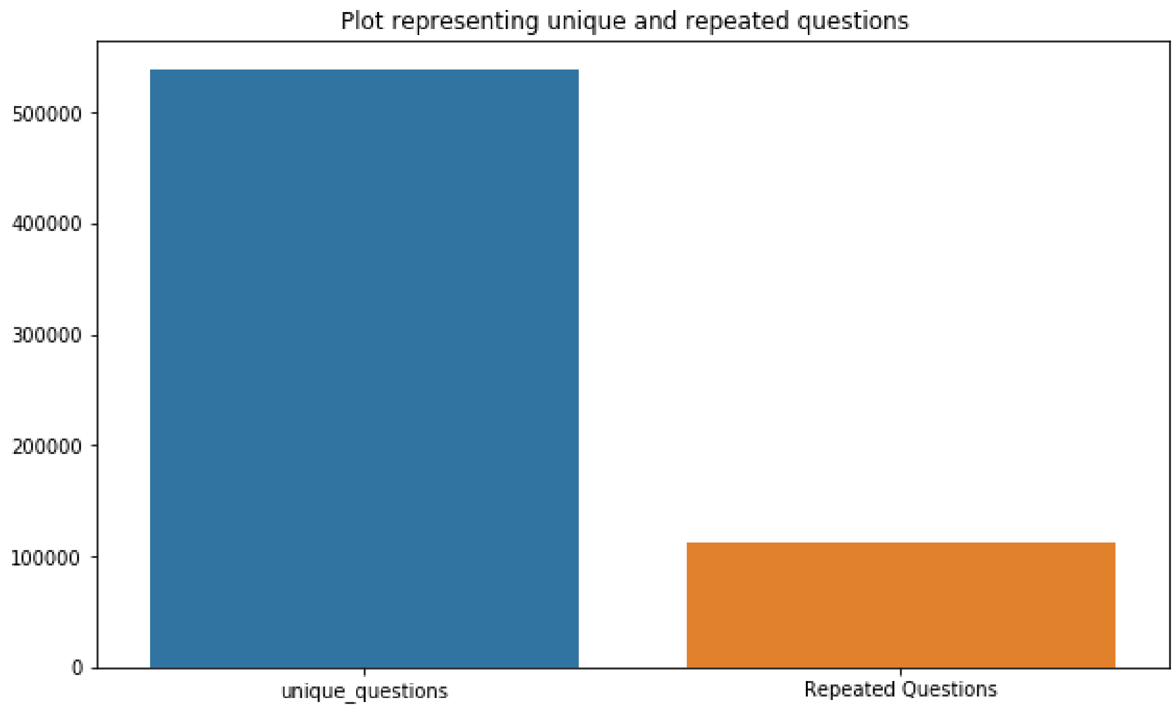
In [0]:

```
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_mo

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

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 (20.779539
45937505%)

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

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

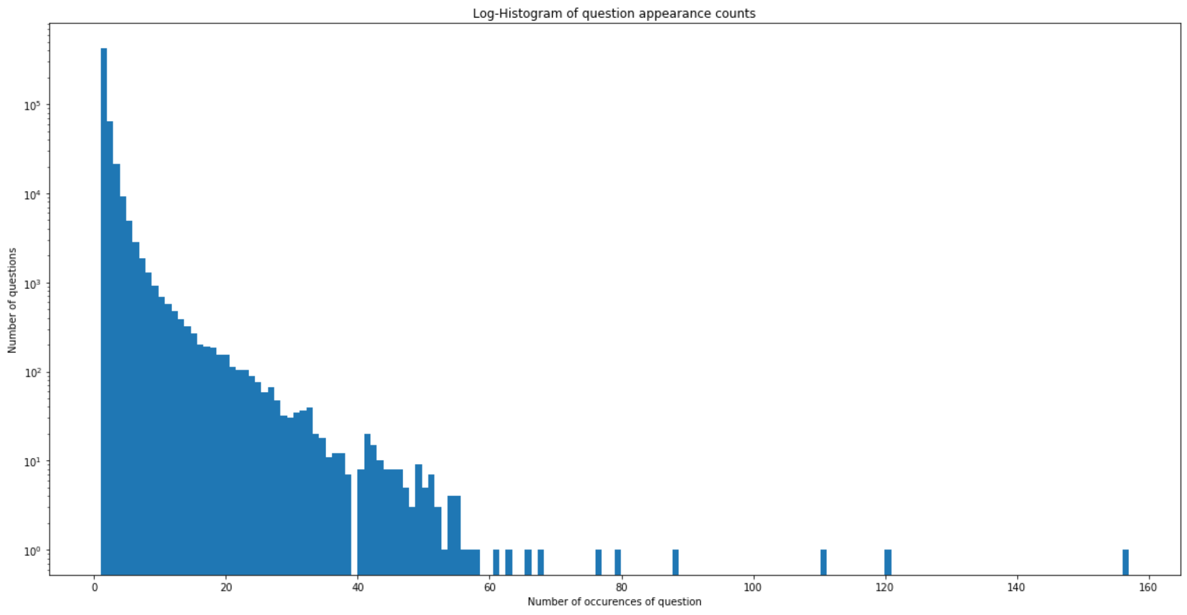
```
Number of duplicate questions 0
```

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_
```

```
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[17]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	1	1	73	59

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24} [/math] i...	0	1	1	50	65
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	3	1	76	39

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

In [0]:

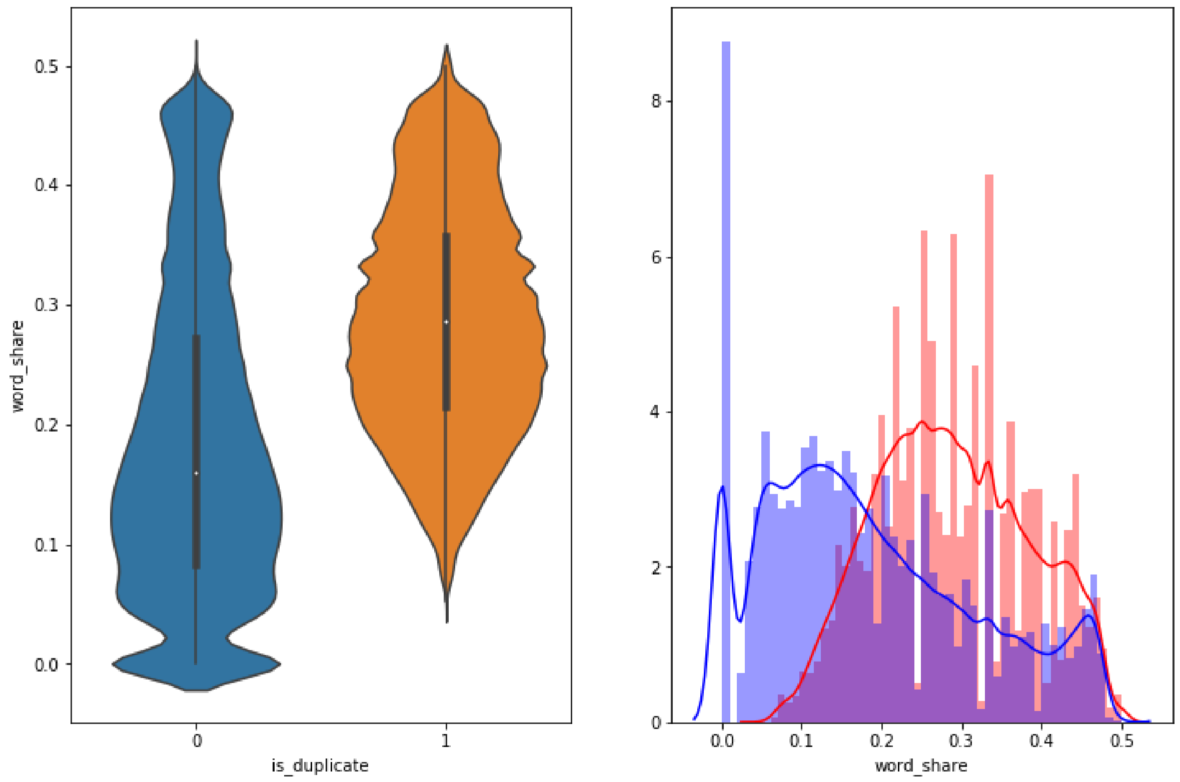
```
print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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

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

```
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()
```



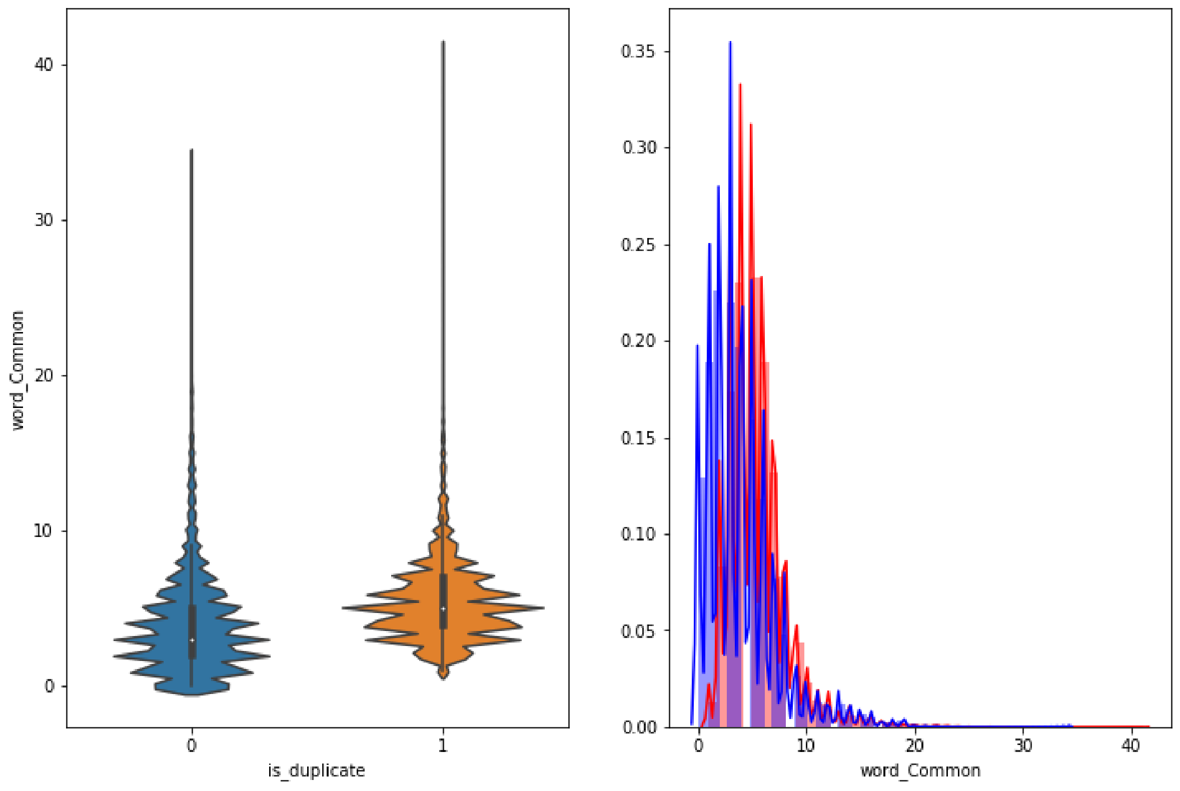
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity

- 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

In [0]:

```
!pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.org/packages/43/ff/74f23998ad2f93b9
45c0309f825be92e04e0348e062026998b5eefef4c33/fuzzywuzzy-0.18.0-py2.py3-none-
any.whl (https://files.pythonhosted.org/packages/43/ff/74f23998ad2f93b945c03
09f825be92e04e0348e062026998b5eefef4c33/fuzzywuzzy-0.18.0-py2.py3-none-any.w
hl)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0
```

In [0]:

```
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
```

In [0]:

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

In [0]:

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

Out[24]:

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1len	q2len	q1_r
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	1	1	66	57	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	4	1	51	88	

3.4 Preprocessing of Text

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

In [0]:

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

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

Out[25]:

```
True
```

In [0]:

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

STOP_WORDS = stopwords.words("english")


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


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

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


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


    return x
```

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In [0]:

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

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

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

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

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

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

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


    token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_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_DI
    token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DI

    # 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"]),

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

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

In [0]:

```
if os.path.isfile('nlp_features_train.csv'):
    df = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
    df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)
```

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

Out[28]:

	id	qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max
0	0	1	2	what is the step by step guide to invest in sh...	what is the step by step guide to invest in sh...	0	0.999980	0.833319	0.999983	0.999983
1	1	3	4	what is the story of kohinoor koh i noor dia...	what would happen if the indian government sto...	0	0.799984	0.399996	0.749981	0.599988

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]:

```python
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',encoding='utf-8')
np.savetxt('train_n.txt', n, delimiter=' ', fmt='%s',encoding='utf-8')
```

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

In [0]:

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

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

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

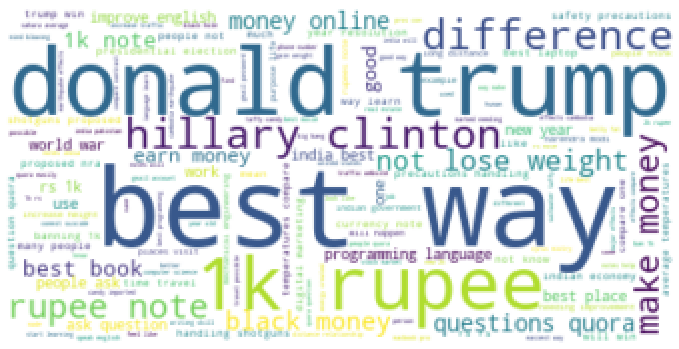
```
Total number of words in duplicate pair questions : 16109886
Total number of words in non duplicate pair questions : 33193067
```

Word Clouds generated from duplicate pair question's text

In [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]:

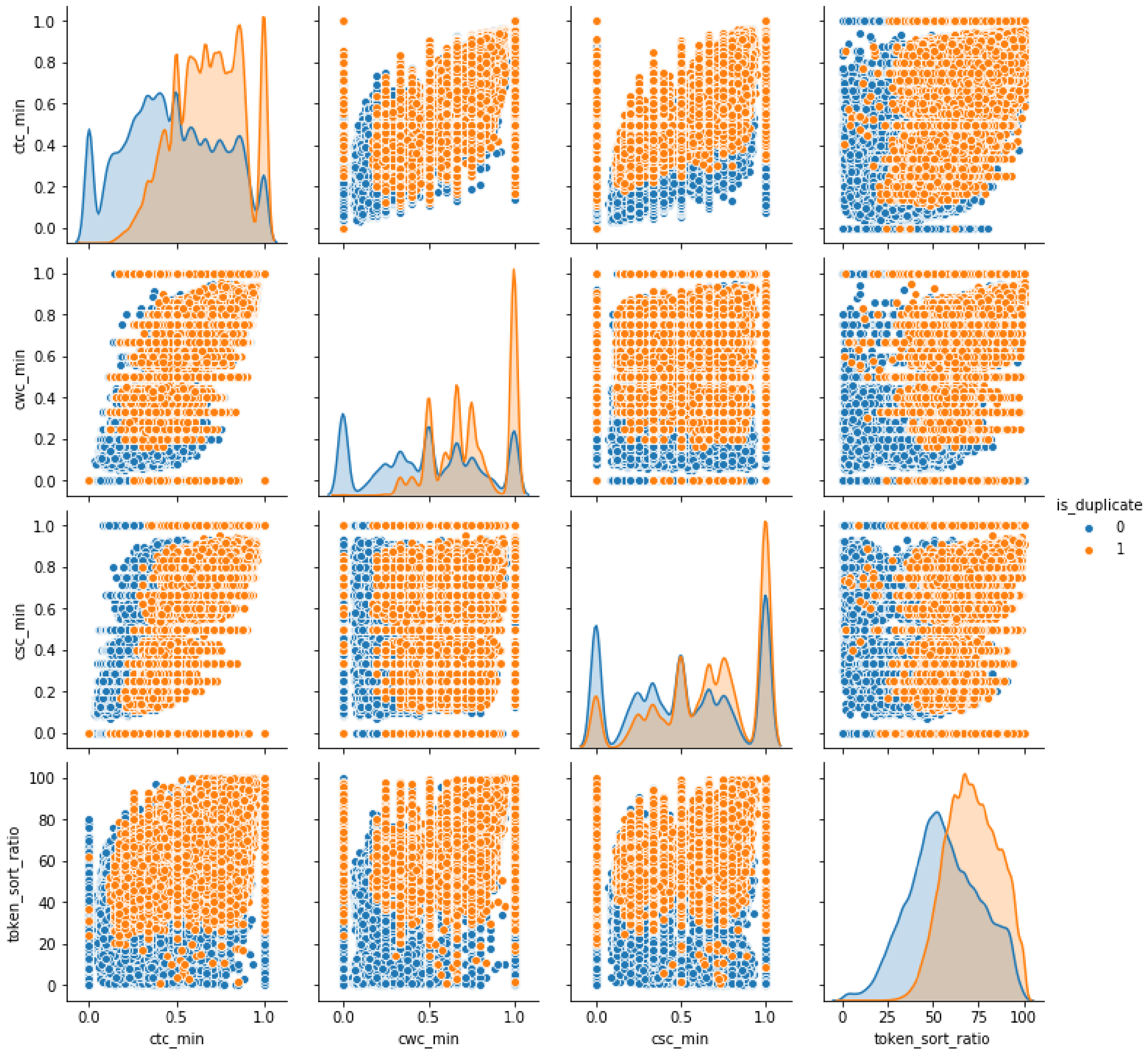
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

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

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

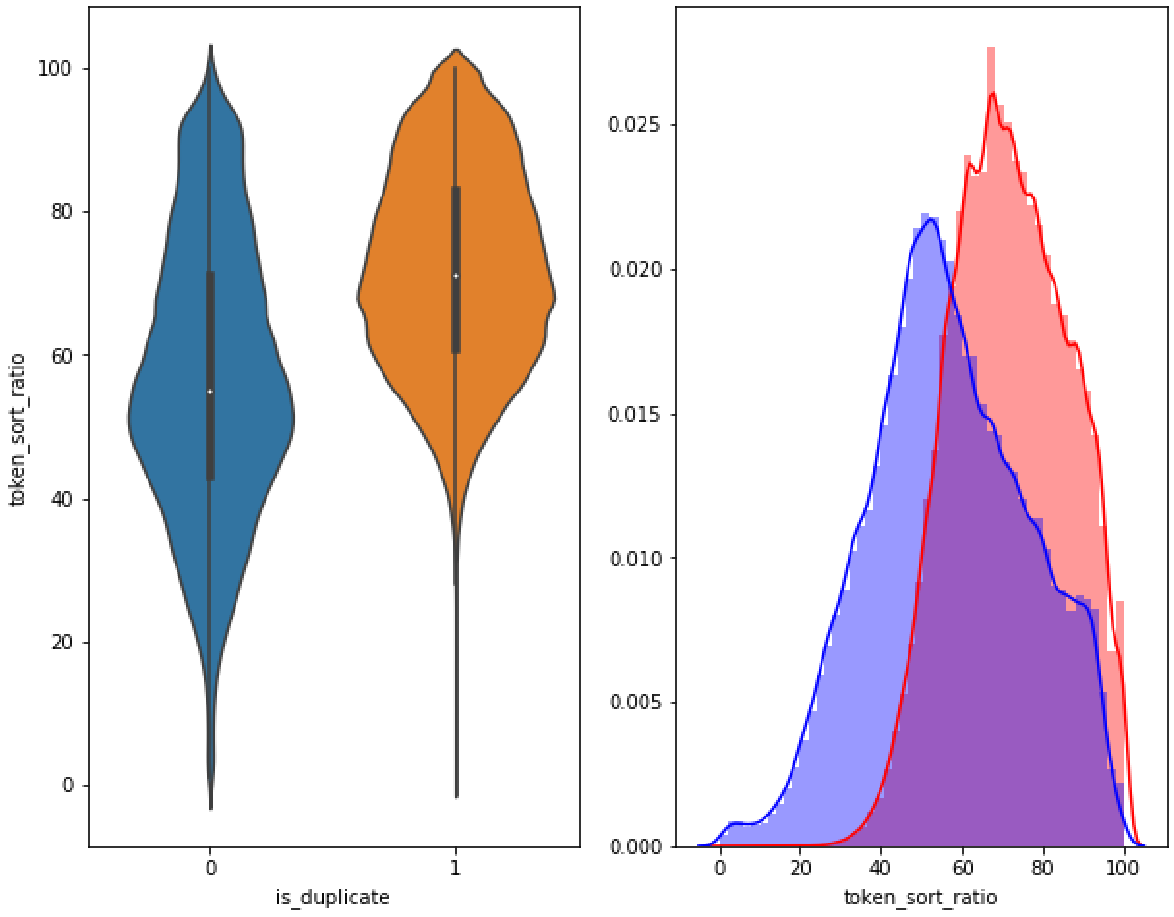
In [0]:

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



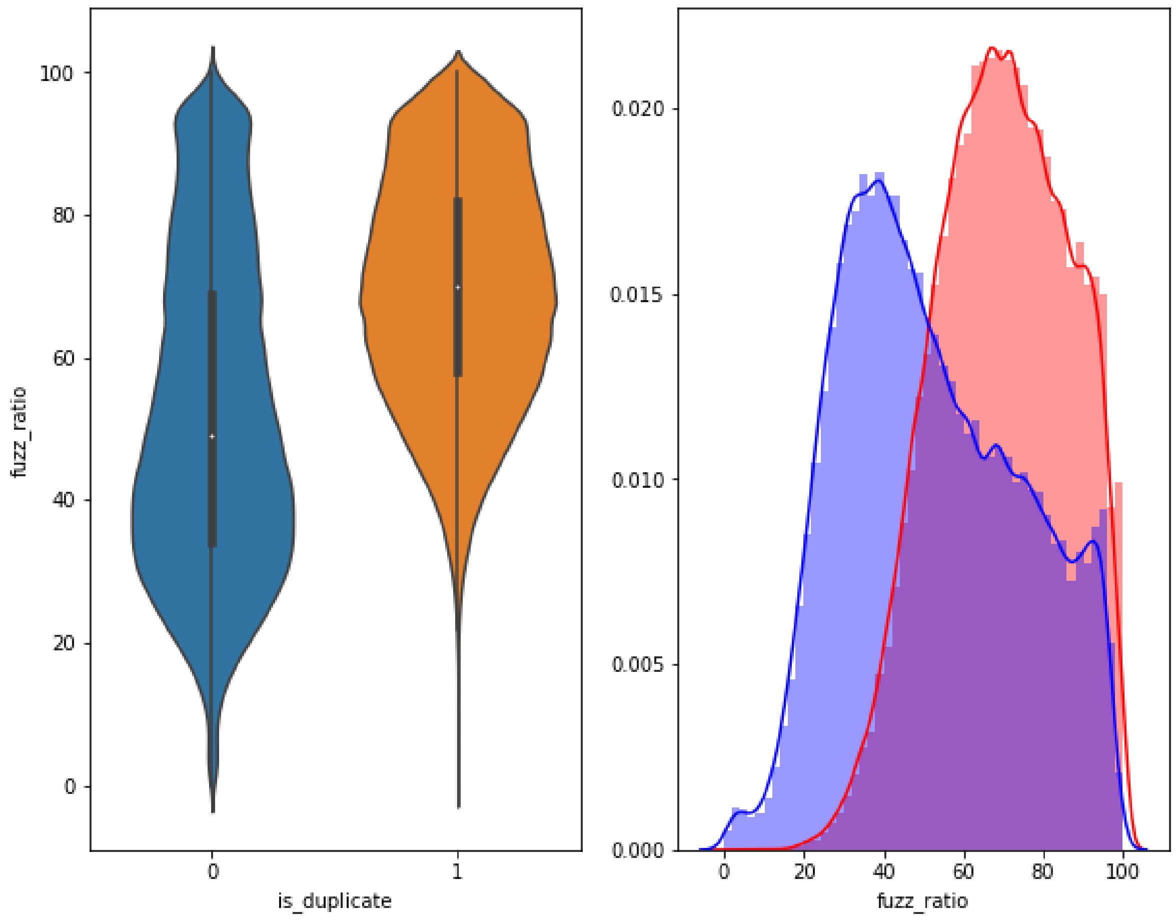
In [0]:

```
# 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 = '
sns.distplot(df[df['is_duplicate'] == 0.0]['token_sort_ratio'][0:] , label = "0" , color =
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]:

```python
# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the data

from sklearn.preprocessing import MinMaxScaler

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

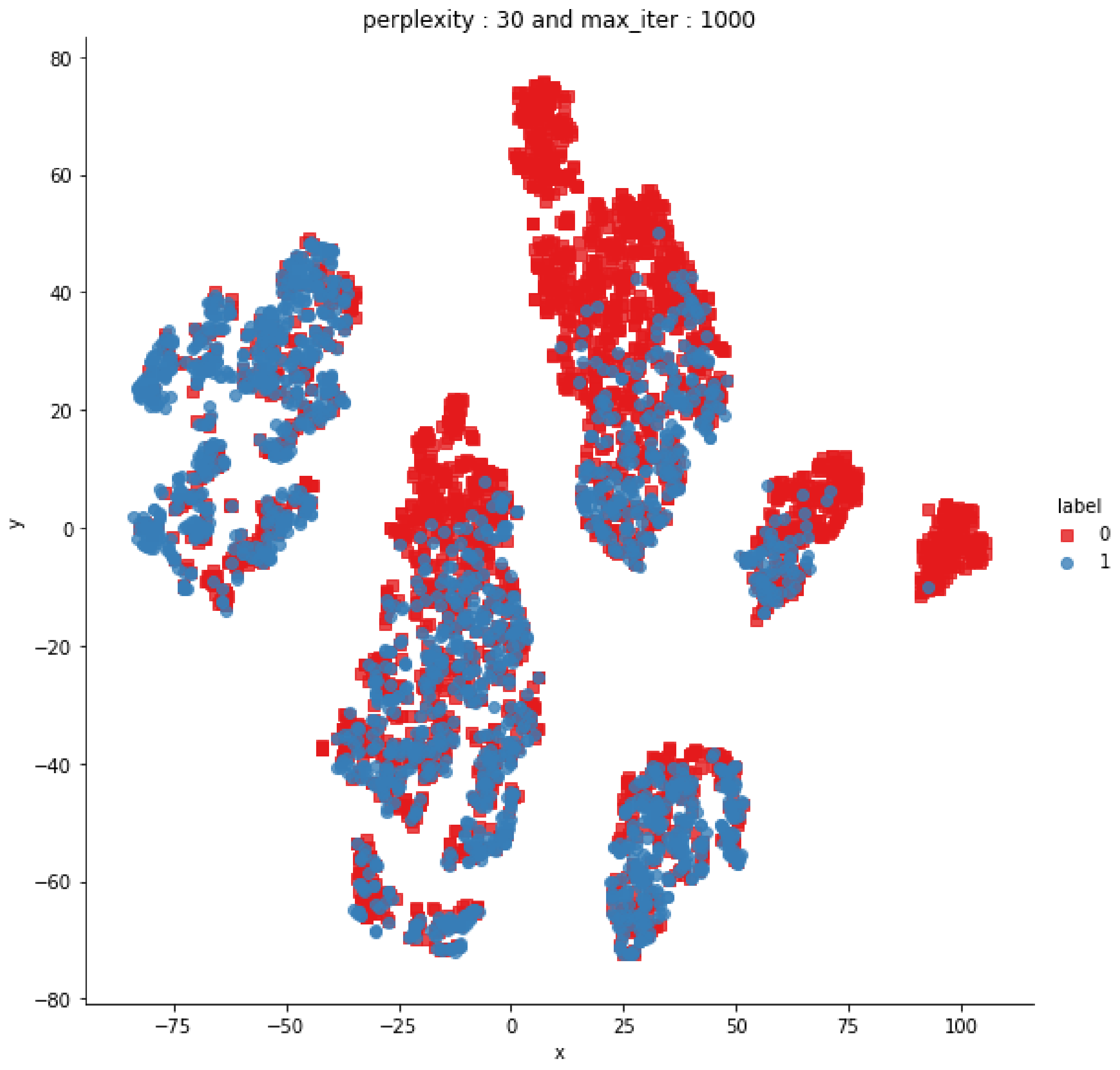
In [0]:

```python
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 0.018s...
[t-SNE] Computed neighbors for 5000 samples in 0.393s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilities in 0.311s
[t-SNE] Iteration 50: error = 81.3346405, gradient norm = 0.0466835 (50 iter
ations in 2.478s)
[t-SNE] Iteration 100: error = 70.6411362, gradient norm = 0.0087385 (50 ite
rations in 1.738s)
[t-SNE] Iteration 150: error = 68.9421158, gradient norm = 0.0055224 (50 ite
rations in 1.670s)
[t-SNE] Iteration 200: error = 68.1217880, gradient norm = 0.0044136 (50 ite
rations in 1.722s)
[t-SNE] Iteration 250: error = 67.6154175, gradient norm = 0.0040027 (50 ite
rations in 1.774s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.61541
7
[t-SNE] Iteration 300: error = 1.7931896, gradient norm = 0.0011886 (50 iter
ations in 1.832s)
[t-SNE] Iteration 350: error = 1.3933632, gradient norm = 0.0004814 (50 iter
ations in 1.814s)
[t-SNE] Iteration 400: error = 1.2277179, gradient norm = 0.0002778 (50 iter
ations in 1.795s)
[t-SNE] Iteration 450: error = 1.1382203, gradient norm = 0.0001874 (50 iter
ations in 1.825s)
[t-SNE] Iteration 500: error = 1.0834213, gradient norm = 0.0001423 (50 iter
ations in 1.809s)
[t-SNE] Iteration 550: error = 1.0472572, gradient norm = 0.0001143 (50 iter
ations in 1.803s)
[t-SNE] Iteration 600: error = 1.0229475, gradient norm = 0.0000992 (50 iter
ations in 1.811s)
[t-SNE] Iteration 650: error = 1.0064161, gradient norm = 0.0000887 (50 iter
ations in 1.813s)
[t-SNE] Iteration 700: error = 0.9950126, gradient norm = 0.0000781 (50 iter
ations in 1.876s)
[t-SNE] Iteration 750: error = 0.9863916, gradient norm = 0.0000739 (50 iter
ations in 1.846s)
[t-SNE] Iteration 800: error = 0.9797955, gradient norm = 0.0000678 (50 iter
ations in 1.861s)
[t-SNE] Iteration 850: error = 0.9741892, gradient norm = 0.0000626 (50 iter
ations in 1.864s)
[t-SNE] Iteration 900: error = 0.9692684, gradient norm = 0.0000620 (50 iter
ations in 1.860s)
[t-SNE] Iteration 950: error = 0.9652691, gradient norm = 0.0000559 (50 iter
ations in 1.878s)
[t-SNE] Iteration 1000: error = 0.9615035, gradient norm = 0.0000559 (50 ite
rations in 1.929s)
[t-SNE] KL divergence after 1000 iterations: 0.961504
```

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



In [0]:

```
from sklearn.manifold import TSNE
tsne3d = TSNE(
    n_components=3,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.024s...
[t-SNE] Computed neighbors for 5000 samples in 0.496s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilities in 0.344s
[t-SNE] Iteration 50: error = 80.5661621, gradient norm = 0.0296227 (50 iter
ations in 9.998s)
[t-SNE] Iteration 100: error = 69.4089432, gradient norm = 0.0033432 (50 ite
rations in 4.633s)
[t-SNE] Iteration 150: error = 67.9962845, gradient norm = 0.0018752 (50 ite
rations in 4.276s)
[t-SNE] Iteration 200: error = 67.4377289, gradient norm = 0.0011330 (50 ite
rations in 4.274s)
[t-SNE] Iteration 250: error = 67.1244202, gradient norm = 0.0008592 (50 ite
rations in 4.302s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.12442
0
[t-SNE] Iteration 300: error = 1.5177890, gradient norm = 0.0007072 (50 iter
ations in 5.810s)
[t-SNE] Iteration 350: error = 1.1818613, gradient norm = 0.0001967 (50 iter
ations in 7.398s)
[t-SNE] Iteration 400: error = 1.0382802, gradient norm = 0.0000992 (50 iter
ations in 7.241s)
[t-SNE] Iteration 450: error = 0.9668908, gradient norm = 0.0000785 (50 iter
ations in 7.215s)
[t-SNE] Iteration 500: error = 0.9298934, gradient norm = 0.0000514 (50 iter
ations in 7.097s)
[t-SNE] Iteration 550: error = 0.9096302, gradient norm = 0.0000429 (50 iter
ations in 6.973s)
[t-SNE] Iteration 600: error = 0.8966513, gradient norm = 0.0000378 (50 iter
ations in 7.051s)
[t-SNE] Iteration 650: error = 0.8874955, gradient norm = 0.0000321 (50 iter
ations in 7.036s)
[t-SNE] Iteration 700: error = 0.8796885, gradient norm = 0.0000325 (50 iter
ations in 7.049s)
[t-SNE] Iteration 750: error = 0.8725138, gradient norm = 0.0000287 (50 iter
ations in 7.040s)
[t-SNE] Iteration 800: error = 0.8659297, gradient norm = 0.0000291 (50 iter
ations in 6.976s)
[t-SNE] Iteration 850: error = 0.8608947, gradient norm = 0.0000276 (50 iter
ations in 7.042s)
[t-SNE] Iteration 900: error = 0.8567888, gradient norm = 0.0000279 (50 iter
ations in 7.019s)
```

```
[t-SNE] Iteration 950: error = 0.8539276, gradient norm = 0.0000273 (50 iter
ations in 6.928s)
[t-SNE] Iteration 1000: error = 0.8515787, gradient norm = 0.0000235 (50 ite
rations in 6.977s)
[t-SNE] KL divergence after 1000 iterations: 0.851579
```

In [0]:

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

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

...

In [0]:

```
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

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

In [0]:

```
# avoid decoding problems
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/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]:

```
df.head()
```

Out[43]:

	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]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)

dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
```

In [0]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
#df3 = df(['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[65]:

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

In [0]:

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

Out[66]:

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

In [0]:

```
df.columns
```

Out[67]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate'], dtyp
e='object')
```

In [0]:

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

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

In [0]:

```
df1.shape, df2.shape
```

Out[69]:

```
((404290, 17), (404290, 12))
```

In [0]:

```
df_new  = df1.merge(df2, on='id',how='left')
df_new['question1'] = df['question1']
df_new['question2'] = df['question2']
```

In [0]:

```
df_new.head()
```

Out[71]:

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

In [0]:

```
df_new.shape
```

Out[72]:

```
(404290, 30)
```

3.5.3 Random train test split(70:30)

In [0]:

```
y= df['is_duplicate'][:100000].values
y.shape
```

Out[73]:

```
(100000,)
```

In [0]:

```
X = df_new.drop(['id','is_duplicate'],axis=1)[:100000]
X.shape
```

Out[74]:

```
(100000, 28)
```

In [0]:

```
X.columns
```

Out[75]:

```
Index(['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_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'question1',
       'question2'],
      dtype='object')
```

In [0]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_spli
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify= y)
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, 28)
Number of data points in test data : (30000, 28)
```

3.6 Featurizing text data with tfidf weighted word-vectors

In [0]:

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

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

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

In [0]:

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
#import spacy
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(X_train['question1']), position = 0, leave= True):
    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)
#X_train['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 70000/70000 [10:56<00:00, 106.62it/s]
```

In [0]:

```
len(list(X_test['question1']))
```

Out[79]:

```
30000
```

In [0]:

```
print(len(vecs1))
print(len(vecs1[0]))
```

```
70000
96
```

In [0]:

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

vecs1_te = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(X_test['question1']), position = 0, leave= True):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1_te = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1_te = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1_te += vec1_te * idf
    mean_vec1_te = mean_vec1_te.mean(axis=0)
    vecs1_te.append(mean_vec1_te)
#X_test['q1_feats_m'] = list(vecs1_te)
```

```
100%|██████████| 30000/30000 [04:40<00:00, 106.95it/s]
```

In [0]:

```python
print(len(vecs1_te))
print(len(vecs1_te[0]))
```

```
30000
96
```

In [0]:

```python
vecs2 = []
for qu2 in tqdm(list(X_train['question2']), position = 0, leave= True):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), 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)
#X_train['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 70000/70000 [11:03<00:00, 105.53it/s]
```

In [0]:

```
print(len(vecs2))
print(len(vecs2[0]))
```

```
70000
96
```

In [0]:

```
vecs2_te = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu2 in tqdm(list(X_test['question2']), position = 0, leave= True):
    doc2 = nlp(qu2)
    # 384 is the number of dimensions of vectors
    mean_vec2_te = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2_te = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            idf = 0
        # compute final vec
        mean_vec2_te += vec2_te * idf
    mean_vec2_te = mean_vec2_te.mean(axis=0)
    vecs2_te.append(mean_vec2_te)
#X_test['q2_feats_m'] = list(vecs2_te)
```

```
100%|██████████| 30000/30000 [04:40<00:00, 107.13it/s]
```

In [0]:

```
print(len(vecs2_te))
print(len(vecs2_te[0]))
```

```
30000
96
```

In [0]:

```
#X_train_ques1_w2v == vecs1
#X_test_ques1_w2v == vecs1_te
#X_train_ques2_w2v == vecs2
#X_test_ques2_w2v == vecs2_te
```

In [0]:

```
X_train.columns
```

Out[87]:

```
Index(['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_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'question1',
       'question2'],
      dtype='object')
```

3.7 Featurizing text data with tfidf

In [0]:

```
#print(X_train.shape, y_train.shape)
#print(X_test.shape, y_test.shape)

#vectorizer = TfidfVectorizer(min_df=10,max_features = 5000)
tfidf.fit(X_train['question1'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_ques1_tfidf = tfidf.transform(X_train['question1'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_ques1_tfidf = tfidf.transform(X_test['question1'].values)

print("After vectorizations")
print(X_train_ques1_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_ques1_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(70000, 36759) (70000,)
(30000, 36759) (30000,)
============================================================================
========================
```

In [0]:

```
#vectorizer = TfidfVectorizer(min_df=10,max_features = 5000)
#vectorizer.fit(X_train['question1'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_ques2_tfidf = tfidf.transform(X_train['question2'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_ques2_tfidf = tfidf.transform(X_test['question2'].values)

print("After vectorizations")
print(X_train_ques2_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_ques2_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(70000, 36759) (70000,)
(30000, 36759) (30000,)
============================================================================
========================
```

In [0]:

```
X_train = X_train.drop(['question1','question2'], axis=1)
X_test = X_test.drop(['question1','question2'], axis=1)
print("Shape of train data ",X_train.shape)
print("Shape of test data ",X_test.shape)
print(X_train.columns)
```

```
Shape of train data  (70000, 26)
Shape of test data  (30000, 26)
Index(['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_qid
2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],
      dtype='object')
```

In [0]:

```
X_train_ques1_tfidf.shape
```

Out[91]:

```
(70000, 36759)
```

In [0]:

```
#merging all the features
from scipy.sparse import hstack
X_tr1 = hstack((X_train.values, X_train_ques1_tfidf, X_train_ques2_tfidf)).tocsr()
X_te1 = hstack((X_test.values, X_test_ques1_tfidf, X_test_ques2_tfidf)).tocsr()
print("Number of data points in train data :",X_tr1.shape)
print("Number of data points in test data :",X_te1.shape)
```

```
Number of data points in train data : (70000, 73544)
Number of data points in test data : (30000, 73544)
```

In [0]:

```
y_test.shape
```

Out[93]:

```
(30000,)
```

4. Machine Learning Models

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:  0.6274571428571428 Class 1:  0.3725428571428571
---------- Distribution of output variable in train data ----------
Class 0:  0.37253333333333333 Class 1:  0.37253333333333333
```

In [0]:

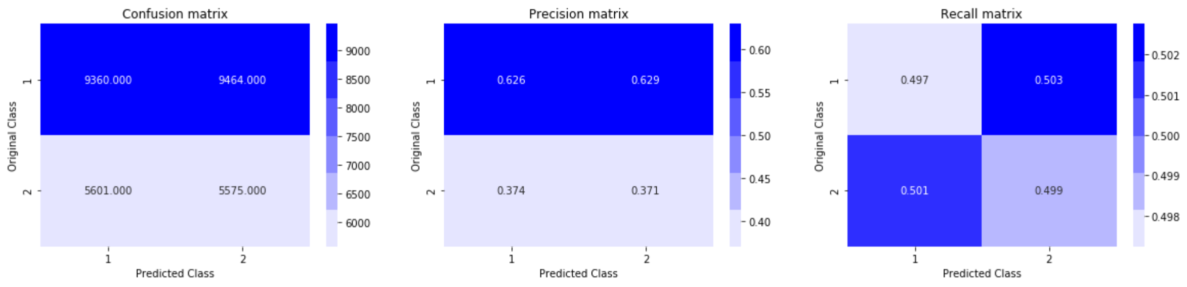
```python
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted

    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 d
    # 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 d
    # 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)
    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()
```

4.1 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 0.8896272837980698
```

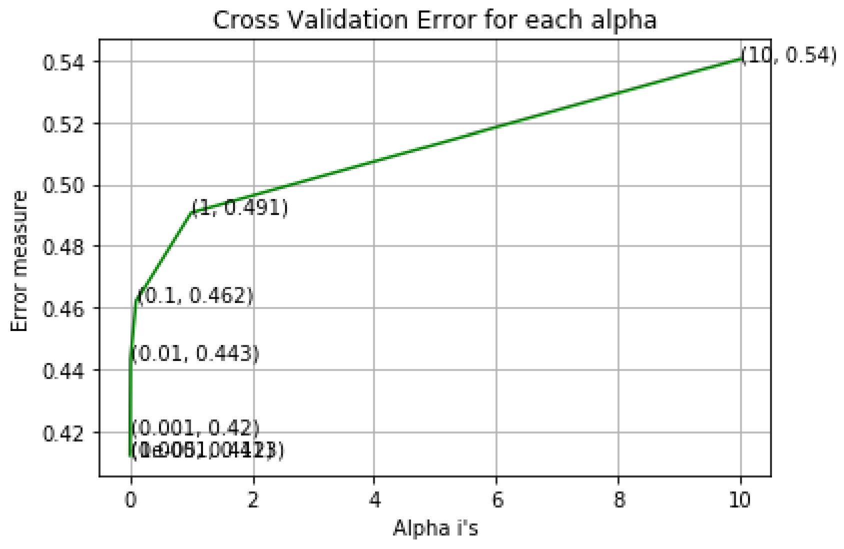


4.2 Logistic Regression with hyperparameter tuning

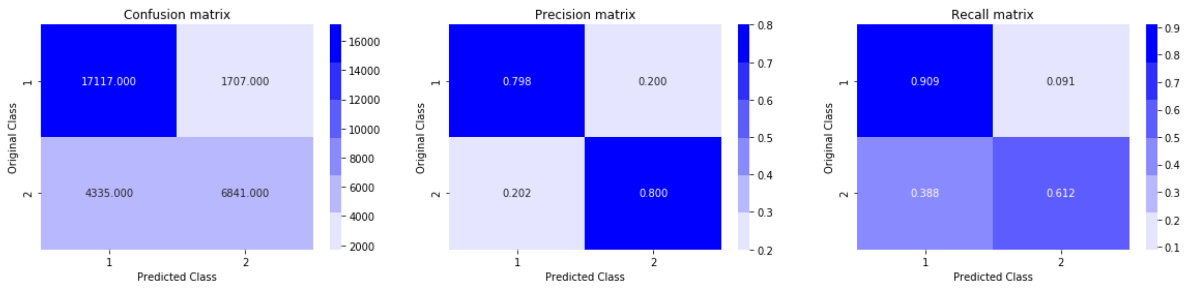
In [0]:

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_tr1, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr1, y_train)
    predict_y = sig_clf.predict_proba(X_te1)
    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, label

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(X_tr1, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr1, y_train)

predict_y = sig_clf.predict_proba(X_tr1)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(X_te1)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
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: 0.412216323077144
For values of alpha =  0.0001 The log loss is: 0.4126456320646974
For values of alpha =  0.001 The log loss is: 0.41955390929722924
For values of alpha =  0.01 The log loss is: 0.44320303044895415
For values of alpha =  0.1 The log loss is: 0.4621214348835899
For values of alpha =  1 The log loss is: 0.49074732052426856
For values of alpha =  10 The log loss is: 0.5404260233470596
```



```
For values of best alpha =  1e-05 The train log loss is: 0.40687527349032 74
For values of best alpha =  1e-05 The test log loss is: 0.412216323077144
Total number of data points : 30000
```

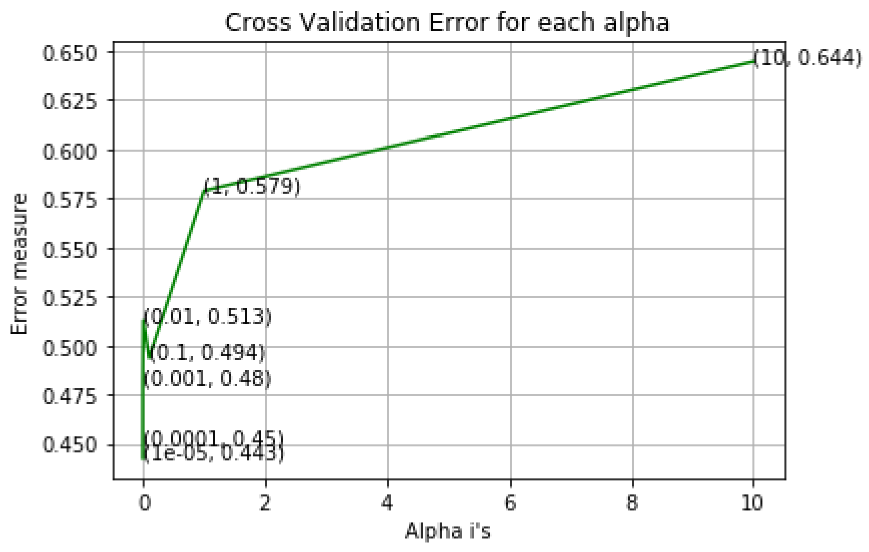


4.3 Linear SVM with hyperparameter tuning

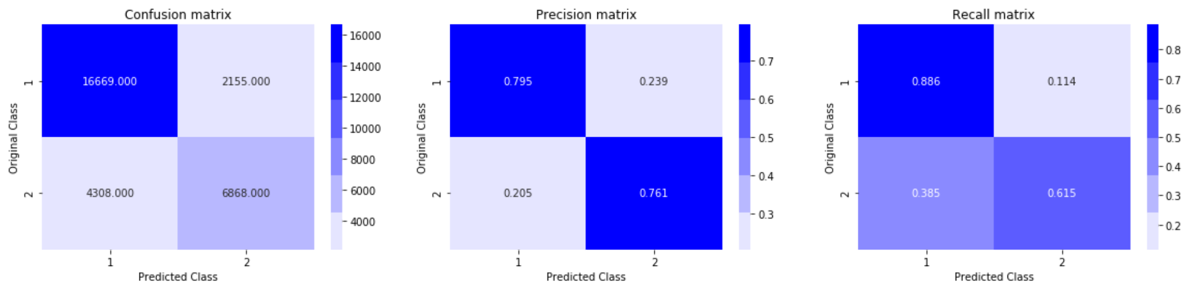
In [0]:

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

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

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_tr1, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr1, y_train)
    predict_y = sig_clf.predict_proba(X_te1)
    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, label

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(X_tr1, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr1, y_train)

predict_y = sig_clf.predict_proba(X_tr1)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y
predict_y = sig_clf.predict_proba(X_te1)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
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: 0.44258316469543035
For values of alpha =  0.0001 The log loss is: 0.44995416410880235
For values of alpha =  0.001 The log loss is: 0.48011187891696394
For values of alpha =  0.01 The log loss is: 0.512737906744285
For values of alpha =  0.1 The log loss is: 0.49351787399080976
For values of alpha =  1 The log loss is: 0.5787591668149112
For values of alpha =  10 The log loss is: 0.6443923491860919
```



```
For values of best alpha =  1e-05 The train log loss is: 0.43475745946812083
For values of best alpha =  1e-05 The test log loss is: 0.44258316469543035
Total number of data points : 30000
```



In [0]:

```
tr_vec1 = pd.DataFrame(vecs1)
tr_vec2 = pd.DataFrame(vecs2)
te_vec1 = pd.DataFrame(vecs1_te)
te_vec2 = pd.DataFrame(vecs2_te)
```

In [0]:

```
#merging all the vectorised features
X_tr2 = hstack((X_train.values, tr_vec1, tr_vec2)).tocsr()
X_te2 = hstack((X_test.values, te_vec1, te_vec2)).tocsr()
print("Number of data points in train data :",X_tr2.shape)
print("Number of data points in test data :",X_te2.shape)
```

```
Number of data points in train data : (70000, 218)
Number of data points in test data : (30000, 218)
```

4.4 XGBoost

In [0]:

```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb
clf = xgb.XGBClassifier()

params ={
    'eta': [0.1,0.2,0.3],
'max_depth' : [2,4,5,6],
'n_estimators': [50,100,150,200,300,500]
}



#d_train = xgb.DMatrix(X_tr2, label=y_train)
#d_test = xgb.DMatrix(X_te2, label=y_test)

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

rs_clf = RandomizedSearchCV(clf , param_distributions= params, cv = 2, verbose= 10, n_jobs=
result = rs_clf.fit(X_tr2,y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 11.7min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 19.0min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 31.5min
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 45.5min remaining:
0.0s
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 45.5min finished
```

In [0]:

```
result.best_params_
```

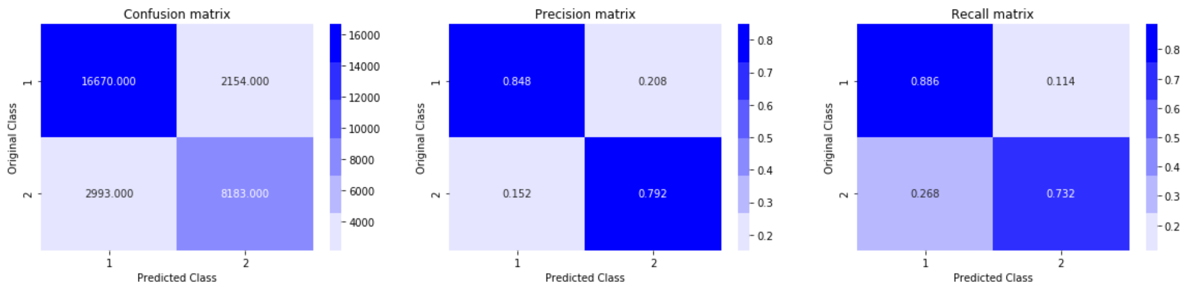
Out[107]:

```
{'eta': 0.3, 'max_depth': 5, 'n_estimators': 200}
```

In [0]:

```
clf = xgb.XGBClassifier(eta= 0.3,max_depth= 5, n_estimators = 200, subsample= 1,objective=
clf.fit(X_tr2,y_train)
predict_y = clf.predict_proba(X_tr2)
print("The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
predict_y = clf.predict_proba(X_te2)
print( "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)
```

```
The train log loss is: 0.283869614983400035
The test log loss is: 0.3394522333965986
Total number of data points : 30000
```



4.5 Conclusion

In [0]:

```python
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","Model","Hyperparameters", "Train Log Loss", "Test Loss"]

x.add_row(["TFIDF","Random", "-", "-", 0.88 ])
x.add_row(["------------------","--------------------","------------------------------","--
x.add_row(["TFIDF","Linear SVM", "alpha = 1e-05", 0.43, 0.44 ])
x.add_row(["------------------","--------------------","------------------------------","--

x.add_row(["TFIDF","Logistic Regression", "alpha = 1e-05", 0.40, 0.41])
x.add_row(["------------------","--------------------","------------------------------","--
x.add_row(["TFIDF_W2V","XGBoost","eta=0.3, max_depth=5, estimators=200", 0.28, 0.33])

print(x)
```

```
+--------------------+----------------------+-------------------------------
-------+----------------+-------------+
|     Vectorizer     |        Model         |           Hyperparameters
| Train Log Loss |  Test Loss  |
+--------------------+----------------------+-------------------------------
-------+----------------+-------------+
|       TFIDF        |        Random        |                  -
|       -        |     0.88    |
| ------------------ | -------------------- |    ---------------------------
---    |  -----------   | ----------- |
|       TFIDF        |      Linear SVM      |            alpha = 1e-05
|      0.43      |     0.44    |
| ------------------ | -------------------- |    ---------------------------
---    |  -----------   | ----------- |
|       TFIDF        | Logistic Regression  |            alpha = 1e-05
|      0.4       |     0.41    |
| ------------------ | -------------------- |    ---------------------------
---    |  -----------   | ----------- |
|     TFIDF_W2V      |       XGBoost        | eta=0.3, max_depth=5, estimato
rs=200 |      0.28      |     0.33    |
+--------------------+----------------------+-------------------------------
-------+----------------+-------------+
```