In [1]:
```
![Quora-1.png](attachment:Quora-1.png)
```

```
/bin/bash: -c: line 0: syntax error near
unexpected token `attachment:Quora-1.png'
/bin/bash: -c: line 0: `[Quora-1.png](att
achment: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 has

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

Useful Links

- Discussions : 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/A dl=0
- Blog 1 : 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

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","question
2","is_duplicate"
"0","1","2","What is the step by step gu
ide to invest in share market in indi
a?","What is the step by step guide to i
nvest in share market?","0"
"1","3","4","What is the story of Kohino
or (Koh-i-Noor) Diamond?","What would ha
ppen if the Indian government stole the
Kohinoor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geolo
gist?","What should I do to be a great g
eologist?","1"
"11","23","24","How do I read and find m
y 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.

Approch method

https://classroom.appliedcourse.com/classrooms/e69

https://github.com/krpiyush5/Quora-Question-Pair-Similarity-Problem-/blob/
master/final_update_QuoraQuestionPair.ipynb

https://www.appliedaicourse.com/

Procedure

1) First we have to perform Exploratory Data Analysis on Quora Question Pair data in which we going perform such as finding number of different questions, checking for duplicates, number of occurence of questions etc.

2) Then we will perform some feature extraction like fuzz ratio, fuzz partial ration, longest common substring etc.

3) After performing feature extraction we will apply some visualisation techniques such as pair-plot, violin plot, TSNE etc.

4) Then we going to peform tfidf-w2vec vectorizer on pair of questions dataset and then we merge each tfidf-w2vec vectors to our advanced featured vectors.

5) In the next step we will apply some machine learning algorithm such as logistic regression, support vector machines etc and found log-loss for both train and test dataset.

6) After choosing best parameters we then plot confusion matrix, precision matrix and recall matrix for each one.

7) we will perform same process for tfidf vectorizer at the end of this project

3. Exploratory Data Analysis

In [0]:

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

```
Go to this URL in a browser: https://acco
unts.google.com/o/oauth2/auth?client_id=9
47318989803-6bn6qk8qdgf4n4g3pfee6491hc0br
c4i.apps.googleusercontent.com&redirect_u
ri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&re
sponse_type=code&scope=email%20https%3a%2
```

```
f%2fwww.googleapis.com%2fauth%2fdocs.tes
t%20https%3a%2f%2fwww.googleapis.com%2fau
th%2fdrive%20https%3a%2f%2fwww.googleapi
s.com%2fauth%2fdrive.photos.readonly%20ht
tps%3a%2f%2fwww.googleapis.com%2fauth%2fp
eopleapi.readonly

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

In [0]:

```
pip install distance
```

```
Collecting distance
  Downloading https://files.pythonhosted.
org/packages/5c/1a/883e47df323437aefa0d0a
92ccfb38895d9416bd0b56262c2e46a47767b8/Di
stance-0.1.3.tar.gz (180kB)
     |████████████████████████████████| 1
84kB 3.5MB/s
Building wheels for collected packages: d
istance
  Building wheel for distance (setup.py)
... done
  Created wheel for distance: filename=Di
stance-0.1.3-cp36-none-any.whl size=16261
sha256=67572c781d95994d908ca9ff7741f419be
c1ecf37ac59920d5bf54401f2d7b50
  Stored in directory: /root/.cache/pip/w
heels/d5/aa/e1/dbba9e7b6d397d645d0f12db1c
66dbae9c5442b39b001db18e
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
```

3.1 Reading data and basic stats

In [0]:

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

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

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

In [0]:

```
df.head()
```

Out[0]:

	id	qid1	qid2	question1	question2	is

	id	qid1	qid2	question1	question2	is
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

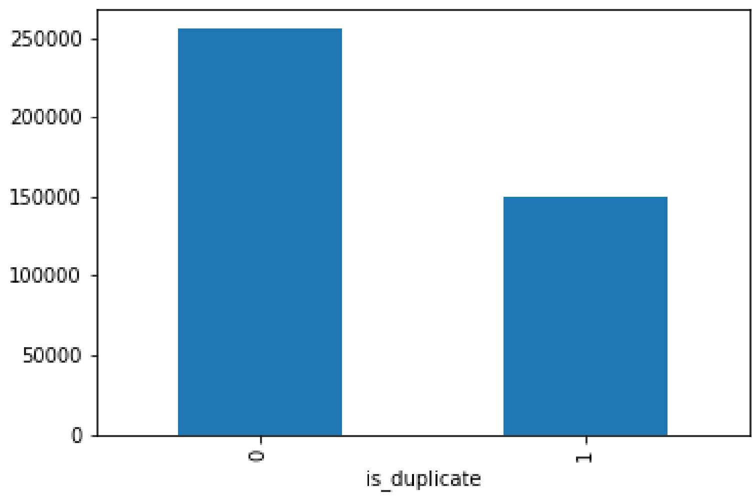
- Number of duplicate(smilar) and non-duplicate(non similar) questions

In [0]:

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

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f032fe83a20>
```



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]:

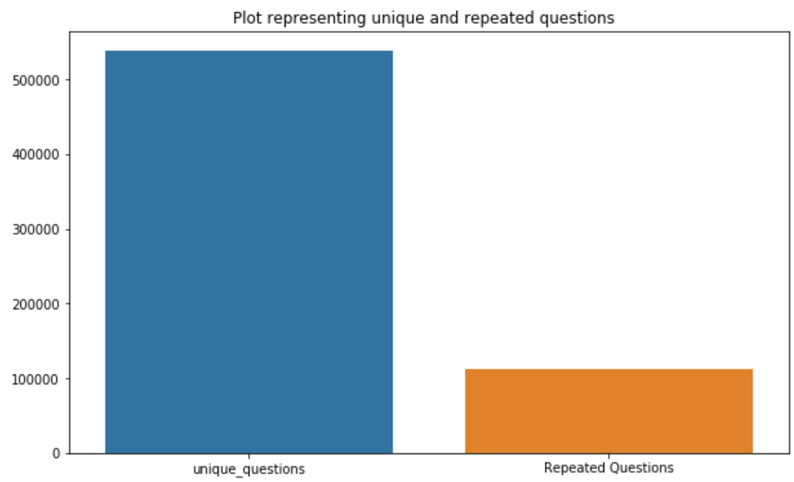
```
qids = pd.Series(df['qid1'].tolist() + df['qid2'].tolist())
unique_qs = len(np.unique(qids))
qs_morethan_onetime = np.sum(qids.value_counts() > 1)
print ('Total number of  Unique Questions are: {}\n'.format(unique_qs))
#print len(np.unique(qids))

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())))

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
```

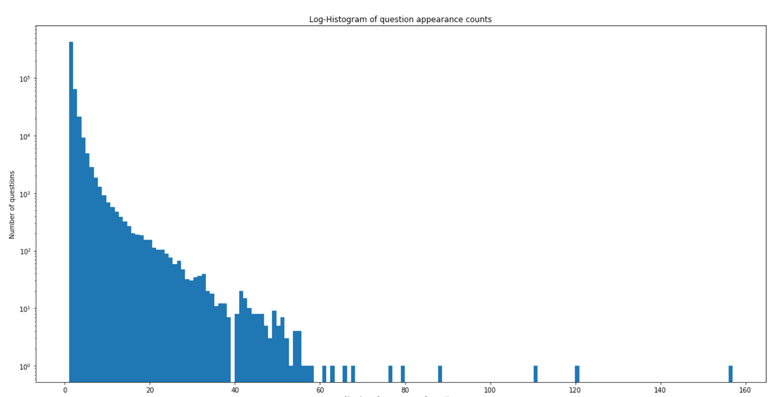
3.2.4 Number of occurrences of each

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, ques
tion2, 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_preprocessi
ng_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(la
mbda row: len(row.split(" ")))
    df['q2_n_words'] = df['question2'].apply(la
```

```
mbda 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_wor
d_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) + le
n(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_trai
```

```
n.csv", index=False)

df.head()
```

Out[0]:

	id	qid1	qid2	question1	question2	is
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

	id	qid1	qid2	question1	question2	is
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	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])
```

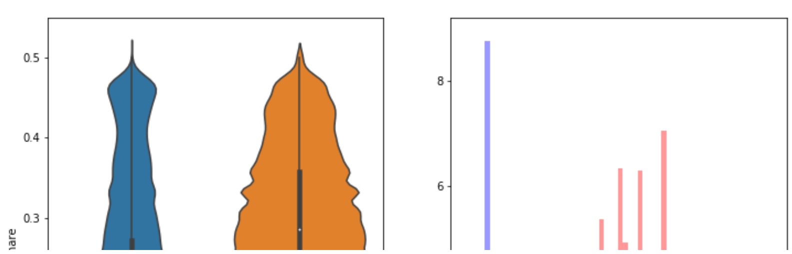
```
Minimum length of the questions in question1 :  1
Minimum length of the questions in question2 :  1
```

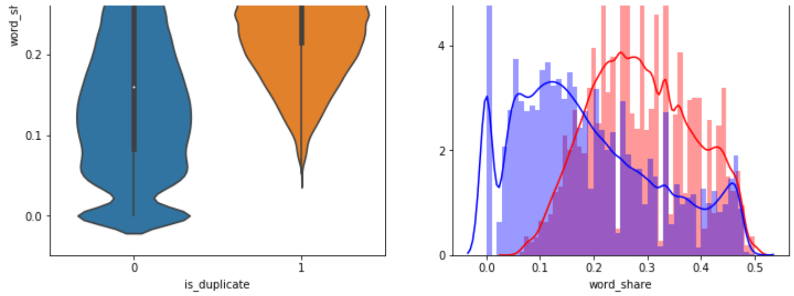
```
Number of Questions with minimum length
[question1] : 67
Number of Questions with minimum length
[question2] : 24
```

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_sh
are', data = df[0:])

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



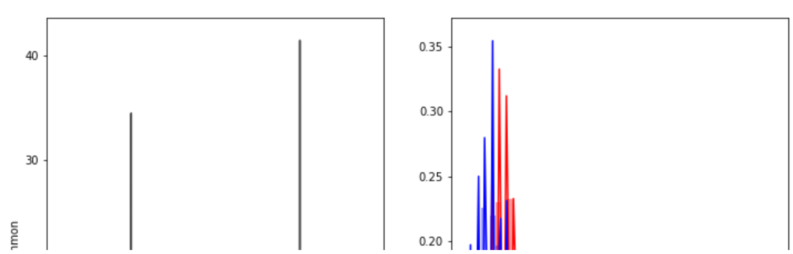


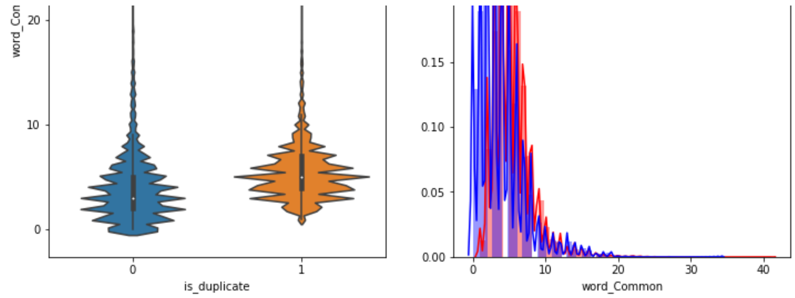
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

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

1.2.1 : EDA: Advanced Feature Extraction.

In [0]:

```
pip install fuzzywuzzy
```

```
Collecting fuzzywuzzy
  Downloading https://files.pythonhosted.
org/packages/d8/f1/5a267addb30ab7eaa1beab
2b9323073815da4551076554ecc890a3595ec9/fu
zzywuzzy-0.17.0-py2.py3-none-any.whl
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.17.0
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from subprocess import check_output
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
```

```
import plotly.graph_objs as go
import plotly.tools as tls
import os
import gc

import re
from nltk.corpus import stopwords
import distance
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
from nltk.corpus import stopwords
# This package is used for finding longest comm
on 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-Clo
ud 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/u
nicodedecodeerror-utf8-codec-cant-decode-byte-0
x9c
if os.path.isfile('df_fe_without_preprocessing_
train.csv'):
    df = pd.read_csv("df_fe_without_preprocessi
ng_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 notebook")
```

In [0]:

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

Out[0]:

	id	qid1	qid2	question1	question2	is_(
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

3.4 Preprocessing of Text

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

In [0]:
```python
import nltk
nltk.download('stopwords')
```

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

Out[0]:
```
True
```

In [0]:
```python
# 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(",00
0", "k").replace("′", "'").replace("’", "'")\
                           .replace("won't", "w
ill not").replace("cannot", "can not").replace(
"can't", "can not")\
                           .replace("n't", " no
t").replace("what's", "what is").replace("it's"
, "it is")\
                           .replace("'ve", " ha
ve").replace("i'm", "i am").replace("'re", " ar
e")\
                           .replace("he's", "he
is").replace("she's", "she is").replace("'s", "
own")\
                           .replace("%", " perc
ent ").replace("₹", " rupee ").replace("$", " d
ollar ")\
                           .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 Questio
n pair
    common_word_count = len(q1_words.intersecti
on(q2_words))

    # Get the common stopwords from Question pa
ir
    common_stop_count = len(q1_stops.intersecti
on(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_tok
en_features(x["question1"], x["question2"]), ax
is=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.seatg
eek.com/fuzzywuzzy-fuzzy-string-matching-in-pyt
hon/
    # https://stackoverflow.com/questions/31806
695/when-to-use-which-fuzz-function-to-compare-
2-strings
    # https://github.com/seatgeek/fuzzywuzzy
    print("fuzzy features..")

    df["token_set_ratio"]       = df.apply(lamb
da x: fuzz.token_set_ratio(x["question1"], x["q
uestion2"]), axis=1)
    # The token sort approach involves tokenizi
ng the string in question, sorting the tokens a
lphabetically, and
    # then joining them back into a string We t
hen compare the transformed strings with a simp
le ratio().
    df["token_sort_ratio"]      = df.apply(lamb
da x: fuzz.token_sort_ratio(x["question1"], x[
"question2"]), axis=1)
    df["fuzz_ratio"]            = df.apply(lamb
da x: fuzz.QRatio(x["question1"], x["question2"
```

```
]), axis=1)
    df["fuzz_partial_ratio"]     = df.apply(lamb
da x: fuzz.partial_ratio(x["question1"], x["que
stion2"]), axis=1)
    df["longest_substr_ratio"]  = df.apply(lamb
da x: get_longest_substr_ratio(x["question1"],
x["question2"]), axis=1)
    return df
```

In [0]:

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

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

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_du
plicate["question2"]]).flatten()
n = np.dstack([dfp_nonduplicate["question1"], d
fp_nonduplicate["question2"]]).flatten()

print ("Number of data points in class 1 (dupli
cate pairs) :",len(p))
print ("Number of data points in class 0 (non d
uplicate 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 (duplica
te pairs) : 298526
Number of data points in class 0 (non dup
licate 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')).rea
d()
textn_w = open(path.join(d, 'train_n.txt')).rea
d()
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))
```

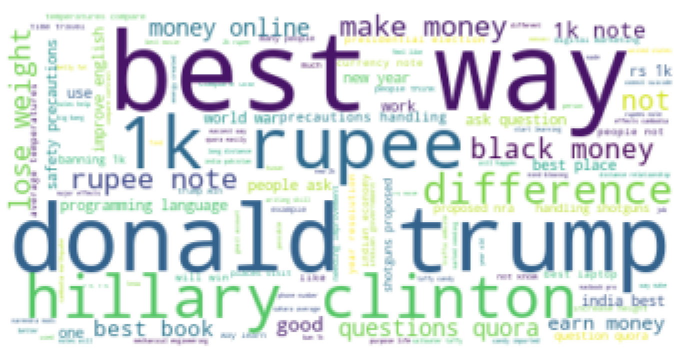
```
Total number of words in duplicate pair q
uestions : 16109886
Total number of words in non duplicate pa
ir questions : 33193067
```

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]:

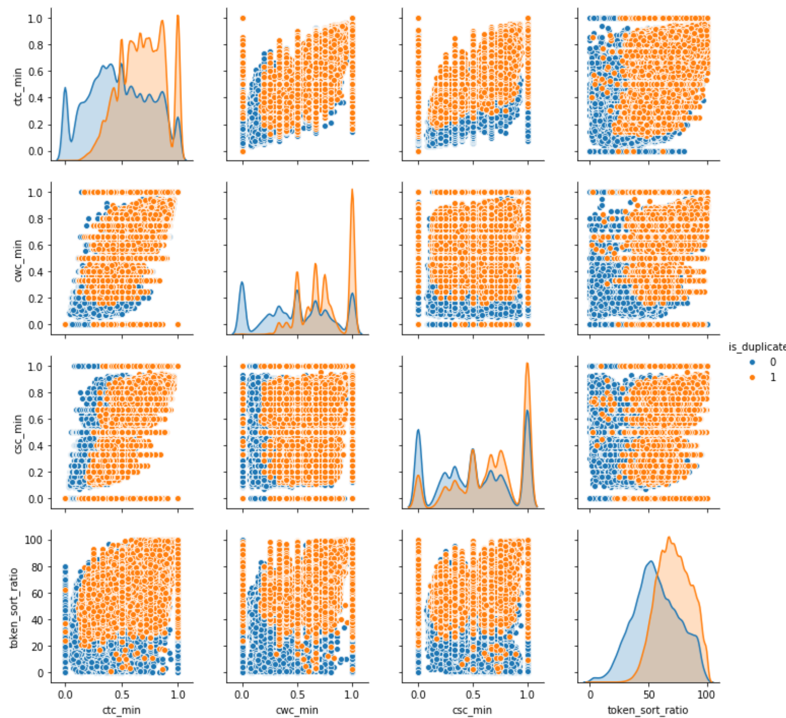
```
wc = WordCloud(background_color="white", max_words=len(textn_w),stopwords=stopwords)
# generate word cloud
wc.generate(textn_w)
print ("Word Cloud for non-Duplicate Question pairs:")
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
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]:

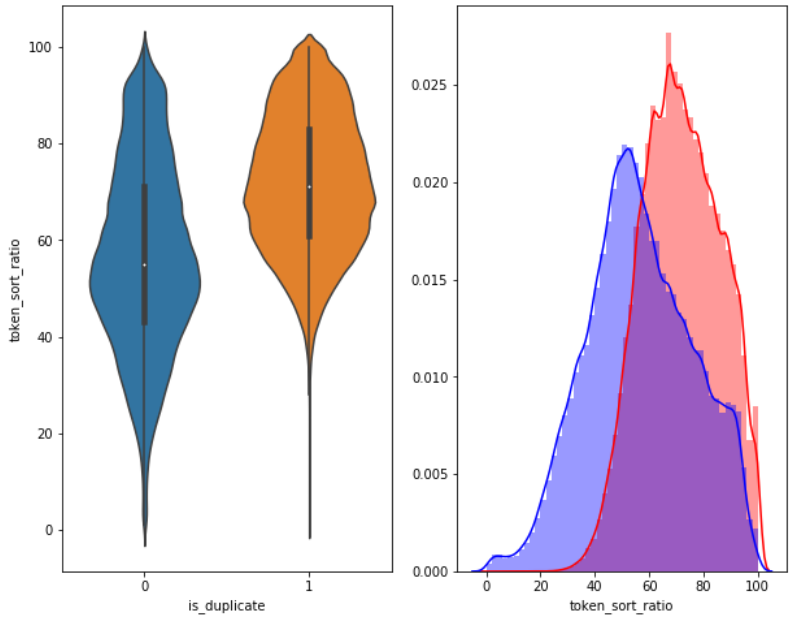
```
n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_mi
n', 'token_sort_ratio', 'is_duplicate']][0:n],
hue='is_duplicate', vars=['ctc_min', 'cwc_min',
'csc_min', 'token_sort_ratio'])
plt.show()
```



In [0]:

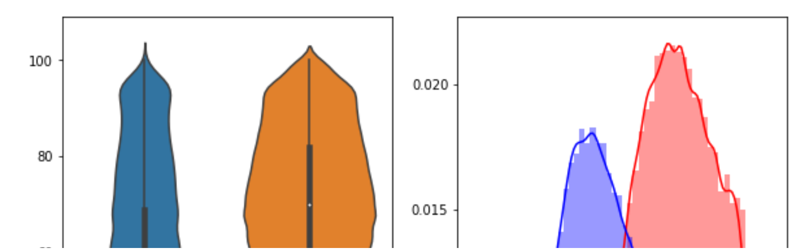
```
# Distribution of the token_sort_ratio
plt.figure(figsize=(10, 8))

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

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate'] == 1.0]['tok
en_sort_ratio'][0:] , label = "1", color = 're
d')
sns.distplot(df[df['is_duplicate'] == 0.0]['tok
en_sort_ratio'][0:] , label = "0" , color = 'bl
```

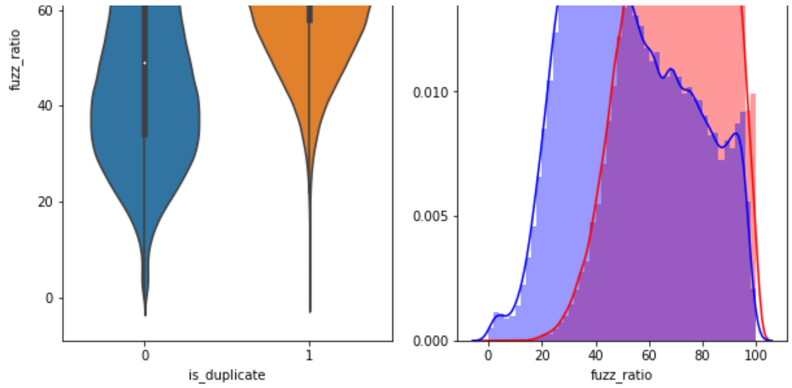
```
ue' )
plt.show()
```



In [0]:

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

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

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

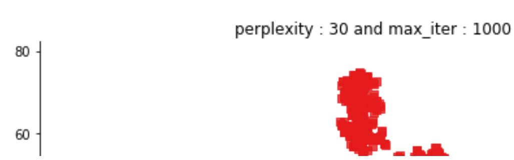
```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5000 samples in 0.015s...
[t-SNE] Computed neighbors for 5000 sampl
es in 0.382s...
[t-SNE] Computed conditional probabilitie
s for sample 1000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 2000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 3000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 4000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilitie
s in 0.300s
[t-SNE] Iteration 50: error = 81.2911148,
gradient norm = 0.0457501 (50 iterations
in 2.775s)
[t-SNE] Iteration 100: error = 70.604415
9, gradient norm = 0.0086692 (50 iteratio
ns in 1.918s)
[t-SNE] Iteration 150: error = 68.912490
8, gradient norm = 0.0056016 (50 iteratio
ns in 1.798s)
[t-SNE] Iteration 200: error = 68.101074
2, gradient norm = 0.0047585 (50 iteratio
ns in 1.872s)
[t-SNE] Iteration 250: error = 67.590797
4, gradient norm = 0.0033576 (50 iteratio
ns in 1.950s)
[t-SNE] KL divergence after 250 iteration
```

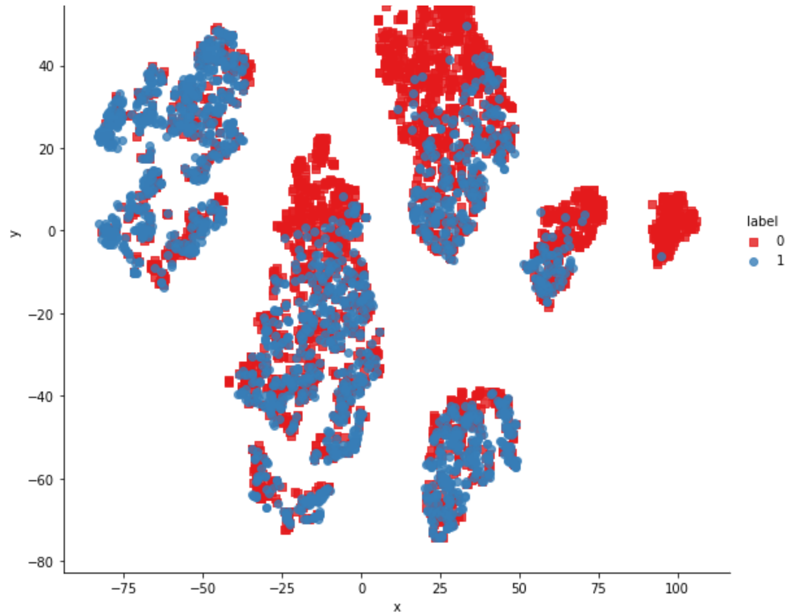
```
s with early exaggeration: 67.590797
[t-SNE] Iteration 300: error = 1.7929677,
gradient norm = 0.0011899 (50 iterations
in 2.001s)
[t-SNE] Iteration 350: error = 1.3937442,
gradient norm = 0.0004817 (50 iterations
in 1.933s)
[t-SNE] Iteration 400: error = 1.2280033,
gradient norm = 0.0002773 (50 iterations
in 1.936s)
[t-SNE] Iteration 450: error = 1.1383208,
gradient norm = 0.0001865 (50 iterations
in 1.960s)
[t-SNE] Iteration 500: error = 1.0834006,
gradient norm = 0.0001423 (50 iterations
in 1.960s)
[t-SNE] Iteration 550: error = 1.0474092,
gradient norm = 0.0001144 (50 iterations
in 1.971s)
[t-SNE] Iteration 600: error = 1.0231259,
gradient norm = 0.0000995 (50 iterations
in 1.985s)
[t-SNE] Iteration 650: error = 1.0066353,
gradient norm = 0.0000895 (50 iterations
in 1.996s)
[t-SNE] Iteration 700: error = 0.9954656,
gradient norm = 0.0000805 (50 iterations
in 2.012s)
[t-SNE] Iteration 750: error = 0.9871529,
gradient norm = 0.0000719 (50 iterations
in 2.035s)
[t-SNE] Iteration 800: error = 0.9801921,
gradient norm = 0.0000657 (50 iterations
in 2.043s)
[t-SNE] Iteration 850: error = 0.9743395,
gradient norm = 0.0000631 (50 iterations
```

```
in 2.041s)
[t-SNE] Iteration 900: error = 0.9693972,
gradient norm = 0.0000606 (50 iterations
in 2.025s)
[t-SNE] Iteration 950: error = 0.9654404,
gradient norm = 0.0000594 (50 iterations
in 2.049s)
[t-SNE] Iteration 1000: error = 0.962230
2, gradient norm = 0.0000565 (50 iteratio
ns in 2.037s)
[t-SNE] KL divergence after 1000 iteratio
ns: 0.962230
```

In [0]:

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

# draw the plot in appropriate place in the gri
d
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 : {}".f
ormat(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 0.008s...
[t-SNE] Computed neighbors for 5000 sampl
es in 0.376s...
[t-SNE] Computed conditional probabilitie
s for sample 1000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 2000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 3000 / 5000
[t-SNE] Computed conditional probabilitie
```

```
s for sample 4000 / 5000
[t-SNE] Computed conditional probabilitie
s for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilitie
s in 0.285s
[t-SNE] Iteration 50: error = 80.5316772,
gradient norm = 0.0296611 (50 iterations
in 12.905s)
[t-SNE] Iteration 100: error = 69.382316
6, gradient norm = 0.0032796 (50 iteratio
ns in 6.122s)
[t-SNE] Iteration 150: error = 67.972602
8, gradient norm = 0.0016793 (50 iteratio
ns in 5.572s)
[t-SNE] Iteration 200: error = 67.417617
8, gradient norm = 0.0010922 (50 iteratio
ns in 5.594s)
[t-SNE] Iteration 250: error = 67.103363
0, gradient norm = 0.0008839 (50 iteratio
ns in 5.561s)
[t-SNE] KL divergence after 250 iteration
s with early exaggeration: 67.103363
[t-SNE] Iteration 300: error = 1.5262967,
gradient norm = 0.0007234 (50 iterations
in 7.581s)
[t-SNE] Iteration 350: error = 1.1826925,
gradient norm = 0.0002056 (50 iterations
in 9.884s)
[t-SNE] Iteration 400: error = 1.0364963,
gradient norm = 0.0000999 (50 iterations
in 9.503s)
[t-SNE] Iteration 450: error = 0.9654390,
gradient norm = 0.0000914 (50 iterations
in 9.483s)
[t-SNE] Iteration 500: error = 0.9289201,
```

```
gradient norm = 0.0000634 (50 iterations
in 9.358s)
[t-SNE] Iteration 550: error = 0.9090494,
gradient norm = 0.0000504 (50 iterations
in 9.167s)
[t-SNE] Iteration 600: error = 0.8954713,
gradient norm = 0.0000525 (50 iterations
in 8.841s)
[t-SNE] Iteration 650: error = 0.8866501,
gradient norm = 0.0000497 (50 iterations
in 8.900s)
[t-SNE] Iteration 700: error = 0.8820391,
gradient norm = 0.0000369 (50 iterations
in 9.120s)
[t-SNE] Iteration 750: error = 0.8775222,
gradient norm = 0.0000342 (50 iterations
in 9.106s)
[t-SNE] Iteration 800: error = 0.8723416,
gradient norm = 0.0000288 (50 iterations
in 9.160s)
[t-SNE] Iteration 850: error = 0.8663230,
gradient norm = 0.0000297 (50 iterations
in 9.217s)
[t-SNE] Iteration 900: error = 0.8605922,
gradient norm = 0.0000286 (50 iterations
in 9.248s)
[t-SNE] Iteration 950: error = 0.8555549,
gradient norm = 0.0000312 (50 iterations
in 9.236s)
[t-SNE] Iteration 1000: error = 0.852174
5, gradient norm = 0.0000278 (50 iteratio
ns in 9.152s)
[t-SNE] KL divergence after 1000 iteratio
ns: 0.852175
```

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 em
bedding with engineered features')
fig=dict(data=data, layout=layout)
py.iplot(fig, filename='3DBubble')
```

3.6 Featurizing text data with tfidf weighted word-vectors

In [0]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import Cou
```

```
ntVectorizer
from sklearn.feature_extraction.text import Tfi
dfVectorizer
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/172
1
# http://landinghub.visualstudio.com/visual-cpp
-build-tools
import spacy
```

In [0]:

```
# avoid decoding problems
df = pd.read_csv("/content/drive/My Drive/Quor
a/train.csv")

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ------------------
---
# df['question1'] = df['question1'].apply(lambd
a x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambd
a 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[0]:

	id	qid1	qid2	question1	question2	is
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]:

```
from sklearn.feature_extraction.text import Tfi
dfVectorizer
from sklearn.feature_extraction.text import Cou
ntVectorizer
# merge texts
questions = list(df['question1']) + list(df['qu
estion2'])

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]:

```
# en_vectors_web_lg, which includes over 1 mill
ion 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 vector
```

```
s
    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 [59:15<00:
00, 113.71it/s]
```

In [0]:

```
# en_vectors_web_lg, which includes over 1 mill
ion unique vectors.
nlp = spacy.load('en_core_web_sm')
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 [1:00:11<0
0:00, 111.95it/s]
```

In [0]:

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

if os.path.isfile('df_fe_without_preprocessing_
train.csv'):
    dfppro = pd.read_csv("/content/drive/My Dri
ve/Quora/df_fe_without_preprocessing_train.csv"
,encoding='latin-1')
else:
    print("download df_fe_without_preprocessing
_train.csv from drive or run previous notebook"
)
```

In [0]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','qu
estion2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','q
uestion2','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
0	0	0	0.999980	0.833319	0.99
1	1	0	0.799984	0.399996	0.74
2	2	0	0.399992	0.333328	0.39
3	3	0	0.000000	0.000000	0.00
4	4	0	0.399992	0.199998	0.99

In [0]:

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

Out[0]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_
0	0	1	1	66	57	14
1	1	4	1	51	88	8
2	2	1	1	73	59	14
3	3	1	1	50	65	11
4	4	3	1	76	39	13

In [0]:

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

Out[0]:

	0	1	2	
0	211.129864	-144.683059	-68.811247	-1
1	144.124685	-114.012484	-111.716694	-1
2	81.757898	-142.184507	0.559867	-1
3	-126.651922	-59.747160	-67.763201	-1
4	299.444044	-188.632001	-22.946291	-2

5 rows × 96 columns

In [0]:

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

Out[0]:

	0	1	2	
0	151.268526	-127.013168	-31.546286	-1
1	152.023095	-44.955390	-103.559249	-1
2	4.930220	-29.029581	-117.808812	-9
3	-6.951929	-44.951731	-17.343082	-6
4	96.174524	-71.613948	21.584882	-9

5 rows × 96 columns

In [0]:

```python
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  dat
```

```
aframe :", df3_q1.shape[1])
print("Number of features in question2 w2v  dat
aframe :", 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 datafr
ame : 12
Number of features in question1 w2v  data
frame : 96
Number of features in question2 w2v  data
frame : 96
Number of features in final dataframe  :
221
```

In [0]:

```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1['id']=df1['id']
    df3_q2['id']=df1['id']
    df1  = df1.merge(df2, on='id',how='left')
    df2  = df3_q1.merge(df3_q2, on='id',how='le
ft')
    result  = df1.merge(df2, on='id',how='left'
)
    result.to_csv('final_features.csv')
```

In [0]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database
```

```
connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import Cou
ntVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassif
ier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accu
racy_score, log_loss
from sklearn.feature_extraction.text import Tfi
dfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassif
ier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedK
Fold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClass
ifierCV
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 GridSearchC
V
```

```
import math
from sklearn.metrics import normalized_mutual_i
nfo_score
from sklearn.ensemble import RandomForestClassi
fier



from sklearn.model_selection import cross_val_s
core
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifi
er

from sklearn import model_selection
from sklearn.linear_model import LogisticRegres
sion
from sklearn.metrics import precision_recall_cu
rve, auc, roc_curve
```

```
/usr/local/lib/python3.6/dist-packages/sk
learn/externals/six.py:31: DeprecationWar
ning: The module is deprecated in version
0.21 and will be removed in version 0.23
since we've dropped support for Python 2.
7. Please rely on the official version of
six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", Dep
recationWarning)
```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

In [0]:

```
#Creating db file from csv
if not os.path.isfile('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('/content/drive/My Drive/Quora/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_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','8
```

```
9_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','10
2_x','103_x','104_x','105_x','106_x','107_x','1
08_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','13
7_x','138_x','139_x','140_x','141_x','142_x','1
43_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','17
2_x','173_x','174_x','175_x','176_x','177_x','1
78_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','20
7_x','208_x','209_x','210_x','211_x','212_x','2
13_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','24
2_x','243_x','244_x','245_x','246_x','247_x','2
48_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','27
7_x','278_x','279_x','280_x','281_x','282_x','2
83_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','31
2_x','313_x','314_x','315_x','316_x','317_x','3
18_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','34
7_x','348_x','349_x','350_x','351_x','352_x','3
53_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','38
2_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','2
6_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','4
6_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','6
6_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','8
6_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','11
1_y','112_y','113_y','114_y','115_y','116_y','1
17_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','14
6_y','147_y','148_y','149_y','150_y','151_y','1
52_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','18
1_y','182_y','183_y','184_y','185_y','186_y','1
87_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','21
6_y','217_y','218_y','219_y','220_y','221_y','2
22_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','25
1_y','252_y','253_y','254_y','255_y','256_y','2
57_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','28
6_y','287_y','288_y','289_y','290_y','291_y','2
92_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','319_y','320_y','32
1_y','322_y','323_y','324_y','325_y','326_y','3
27_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','35
6_y','357_y','358_y','359_y','360_y','361_y','3
62_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_exist
s='append')
        index_start = df.index[-1] + 1
```

```
180000 rows
360000 rows
540000 rows
```

In [0]:

```
#http://www.sqlitetutorial.net/sqlite-python/cr
eate-tables/
def create_connection(db_file):
    """ create a database connection to the SQL
ite 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]:

```
read_db = '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_dupli
cate'], axis=1, inplace=True)
```

In [0]:

```
data.head()
```

Out[0]:

	cwc_min	cwc_max
1	0.333327777870369	0.222219753113854
2	0.285710204139941	0.249996875039062
3	0.199996000079998	0.14285510206997
4	0.999975000624984	0.799984000319994
5	0.0	0.0

5 rows × 794 columns

4.2 Converting strings to numerics

In [0]:

```
# after we read from sql table each entry was r
ead it as a string
# we convert all the features into numaric befo
re 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
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
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]:
```
# after we read from sql table each entry was r
ead it as a string
# we convert all the features into numaric befo
re we apply any model
cols = list(data.columns)
data = pd.DataFrame(np.array(data.values,dtype=
np.float64),columns=cols)
```

In [0]:
```
# https://stackoverflow.com/questions/7368789/c
onvert-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_sp
lit(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 : (70
000, 794)
Number of data points in test data : (300
00, 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 variabl
e in train data ----------
Class 0:  0.6319 Class 1:  0.3681
---------- Distribution of output variabl
e in train data ----------
Class 0:  0.3681 Class 1:  0.3681
```

In [0]:

```
# This function plots the confusion matrices gi
ven 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) represent
s number of points of class i are predicted cla
ss 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 col
umns and axis=1 corresponds to rows in two diam
ensional 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)
    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()
```

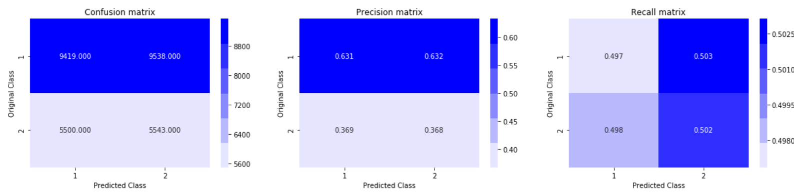
4.4 Building a random model (Finding worst-case log-loss)

In [0]:

```
# we need to generate 9 numbers and the sum of
 numbers should be 1
# one solution is to genarate 9 numbers and div
ide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/408
4039
# we create a output array that has exactly sam
e 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 Mode
l",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
```

```
0.8884007769125581
```



4.4 Logistic Regression with hyperparameter tuning

In [0]:

```
nan_rows = X_train[X_train.isnull().any(1)]
print (nan_rows)
```

```
Empty DataFrame
Columns: [cwc_min, cwc_max, csc_min, csc_
max, ctc_min, ctc_max, last_word_eq, firs
t_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_Tota
l, 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, 2
9_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, ...]

Index: []
```

```
[0 rows x 794 columns]
```

In [0]:

```python
nan_rows = X_train[X_train.isnull().any(1)]
print (nan_rows)
```

```
Empty DataFrame
Columns: [cwc_min, cwc_max, csc_min, csc_
max, ctc_min, ctc_max, last_word_eq, firs
t_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_Tota
l, 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, 2
9_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, ...]
Index: []

[0 rows x 794 columns]
```

In [0]:

```python
# Filling the null values with ' '
X_train = X_train.fillna('')
nan_rows = X_train[X_train.isnull().any(1)]
print (nan_rows)
```

```
Empty DataFrame
```

```
Columns: [cwc_min, cwc_max, csc_min, csc_
max, ctc_min, ctc_max, last_word_eq, firs
t_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_Tota
l, 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, 2
9_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, ...]
Index: []

[0 rows x 794 columns]
```

In [0]:

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

# read more about SGDClassifier() at http://sci
kit-learn.org/stable/modules/generated/sklearn.
linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alp
ha=0.0001, l1_ratio=0.15, fit_intercept=True, m
ax_iter=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 Desce
nt.
# predict(X)    Predict class labels for sample
s 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_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, metho
d="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log
loss is:",log_loss(y_test, predict_y, labels=cl
f.classes_, eps=1e-15))

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

```
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], pe
nalty='l2', loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="s
igmoid")
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_tr
ain, 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_tes
t, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(pred
icted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss
is: 0.5725280378306258
For values of alpha =  0.0001 The log los

s is: 0.4552477552698689
```

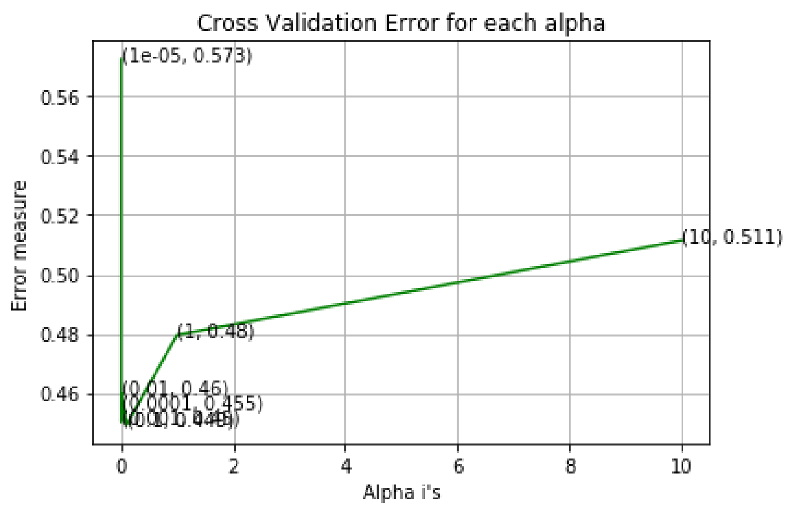
For values of alpha = 0.001 The log loss is: 0.450213019053 4189

For values of alpha = 0.01 The log loss is: 0.459910473927018

For values of alpha = 0.1 The log loss is: 0.44922600318403383

For values of alpha = 1 The log loss is: 0.47955708651426576

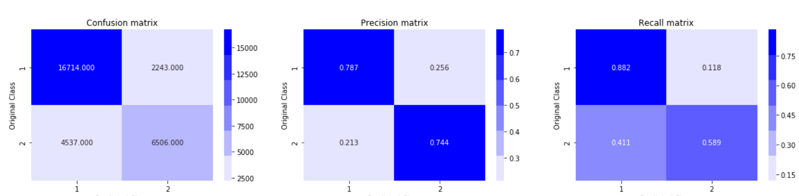
For values of alpha = 10 The log loss is: 0.5113949951147514



For values of best alpha = 0.1 The train log loss is: 0.4399629073595153

For values of best alpha = 0.1 The test log loss is: 0.44922600318403383

Total number of data points : 30000



4.5 Linear SVM with hyperparameter tuning

In [0]:

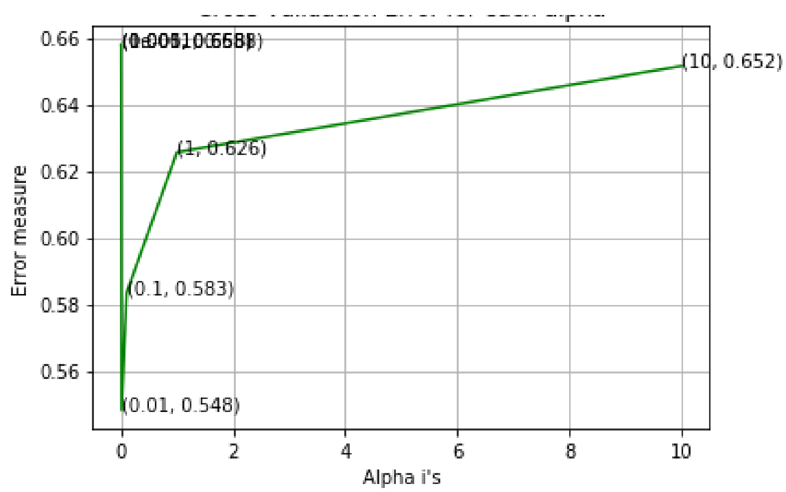
```
%%time
alpha = [10 ** x for x in range(-5, 2)] # hyper
param for SGD classifier.

# read more about SGDClassifier() at http://sci
kit-learn.org/stable/modules/generated/sklearn.
linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alp
ha=0.0001, l1_ratio=0.15, fit_intercept=True, m
ax_iter=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 Desce
nt.
# predict(X)    Predict class labels for sample
s 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_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, metho
d="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log
loss is:",log_loss(y_test, predict_y, labels=cl
f.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], pe
nalty='l1', loss='hinge', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="s
igmoid")
sig_clf.fit(X_train, y_train)
```

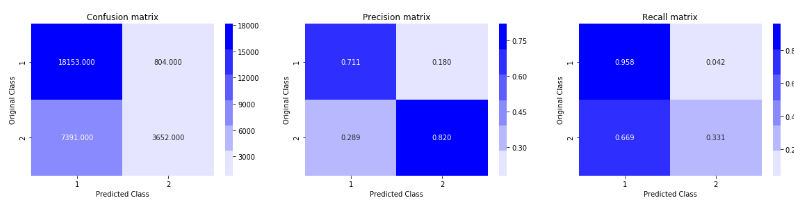
```
predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, 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: 0.6579367200579354
For values of alpha =  0.0001 The log loss is: 0.6579367200579354
For values of alpha =  0.001 The log loss is: 0.6579367200579354
For values of alpha =  0.01 The log loss is: 0.5481499979750184
For values of alpha =  0.1 The log loss is: 0.5832984756289242
For values of alpha =  1 The log loss is: 0.6256469625313519
For values of alpha =  10 The log loss is: 0.6515001183736375
```

Cross Validation Error for each alpha




```
For values of best alpha =  0.01 The trai
n log loss is: 0.545125304525286
For values of best alpha =  0.01 The test
log loss is: 0.5481499979750184
Total number of data points : 30000
```




```
CPU times: user 2h 29min 53s, sys: 3min 1
8s, total: 2h 33min 11s
Wall time: 2h 29min 12s
Parser   : 135 ms
```

4.6 XGBoost

In [0]:

```
%%time
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, 'vali
d')]

bst = xgb.train(params, d_train, 400, watchlis
t, 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:0.684854  valid-log
loss:0.684817
Multiple eval metrics have been passed:
'valid-logloss' will be used for early st
opping.

Will train until valid-logloss hasn't imp
roved in 20 rounds.
```

```
[10]    train-logloss:0.615155  valid-log
loss:0.615065
[20]    train-logloss:0.563704  valid-log
loss:0.563578
[30]    train-logloss:0.525997  valid-log
loss:0.525726
[40]    train-logloss:0.496892  valid-log
loss:0.496527
[50]    train-logloss:0.473233  valid-log
loss:0.472884
```

```
[60]	train-logloss:0.454724	valid-log
loss:0.454407
[70]	train-logloss:0.43984	valid-log
loss:0.439597
[80]	train-logloss:0.427771	valid-log
loss:0.427637
[90]	train-logloss:0.417995	valid-log
loss:0.417895
[100]	train-logloss:0.409728	valid-log
loss:0.409714
[110]	train-logloss:0.402778	valid-log
loss:0.402847
[120]	train-logloss:0.396942	valid-log
loss:0.397114
[130]	train-logloss:0.392061	valid-log
loss:0.392382
[140]	train-logloss:0.387872	valid-log
loss:0.388373
[150]	train-logloss:0.384392	valid-log
loss:0.385178
[160]	train-logloss:0.381274	valid-log
loss:0.382299
[170]	train-logloss:0.378437	valid-log
loss:0.379793
[180]	train-logloss:0.375854	valid-log
loss:0.37747
[190]	train-logloss:0.373571	valid-log
loss:0.375462
[200]	train-logloss:0.371199	valid-log
loss:0.373368
[210]	train-logloss:0.368932	valid-log
loss:0.371358
[220]	train-logloss:0.367146	valid-log
loss:0.369816
[230]	train-logloss:0.365382	valid-log
loss:0.368348
```

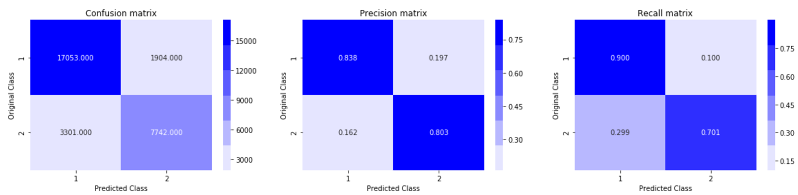
```
[240]   train-logloss:0.363378  valid-log
loss:0.366641
[250]   train-logloss:0.361745  valid-log
loss:0.365325
[260]   train-logloss:0.36017   valid-log
loss:0.364083
[270]   train-logloss:0.358561  valid-log
loss:0.36283
[280]   train-logloss:0.357068  valid-log
loss:0.361743
[290]   train-logloss:0.355691  valid-log
loss:0.360688
[300]   train-logloss:0.354336  valid-log
loss:0.359735
[310]   train-logloss:0.35309   valid-log
loss:0.35884
[320]   train-logloss:0.351938  valid-log
loss:0.358094
[330]   train-logloss:0.35075   valid-log
loss:0.357309
[340]   train-logloss:0.349477  valid-log
loss:0.356474
[350]   train-logloss:0.348308  valid-log
loss:0.355684
[360]   train-logloss:0.347177  valid-log
loss:0.355021
[370]   train-logloss:0.346093  valid-log
loss:0.354314
[380]   train-logloss:0.344985  valid-log
loss:0.353589
[390]   train-logloss:0.343936  valid-log
loss:0.352981
[399]   train-logloss:0.342996  valid-log
loss:0.352343
The test log loss is: 0.352431428180212
CPU times: user 1h 1min 14s, sys: 288 ms,
```

```
total: 1h 1min 14s
Wall time: 1min 35s
```

In [0]:

```python
predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

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



5. Assignments

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

In [3]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
```

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

```
from sklearn.ensemble import RandomForestClassi
fier



from sklearn.model_selection import cross_val_s
core
from sklearn.linear_model import SGDClassifier
from mlxtend.classifier import StackingClassifi
er

from sklearn import model_selection
from sklearn.linear_model import LogisticRegres
sion
from sklearn.metrics import precision_recall_cu
rve, auc, roc_curve
```

```
/usr/local/lib/python3.6/dist-packages/sk
learn/externals/six.py:31: DeprecationWar
ning: The module is deprecated in version
0.21 and will be removed in version 0.23
since we've dropped support for Python 2.
7. Please rely on the official version of
six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", Dep
recationWarning)
```

Perfom Modeling on complete dataset with TF-IDF Features

In [0]:

```
# This function plots the confusion matrices gi
```

```
ven 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) represent
s number of points of class i are predicted cla
ss 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 col
umns and axis=1 corresponds to rows in two diam
ensional 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 col
umns and axis=1 corresponds to rows in two diam
ensional 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)
    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()
```

In [0]:

```
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
```

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

```
V
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]:

```
dfnlp = pd.read_csv("/content/drive/My Drive/Quora/nlp_features_train.csv",encoding='latin-1')
dfppro = pd.read_csv("/content/drive/My Drive/Quora/df_fe_without_preprocessing_train.csv",encoding='latin-1')
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3 = dfnlp[['id','question1','question2']]
duplicate = dfnlp.is_duplicate
```

In [0]:

```
df3 = df3.fillna(' ')
#assigning new dataframe with columns question
```

```
(q1+q2) and id same as df3
new_df = pd.DataFrame()
new_df['questions'] = df3.question1 + ' ' + df3
.question2
new_df['id'] = df3.id
df2['id']=df1['id']
new_df['id']=df1['id']
final_df = df1.merge(df2, on='id',how='left') #
merging df1 and df2
X  = final_df.merge(new_df, on='id',how='left')
#merging final_df and new_df
```

In [8]:

```
#removing id from X
X=X.drop('id',axis=1)
X.columns
```

Out[8]:

```
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_rat
io', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_sub
str_ratio', 'freq_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q
2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_
q1+q2', 'freq_q1-q2', 'questions'],
      dtype='object')
```

In [0]:

```
y=np.array(duplicate)
```

In [0]:

```
#splitting data into train and test
X_train,X_test,y_train,y_test=train_test_split(
X,y,random_state=3,test_size=0.3)
```

In [11]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(283003, 27)
(283003,)
(121287, 27)
(121287,)
```

In [0]:
```
#seperating questions for tfidf vectorizer
X_train_ques=X_train['questions']
X_test_ques=X_test['questions']

X_train=X_train.drop('questions',axis=1)
X_test=X_test.drop('questions',axis=1)
```

In [0]:
```
#tfidf vectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,4
),min_df=10)
X_train_tfidf=tf_idf_vect.fit_transform(X_train
_ques)
X_test_tfidf=tf_idf_vect.transform(X_test_ques)
```

In [14]:
```
#adding tfidf features to our train and test da
ta using hstack
X_train = hstack((X_train.values,X_train_tfidf
))
X_test= hstack((X_test.values,X_test_tfidf))
print(X_train.shape)
print(X_test.shape)
```

```
(283003, 156393)
(121287, 156393)
```

In [0]:
```
# #standardising data
# from sklearn import preprocessing
# scaler = preprocessing.StandardScaler(with_me
an=False)
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)
```

Applying Logistic Regression

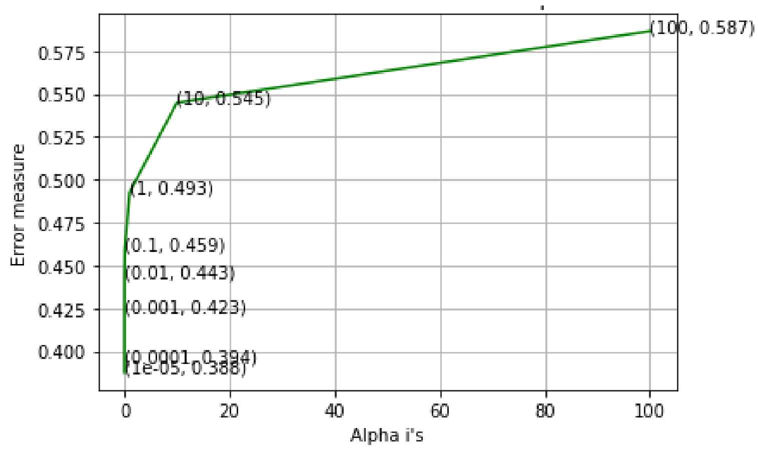
In [0]:
```
alpha = [10 ** x for x in range(-5, 3)] # hyper
param for SGD classifier.

# read more about SGDClassifier() at http://sci
kit-learn.org/stable/modules/generated/sklearn.
linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alp
ha=0.0001, l1_ratio=0.15, fit_intercept=True, m
ax_iter=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 Desce
nt.
# predict(X)    Predict class labels for sample
s 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_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, metho
d="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log
loss is:",log_loss(y_test, predict_y, labels=cl
f.classes_, eps=1e-15))

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


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], pe
nalty='l2', loss='log', random_state=42)
```

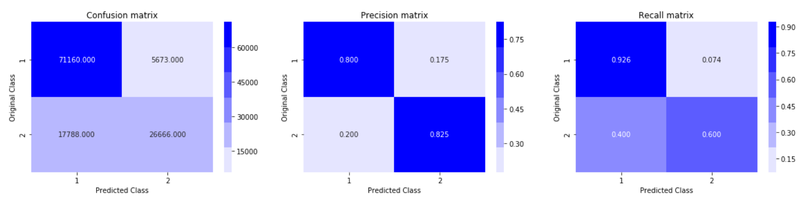
```
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="s
igmoid")
sig_clf.fit(X_train, y_train)

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_tr
ain, 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_tes
t, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(pred
icted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss
is: 0.3876589202822967
For values of alpha =  0.0001 The log los
s is: 0.3937861342051053
For values of alpha =  0.001 The log loss
is: 0.4233023910050718
For values of alpha =  0.01 The log loss
is: 0.4434536788024677
For values of alpha =  0.1 The log loss i
s: 0.4591654650352666
For values of alpha =  1 The log loss is:
0.4926263271528424
For values of alpha =  10 The log loss i
s: 0.5448710637409396
For values of alpha =  100 The log loss i
s: 0.5865222819503731
```

Cross Validation Error for each alpha




```
For values of best alpha =  1e-05 The train log loss is: 0.3849777035821042
For values of best alpha =  1e-05 The test log loss is: 0.3876589202822967
Total number of data points : 121287
```



Applying Linear SVM

In [0]:

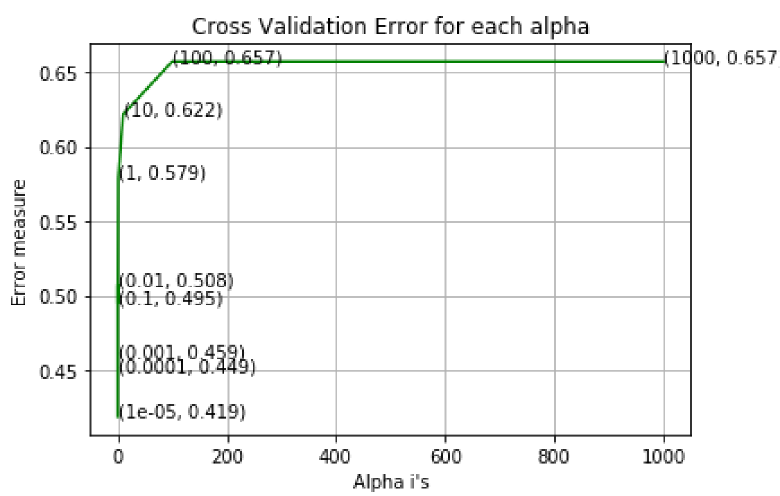
```
%%time
alpha = [10 ** x for x in range(-5, 4)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alp
```

```
ha=0.0001, l1_ratio=0.15, fit_intercept=True, m
ax_iter=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 Desce
nt.
# predict(X)    Predict class labels for sample
s 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_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, metho
d="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, pre
dict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log
loss is:",log_loss(y_test, predict_y, labels=cl
f.classes_, eps=1e-15))

fig, ax = plt.subplots()
```

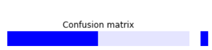
```
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_arra
y,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (al
pha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alph
a")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], pe
nalty='l1', loss='hinge', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="s
igmoid")
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_tr
ain, predict_y, labels=clf.classes_, eps=1e-1
5))
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_tes
t, predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(pred
icted_y))
plot_confusion_matrix(y_test, predicted_y)
```

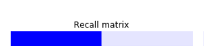
```
For values of alpha =  1e-05 The log loss
```

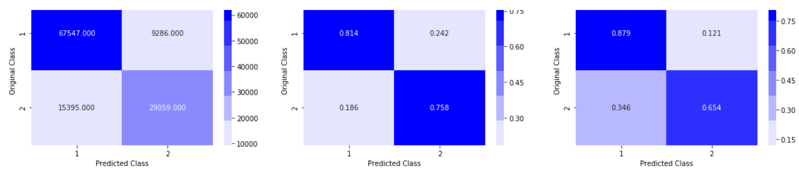
For values of alpha = 1e-05 The log loss
is: 0.4187868285636254
For values of alpha = 0.0001 The log los
s is: 0.4494938960115246
For values of alpha = 0.001 The log loss
is: 0.4590994398298589
For values of alpha = 0.01 The log loss
is: 0.507502478793165
For values of alpha = 0.1 The log loss i
s: 0.49495228963091503
For values of alpha = 1 The log loss is:
0.5790224258771397
For values of alpha = 10 The log loss i
s: 0.6216484120064324
For values of alpha = 100 The log loss i
s: 0.6571084989895938
For values of alpha = 1000 The log loss
is: 0.6571084989603588



For values of best alpha = 1e-05 The tra
in log loss is: 0.4170704354727345
For values of best alpha = 1e-05 The tes
t log loss is: 0.4187868285636254
Total number of data points : 121287








```
CPU times: user 37min 4s, sys: 16min 14
s, total: 53min 18s
Wall time: 34min 7s
```

XGBOOST

In [0]:
```python
import xgboost as xgb
```

In [0]:
```python
%%time
n_estimators = [50,100,150,200,300,400,500]
test_scores = []
train_scores = []
for i in n_estimators:
    clf = xgb.XGBClassifier(learning_rate=0.1,n
_estimators=i,n_jobs=-1)
    clf.fit(X_train,y_train)
    y_pred = clf.predict_proba(X_train)
    log_loss_train = log_loss(y_train, y_pred,
 eps=1e-15)
    train_scores.append(log_loss_train)
    y_pred = clf.predict_proba(X_test)
    log_loss_test = log_loss(y_test, y_pred, ep
s=1e-15)
    test_scores.append(log_loss_test)
    print('For n_estimators = ',i,'Train Log Lo
ss ',log_loss_train,'Test Log Loss ',log_loss_t
est)
```

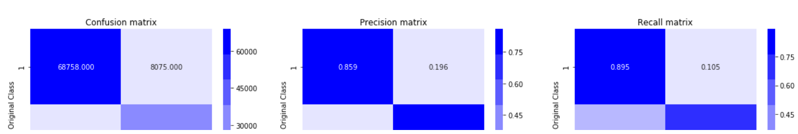
```
For n_estimators =  50 Train Log Loss  0.
3802022451846393 Test Log Loss  0.3815880
```

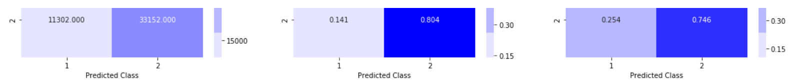
```
153715477
For n_estimators =  100 Train Log Loss
0.3601035226913487 Test Log Loss  0.36236
71979711734
For n_estimators =  150 Train Log Loss
0.3506867034074368 Test Log Loss  0.35379
625301920425
For n_estimators =  200 Train Log Loss
0.3428548466945084 Test Log Loss  0.34656
819162382024
For n_estimators =  300 Train Log Loss
0.33327452415665726 Test Log Loss  0.3382
9663616353656
For n_estimators =  400 Train Log Loss
0.3269292490820392 Test Log Loss  0.33304
25109036889
For n_estimators =  500 Train Log Loss
0.32188343164123673 Test Log Loss  0.3291
1847991728505
CPU times: user 3h 23min 11s, sys: 6.24
s, total: 3h 23min 18s
Wall time: 11min 57s
```

In [0]:

```python
clf=xgb.XGBClassifier(learning_rate=0.1,n_estimators=500,n_jobs=-1)
clf.fit(X_train,y_train)
y_pred=clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, y_pred, eps=1e-15))
predicted_y =np.argmax(y_pred,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

```
The test log loss is: 0.32911847991728505
```





Hyperparameter tunning using RandomSearch

In [0]:

```
import xgboost as xgb
```

In [18]:

```
%%time
from sklearn.model_selection import RandomizedS
earchCV
param_grid = {"max_depth":[2, 5, 8, 10],
              "n_estimators":[5, 10, 50, 100]}

model = RandomizedSearchCV(xgb.XGBClassifier(n_
jobs=-1,random_state=25), param_distributions=p
aram_grid,n_iter=30,scoring='neg_log_loss',cv=
3,n_jobs=-1)

model.fit(X_train,y_train)
model.best_params_
```

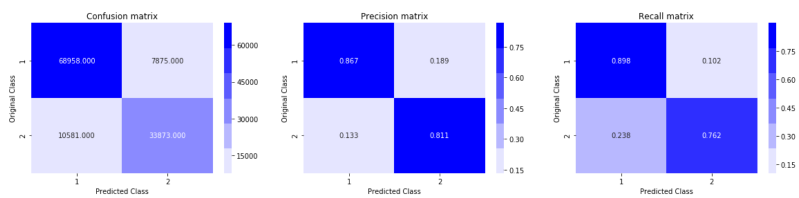
```
CPU times: user 1h 31min 26s, sys: 2.64
s, total: 1h 31min 29s
Wall time: 13min 2s
```

In [20]:

```
%%time
clf=xgb.XGBClassifier(n_jobs=-1,random_state=2
5,max_depth=10,n_estimators=100)
clf.fit(X_train,y_train)
y_pred_test=clf.predict_proba(X_test)
y_pred_train=clf.predict_proba(X_train)
log_loss_train = log_loss(y_train, y_pred_trai
```

```
n, eps=1e-15)
log_loss_test=log_loss(y_test,y_pred_test,eps=1
e-15)
print('Train log loss = ',log_loss_train,' Test
log loss = ',log_loss_test)
predicted_y=np.argmax(y_pred_test,axis=1)
plot_confusion_matrix(y_test,predicted_y)
```

```
Train log loss =  0.2793043393157809  Tes
t log loss =  0.3143578907429847
```




```
CPU times: user 35min 32s, sys: 1.35 s, t
otal: 35min 33s
Wall time: 2min 25s
```

In [0]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model","vectorizer","log los
s"]
x.add_row(['Logistic regression','TFIDF w2vec',
'0.4492'])
x.add_row(['Linear SVM','TFIDF w2vec',' 0.5481'
])
x.add_row(['XGBOOST','TFIDF w2vec','0.3523'])
x.add_row(['Logistic regression','TFIDF ',' 0.3
876'])
x.add_row(['Linear SVM','TFIDF','0.4187'])
x.add_row(['XGBOOST','TFIDF ','0.3143'])

print(x)
```

STEP BY STEP PROCEDURE:

1.As we know, we have a data set containing Number of rows 404,290, containing 5 columns: qid1, qid2, question1, question2, is duplicate from which ' is duplicate ' is a class label specifying that Question 1 and Question 2 are similar or not, and this is a binary classification problem, and we need to predict whether or not they are duplicate.

2.Firstly we preprocessed our data,did feature engineering to create new features which might help us and created our dataframes, then we merged dataframes and got out final matrix.Now after doing simple EDA on dataset we will try some Basic Feature Extraction (before cleaning) the datset like Frequency of qid1's ,word_Common and etc. and using this featured datset we will do some EDA on it so that we will able to rectify which features are most useful features our of all features i.e(wich feature is helpful for classification)

3.We will try some Advanced Feature Extraction using NLP and Fuzzy Features after doing basic Basic Feature Extractions, but before we do this we will do Text Preprocessing and then we will do Advanced Feature Extraction and try to show our Advanced Feature using EDA, PCA and word clouds.

4.Then we randomly split data. We could also have splitted time-based, as the model could also forecast unknown information for the future. But, no timestamp column was provided, so the only option was to randomly split it.

5.Now that we know we have columns of two questions i.e. Question 1 and Question 2 and we will vector that col using tfidf weighted word-vectors so that we can apply models on it and after doing all these we will merge all the features i.e. besic features + advance features + Question 1 tfidf w2v + and Question 2 tfidf w2v together. And now we're going to apply models to it after doing all of it.

6.In this case study, as we know, we use two main performance matrix, i.e. log-loss and confusion matrix, and we'll get our performance of the models there.

7.Let's start: here's a model i.e. Logistic Regression linear svm and XgBoost and a random model where the worst-case log-loss is found and then we're trying to compare it.

8.In the next step we will test our models with other vectorizers i.e. tfidf instead of tfidf weighted w2v and seek some hyperparameter tuning to improve the performance of the model.

9.We used a simple Random / Dumb model now. It gave 0.88 log loss. That's the worst log-loss scenario. This will serve as a foundation and any system that we build should have a lower log-loss than this dumb model.

10.After that we have applied Logistic Regression with hyperparameter tuning. It gave a log-loss of 0.38, which is lower than Random Model. We can also see that there is no Overfitting problem , since, Train log-loss and Test log-loss and very close.

11.After that we have applied Linear SVM with hyperparameter tuning. It gave the log-loss of 0.41,which is lower than Random Model. We can also see that there is no Overfitting problem , since, Train log-loss and Test log-loss and very close.

12.After that we have applied Xgboost with hyperparameter tuning. It gave the log-loss of