Quora Question Pairs

1. Business Problem

1.1 Description

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

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

credits:kaggle

Problem Statement

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

1.3 Source/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/AACZdtsApc1QSTQc7X0H3QZ5a?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

- 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

2.2.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

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.

3. Exploratory Data Analysis

In [0]:

```
!pip install fuzzywuzzy
```

```
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.6/dist-packages (0.18.0)
```

Importing required libraries

In [0]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly
import math
import string
import re
import nltk
from nltk import SnowballStemmer , PorterStemmer
import collections
from bs4 import BeautifulSoup
from wordcloud import STOPWORDS
from fuzzywuzzy import fuzz
from fuzzywuzzy import process
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss , confusion_matrix
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
/usr/local/lib/python3.6/dist-packages/fuzzywuzzy/fuzz.py:11: UserWarning:

Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning
```

Loading Dataset

In [0]:

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

In [0]:

```
df.info()
```

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

In [0]:

```
df
```

Out[0]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0
...	...	...	...	...	...	...
404285	404285	433578	379845	How many keywords are there in the Racket prog...	How many keywords are there in PERL Programmin...	0

	id	qid1	qid2	question1	question2	is_duplicate
404286	404286	18840	155606	Do you believe there is life after death?	Is it true that there is life after death?	1
404287	404287	537928	537929	What is one coin?	What's this coin?	0
404288	404288	537930	537931	What is the approx annual cost of living while...	I am having little hairfall problem but I want...	0
404289	404289	537932	537933	What is like to have sex with cousin?	What is it like to have sex with your cousin?	0

404290 rows × 6 columns

In [0]:

```
df.shape
```

Out[0]:

```
(404290, 6)
```

In [0]:

```
df.columns
```

Out[0]:

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

The columns present in the data set are id, qid1 , qid2, question1 , question2 , is_duplicate. So, our dependent Vars are qid1 , qid2, question1 , question2 and indepent var/Target var is is_duplicate

Before proceeding for any thing , I have to check for NaN values because they cause some problem and they should be handled.

In [0]:

```
df[df.isna().any(1)]
```

Out[0]:

	id	qid1	qid2	question1	question2	is_duplicate
105780	105780	174363	174364	How can I develop android app?	NaN	0
201841	201841	303951	174364	How can I create an Android app?	NaN	0
363362	363362	493340	493341	NaN	My Chinese name is Haichao Yu. What English na...	0

As we can there are three questions that has NaN values. I need to replace them with something , better go for replacing them with emtpy strings

In [0]:

```
df=df.fillna(value=" ")
```

Lets check the changes, and find if we can see any other NaN values

In [0]:

```
df[df.isna().any(1)]
```

Out[0]:

	id	qid1	qid2	question1	question2	is_duplicate

In [0]:

```
df
```

Out[0]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0
...	...	...	...	...	...	...
404285	404285	433578	379845	How many keywords are there in the Racket prog...	How many keywords are there in PERL Programmin...	0
404286	404286	18840	155606	Do you believe there is life after death?	Is it true that there is life after death?	1
404287	404287	537928	537929	What is one coin?	What's this coin?	0
404288	404288	537930	537931	What is the approx annual cost of living while...	I am having little hairfall problem but I want...	0
404289	404289	537932	537933	What is like to have sex with cousin?	What is it like to have sex with your cousin?	0

404290 rows × 6 columns

3.1 Basic questions on Dataset / distribution of datapoints with respect to class labels

Q1: How is the class label (is_duplicate) distributed with respect to data points?

In [0]:

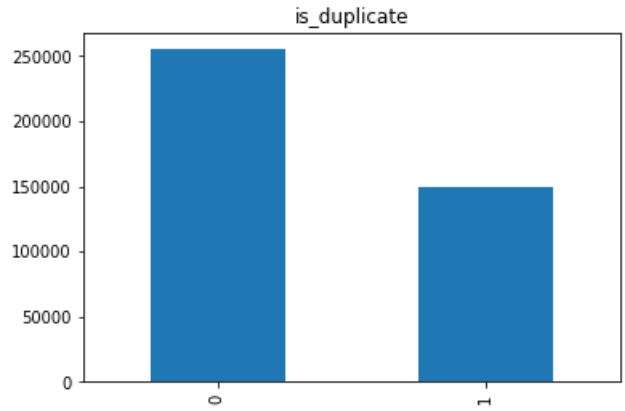
```
df.is_duplicate.value_counts()
```

Out[0]:

```
0    255027
1    149263
Name: is_duplicate, dtype: int64
```

In [0]:

```
df.is_duplicate.value_counts().plot.bar()
plt.title("is_duplicate")
plt.show()
```



As we can see we have fairly unbalanced dataset, for is_duplicate = 0 we have 255027 data points and for is_duplicate = 1 we have 149263 data points

Q2.Are these questions repeating multiple times?

Simply by Logic to repeat multiple times there should be two or more other data points with same 'qid1', 'qid2', 'question1', 'question2' , 'is_duplicate'

so , i can drop these data.

In [0]:

```
final_df=df.drop_duplicates(subset={'qid1','qid2','question1','question2','is_duplicate'}, keep='first', inplace=False)
```

In [0]:

```
final_df.shape
```

Out[0]:

```
(404290, 6)
```

In [0]:

```
# Intial DF
df.shape
```

Out[0]:

```
(404290, 6)
```

As we can see there were no duplicates in the data set by seeing the intial df size and after removing duplicates the df size.

Q3.Can we see unique questions and repeated questions ?

we can know them by looking at Question Id's

In [0]:

```
x_total_questions = df.qid1.values.tolist() + df.qid2.values.tolist()
```

In [0]:

```
y_repeated_questions=pd.DataFrame(x_total_questions)
```

In [0]:

```
# These are the total questions in data set with repeations
total_questions_in_dataFrame=len(x_total_questions)
```

In [0]:

```
# these are the unique questions
totalnumber_of_unique_questions = len(set(x_total_questions))
```

In [0]:

```
# these are no of questions appreared more than one time
noof_questions_appeared_morethanonetime = np.sum((y_repeated_questions[0].value_counts()>1))
```

In [0]:

```
y_repeated_questions
type(y_repeated_questions)
```

Out[0]:

```
pandas.core.frame.DataFrame
```

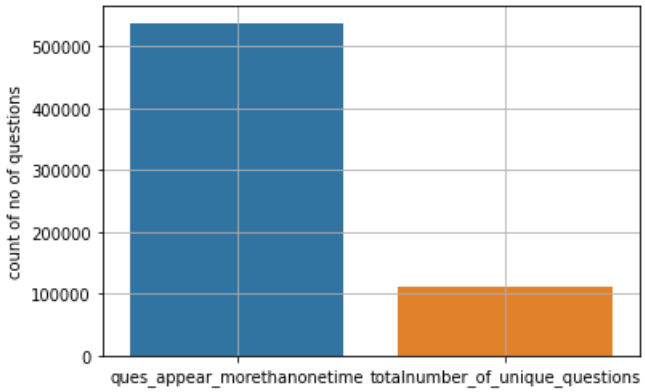
In [0]:

```
print("the total no of questions in Dataframe is {0} , the total no of unique questions in data fr
ame is {1} and \nthe number of questions repeated more than one time is
{2}".format(total_questions_in_dataFrame,totalnumber_of_unique_questions,noof_questions_appeared_mo
ethanonetime))
```

```
the total no of questions in Dataframe is 808580 , the total no of unique questions in data frame
is 537933 and
the number of questions repeated more than one time is  111780
```

In [0]:

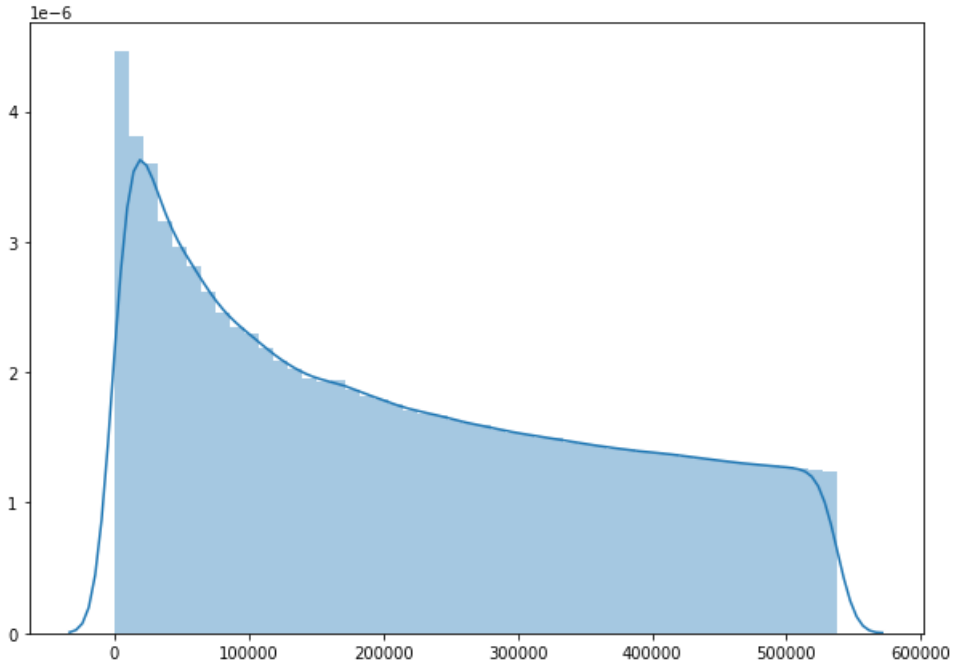
```
#plotting the above
x=["ques_appear_morethanonetime","totalnumber_of_unique_questions"]
y=[totalnumber_of_unique_questions,noof_questions_appeared_morethanonetime]
sns.barplot( x,y)
plt.ylabel("count of no of questions")
plt.grid("white")
plt.show()
```



As we can see there are more no of questions that appeared more than once

In [0]:

```
#plt.hist( , data=noof_questions_appeared_morethanonetime)
plt.figure(figsize=(10,7))
sns.distplot(y_repeated_questions)
plt.show()
```



As we answered the questions lets go to the featurisations part to get insights about data and see if it can help in out objective of classification or not.

3.2 Fearisation to get more insights about the data that help in objective of classification

As our data set is having question1 and question2 features just by looking at these we cannot make sense as we cannot plot them as they are actual questions itself and by logic we know that if two questions are different then there will/will not be different/not different words with or without the semantic meanings of the words everything depends on the context. As we are humans reading the pair of questions it will be easy to understand for us and differentiate .For a machine to differentiate means it needs data in machine readable form that is numbers. Here in this part we will create some own features based on the questions we have with out cleaning the questions and preproccesing them and perform **EDA** on them ,Later we can convert sentances and create advance features and do **EDA** on them as well to know these features are helpful or not.

Defining these Features :---

- no_words_in_question1 :- total words in question1
- no_words_in_question2 :- total words in question2
- len_of_question1 :- length of the question1
- len_of_question2 :- length of the question2
- unique_commonwords_inboth_qestions :- total common words which are unique to both questions
- frequency_of_question1 :- no of times this question1 occurs
- frequency_of_question2 :- no of times this question2 occurs
- word_share :- this is basically words shared between two sentances,uniquecmmnwords q1+q2/totalnoofwordsin q1+q2
- freq1+freq2 :- freqency of q1 + freq q2

- freq1-freq2 :- abs(frequency of q1 - freq q2)
- total_noof_words_q1+q2 :- no of words in question1+question2

In [0]:

```
# creating functions for these features

def noWordsInQuestion1(data):
  '''
  This function is used to take a element  and compute the no of words in each element
  '''

  return (len((data).split(" ")))


def noWordsInQuestion2(data):
  '''
  This function is used to take a element and compute the no of words in each element
  '''

  return (len((data).split(" ")))

def lengthOfQuestion1(data):
  '''
  This Function is used to compute the length of the element
  '''
  return len(data)

def lengthOfQuestion2(data)  :
  '''
  This Function is used to compute the length of the element
  '''
  return len(data)

def uniqueCommonWordsInBothQestions(data):
  '''
  This Dunction is used to compute the Total common words shared between two questions
  '''

  q1=data['question1']
  q2=data['question2']

  q1_words=(set(q1.split(" ")))
  q2_words=(set(q2.split(" ")))

  return len((q1_words.intersection(q2_words)))

def wordShare(data):
  '''
  This function is used to caluculate the wordshare
  '''

  q1=data['question1']
  q2=data['question2']

  q1_words=(set(q1.split(" ")))
  q2_words=(set(q2.split(" ")))
  length_numerator=len((q1_words.intersection(q2_words)))

  q1_words_length=len(q1.split(" "))
  q2_words_length=len(q2.split(" "))
  length_denominator=q1_words_length + q2_words_length

  total=length_numerator/length_denominator
  return total
```

***applying the functions to the dataset**

In [0]:

```
df['no_words_in_question1']=df['question1'].apply(noWordsInQuestion1)
df['no_words_in_question2']=df['question2'].apply(noWordsInQuestion2)
df['len_of_question1']=df['question1'].apply(lengthOfQuestion1)
```

```
df['len_of_question2']=df['question2'].apply(lengthOfQuestion2)
df['commonUniqueWords_inBothQuestions']=df.apply(uniqueCommonWordsInBothQestions , axis=1)
df['frequency_of_question1'] = df.groupby('qid1')['qid1'].transform('count')
df['frequency_of_question2'] = df.groupby('qid2')['qid2'].transform('count')
df['wordshare']=df.apply(wordShare , axis=1)
df['fq1+fq2']=df['frequency_of_question1']+df['frequency_of_question2']
df['fq1-fq2']=abs(df['frequency_of_question1']-df['frequency_of_question2'])
df['total_no_of_words_q1+q2']=df['no_words_in_question1']+df['no_words_in_question2']
```

In [0]:

```
df.columns
```

Out[0]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'no_words_in_question1', 'no_words_in_question2', 'len_of_question1',
       'len_of_question2', 'commonUniqueWords_inBothQuestions',
       'frequency_of_question1', 'frequency_of_question2', 'wordshare',
       'fq1+fq2', 'fq1-fq2', 'total_no_of_words_q1+q2'],
      dtype='object')
```

As we have added extra features lets do EDA on them and check if they justify to our objective

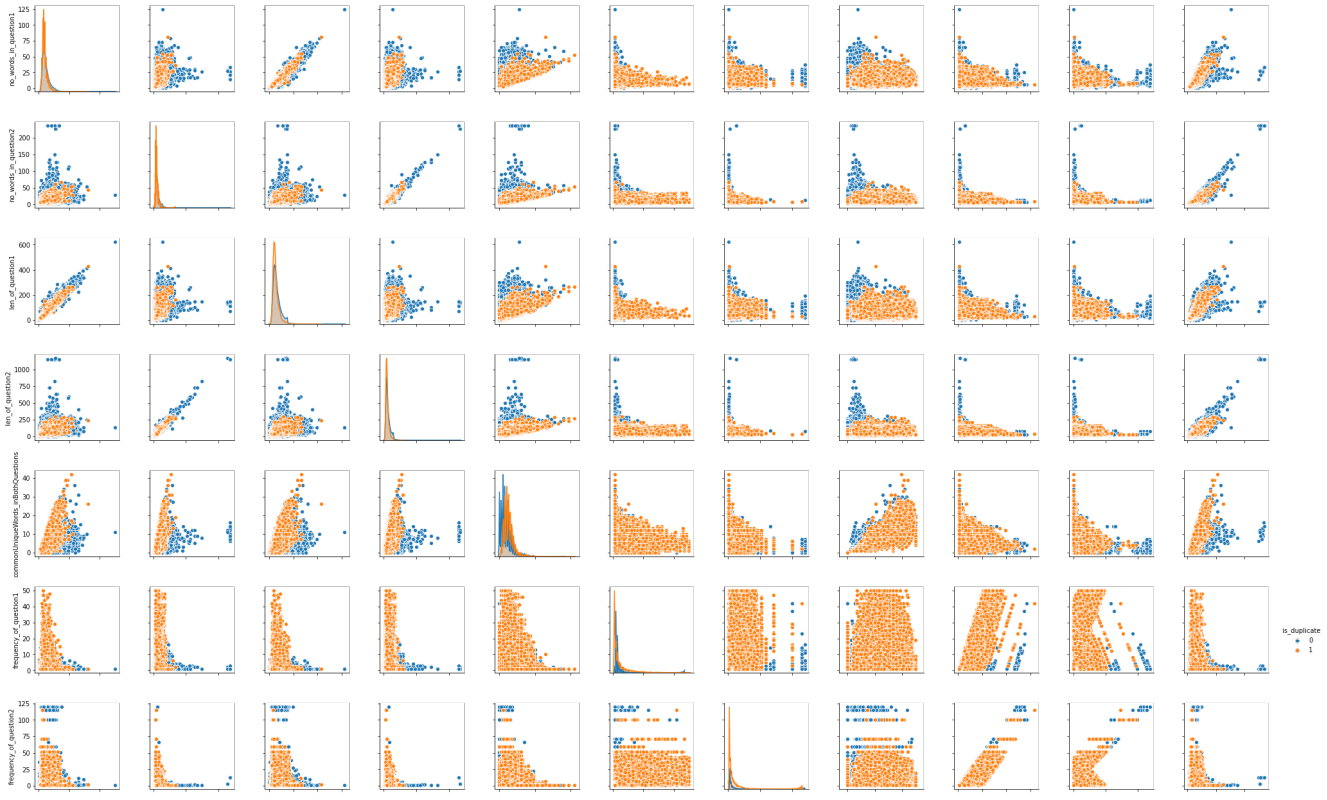
3.2.1 EDA on Basic Features Created

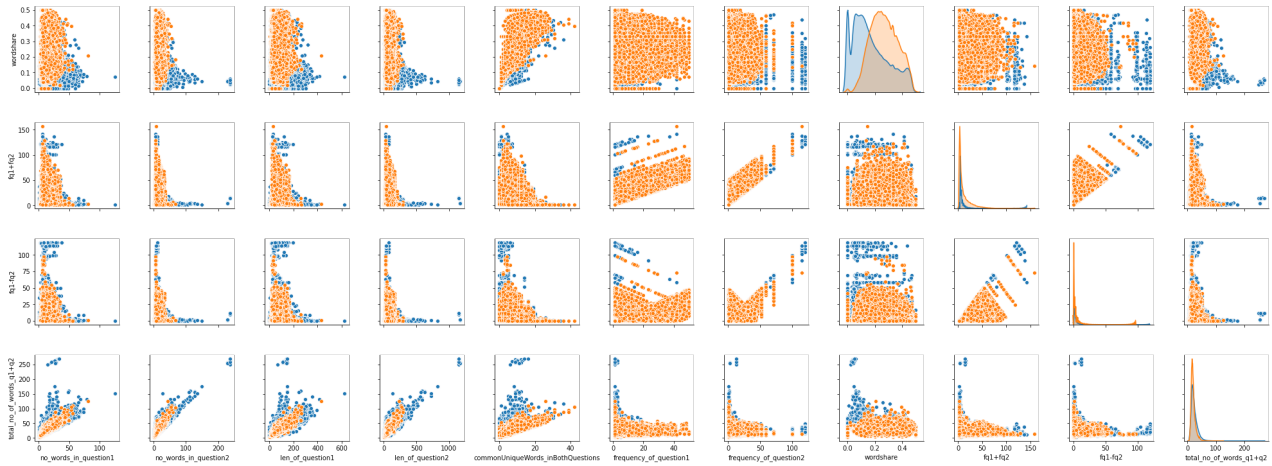
In [0]:

```
dnew_eda=df[['no_words_in_question1','no_words_in_question2','len_of_question1',
       'len_of_question2', 'commonUniqueWords_inBothQuestions',
       'frequency_of_question1', 'frequency_of_question2', 'wordshare',
       'fq1+fq2', 'fq1-fq2', 'total_no_of_words_q1+q2','is_duplicate']]
```

In [0]:

```
sns.pairplot(dnew_eda,hue='is_duplicate')
plt.show()
```





by looking at above observations i can see word share and common words are performing good, " word share and common unique words " than others lets plot these features for pdfs , and histograms

3.2.2 Univariate Analysis and Bi variate Analysis

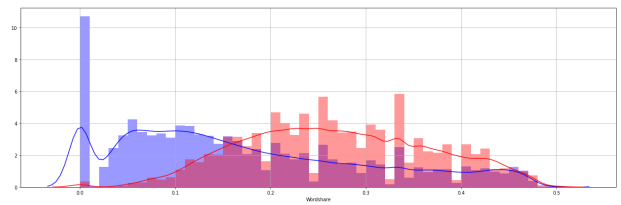
By Looking at the previous plots we came to conclusion that word share and common unique words are the two features that help towards our objective at hand comparitively than other features

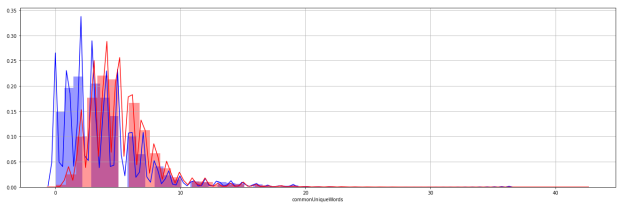
Lets perform univariate analysis on them.

Univariate Analysis :

In [0]:

```
plt.figure(1 ,figsize=(50,7))
plt.subplot(1,2,1  )
sns.distplot(df[df['is_duplicate']== 0.0]['wordshare'],color='blue' , bins = 50)
sns.distplot(df[df['is_duplicate']==1.0]['wordshare'] ,color='red',bins = 50)
plt.xlabel('Wordshare')
plt.grid('white')

plt.subplot(1,2,2)
sns.distplot(df[df['is_duplicate']== 0.0]['commonUniqueWords_inBothQuestions'],color='blue', bins =
50)
sns.distplot(df[df['is_duplicate']== 1.0]['commonUniqueWords_inBothQuestions'],color='red', bins =
50)
plt.grid('White')
plt.xlabel('commonUniqueWords')

plt.show()
```



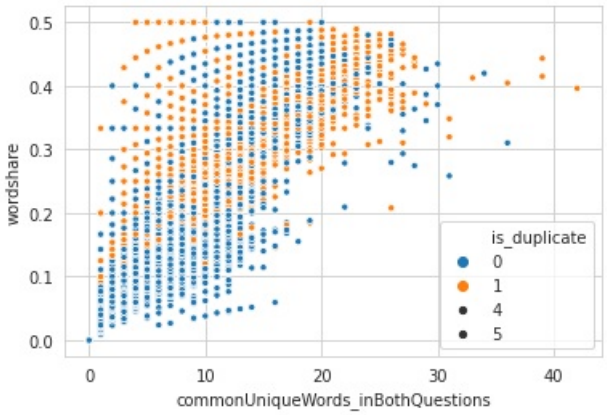


- There is some sort of seperation in intial part of the graph, so we can say that these two new features are usefull to some extent in our objective of classification.

BiVariable Analysis :

In [0]:

```
sns.set_style('whitegrid')
sns.scatterplot(data=df,y='wordshare',x='commonUniqueWords_inBothQuestions',size=5,hue='is_duplicat
e')
plt.show()
```



- As you can see by scatterplot above we can conclude that there is atleast some seperation of is_duplicate=0 and is_dulicate=1 points so this two features are helpful in our objective of classification.

- As the EDA part is done lets go to data cleaning part so that after cleaning we can create advance features and perform analyzing

- Lets add some advanced Features in to our dataset

3.2.2 Advaced 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/

lets write functions to acheive the features we need

In [0]:

```
# token:- when we split by space we get token words
# word :- which is a token and not a stop word
# stop words :- stopwords

def cwc_min_ratio(data):
  '''
  This function is used to caluculate ratio common word count to min (len(q1),len(q2)) given two q
uestions
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  words_q1=q1_words.split(" ")
  words_q2 = q2_words.split(" ")

  w_q1=[ word for word in words_q1 if word not in STOPWORDS]
  w_q2=[ word for word in words_q2 if word not in STOPWORDS]

  cwc_numerator= len((set(w_q1)).intersection(set(w_q2)))
  cwc_denominator = (min(len(w_q1), len(w_q2)) +0.0001)

  return (cwc_numerator / cwc_denominator )

def cwc_max_ratio(data):
  '''
  This function is used to caluculate ratio common word count to max (len(q1),len(q2)) given two q
uestions
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  words_q1=q1_words.split(" ")
  words_q2 = q2_words.split(" ")

  w_q1=[ word for word in words_q1 if word not in STOPWORDS]
  w_q2=[ word for word in words_q2 if word not in STOPWORDS]

  cwc_numerator= len((set(w_q1)).intersection(set(w_q2)))
  cwc_denominator = (max(len(w_q1), len(w_q2)) + +0.0001)
```

```
  return (cwc_numerator / cwc_denominator )

def ctc_min_ratio(data):
  '''
  THis function is used to caluculate the ratio of common tokens to min( len(q1),len(q2) )
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  tokens_q1=q1_words.split(" ")
  tokens_q2 = q2_words.split(" ")

  t_q1= set(tokens_q1)
  t_q2=set(tokens_q2)

  ctc_numerator = len(t_q1.intersection(t_q2))
  ctc_denominator= (min(len(tokens_q1),len(tokens_q2)) +0.0001)

  return (ctc_numerator/ ctc_denominator )

def ctc_max_ratio(data):
  '''
  THis function is used to caluculate the ratio of common tokens to max( len(q1),len(q2) )
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  tokens_q1=q1_words.split(" ")
  tokens_q2 = q2_words.split(" ")

  t_q1= set(tokens_q1)
  t_q2=set(tokens_q2)

  ctc_numerator = len(t_q1.intersection(t_q2))
  ctc_denominator= (max(len(tokens_q1),len(tokens_q2))   +0.0001)

  return (ctc_numerator / ctc_denominator)

def csc_min_ratio(data):
  '''
  This function is used to caluculate ratio common stop word count to min (len(q1),len(q2)) given
two questions
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  words_q1=q1_words.split(" ")
  words_q2 = q2_words.split(" ")

  stopwords_q1=[ word for word in words_q1 if word  in STOPWORDS]
  stopwords_q2=[ word for word in words_q2 if word  in STOPWORDS]

  csc_numerator= len((set(stopwords_q1)).intersection(set(stopwords_q2)))
  csc_denominator = ((min(len(stopwords_q1), len(stopwords_q2))) +0.0001)

  return (csc_numerator / csc_denominator )

def csc_max_ratio(data):
  '''
  This function is used to caluculate ratio common stop word count to max (len(q1),len(q2)) given
two questions
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  words_q1=q1_words.split(" ")
  words_q2 = q2_words.split(" ")

  stopwords_q1=[ word for word in words_q1 if word  in STOPWORDS]
  stopwords_q2=[ word for word in words_q2 if word  in STOPWORDS]

  csc_numerator= len((set(stopwords_q1)).intersection(set(stopwords_q2)))
  csc_denominator = (max(len(stopwords_q1), len(stopwords_q2)) +0.0001)

  return (csc_numerator / csc_denominator )

def lastWordEqual(data):
```

```
  '''
  This function is used to compareLast words of two pair of questions and return 1 or 0
  '''
  q_1=data['question1']
  q_2=data['question2']

  q_1_words=q_1.split(" ")
  q_2_words=q_2.split(" ")

  if q_1_words[-1] == q_2_words[-1]:
    return (1)
  else:
    return (0)


def firstWordEqual(data):
  '''
  This function is used to compareFirst words of two pair of questions and return 1 or 0
  '''
  q_1=data['question1']
  q_2=data['question2']

  q_1_words=q_1.split(" ")
  q_2_words=q_2.split(" ")

  if q_1_words[0] == q_2_words[0]:
    return (1)
  else:
    return (0)

def tokenLengthDIff(data):
  '''
  This function is used to caluculate the ABS diff of len(q1_tokes) and len (Q2_tokens)
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  tokens_q1=q1_words.split(" ")
  tokens_q2 = q2_words.split(" ")
  diff=abs(len(tokens_q1)- len(tokens_q2))
  return (diff )

def tokenLengthAvg(data):
  '''
  This function is used to caluculate the avg of len(q1_tokes) and len (Q2_tokens)
  '''
  q1_words=data['question1']
  q2_words=data['question2']

  tokens_q1=q1_words.split(" ")
  tokens_q2 = q2_words.split(" ")
  avg=(len(tokens_q1)+ len(tokens_q2))/2
  return (avg)

def fuzzRatio(data):
  '''
  this function is used to calculate the FuzzRatio of pari of questions
  '''
  return fuzz.ratio(data['question1'],data['question2'])


def fuzzPartialRatio(data):
  '''
  This function is used to compute fuzz partial ratio of two questions
  '''
  return fuzz.partial_ratio(data['question1'],data['question2'])


def tokeSetRatio(data):
  '''
  This function is used to compute tokenset ratio of two questions
  '''
  return fuzz.token_set_ratio(data['question1'],data['question2'])

def tokenSortRatio(data):
  '''
  This function is used to cimpute token sort ratio of two questions
```

```
    '''
    return fuzz.token_sort_ratio(data['question1'],data['question2'])
```

In [0]:

```
testingFuzzdf=df
```

In [0]:

```
testingfuzzdf1=testingFuzzdf
```

- Lets apply these functions to the data frame and get the final dataframe for eda on these new features

In [0]:

```
testingfuzzdf1['fuzzpartial']=testingfuzzdf1.apply(fuzzPartialRatio , axis=1)
testingfuzzdf1['fuzztokenset']=testingfuzzdf1.apply(tokeSetRatio , axis=1)
testingfuzzdf1['fuzztokensort']=testingfuzzdf1.apply(tokenSortRatio , axis=1)
testingfuzzdf1['fuzzratio']=testingfuzzdf1.apply(fuzzRatio ,axis =1)

testingfuzzdf1['cwcminratio']=testingfuzzdf1.apply(cwc_min_ratio , axis=1)
testingfuzzdf1['cwcmaxratio']=testingfuzzdf1.apply(cwc_max_ratio , axis=1)

testingfuzzdf1['cscminratio']=testingfuzzdf1.apply(csc_min_ratio , axis=1)
testingfuzzdf1['cscmaxratio']=testingfuzzdf1.apply(csc_max_ratio , axis=1)

testingfuzzdf1['lwordQual']=testingfuzzdf1.apply(lastWordEqual , axis=1)
testingfuzzdf1['fwordQueal']=testingfuzzdf1.apply(firstWordEqual , axis=1)

testingfuzzdf1['difftokens']=testingfuzzdf1.apply(tokenLengthDIff , axis=1)
testingfuzzdf1['avgtokens']=testingfuzzdf1.apply(tokenLengthAvg , axis=1)

testingfuzzdf1['ctcminratio']=testingfuzzdf1.apply(ctc_min_ratio , axis=1)
testingfuzzdf1['ctcmaxratio']=testingfuzzdf1.apply(ctc_max_ratio , axis=1)
```

In [0]:

```
testingfuzzdf1.shape
```

Out[0]:

```
(404290, 31)
```

In [0]:

```
#oroginal DataFrame with the changes as above with new features
df.shape
```

Out[0]:

```
(404290, 31)
```

In [0]:

```
df.columns
```

Out[0]:

```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'no_words_in_question1', 'no_words_in_question2', 'len_of_question1',
       'len_of_question2', 'commonUniqueWords_inBothQuestions',
       'frequency_of_question1', 'frequency_of_question2', 'wordshare',
       'fq1+fq2', 'fq1-fq2', 'total_no_of_words_q1+q2', 'fuzzpartial',
       'fuzztokenset', 'fuzztokensort', 'fuzzratio', 'cwcminratio',
       'cwcmaxratio', 'cscminratio', 'cscmaxratio', 'lwordQual', 'fwordQueal',
       'difftokens', 'avgtokens', 'ctcminratio', 'ctcmaxratio'],
      dtype='object')
```

3.2.3 EDA of newly created features

- lets remove the original features for testingdataset1

In [0]:

```
testingfuzzdf2=testingfuzzdf1
```

In [0]:

```
testingfuzzdf2=testingfuzzdf2.drop(columns=['id', 'qid1', 'qid2', 'question1', 'question2','no_words_in_question1', 'no_words_in_question2', 'len_of_question1','len_of_question2',
'commonUniqueWords_inBothQuestions','frequency_of_question1', 'frequency_of_question2',
'wordshare','fq1+fq2', 'fq1-fq2', 'total_no_of_words_q1+q2'])
```

In [0]:

```
#backinguporiginalDF
backup_orogianlDF_with31Features=df
```

In [0]:

```
testingfuzzdf2.columns
```

Out[0]:

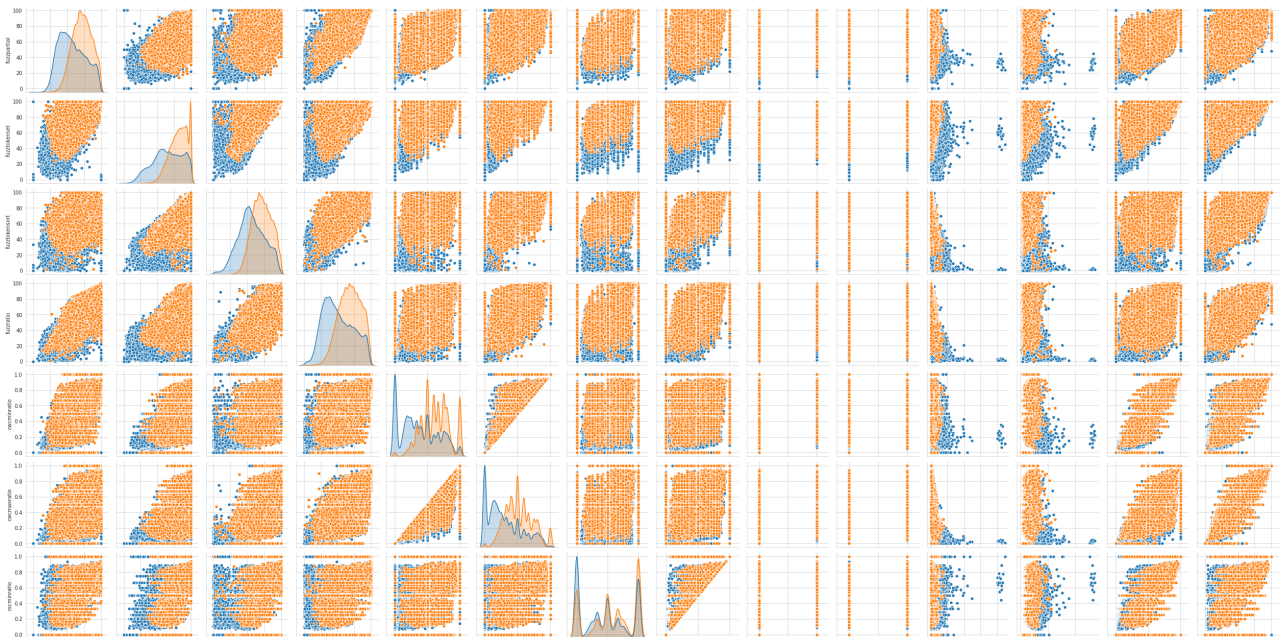
```
Index(['is_duplicate', 'fuzzpartial', 'fuzztokenset', 'fuzztokensort',
       'fuzzratio', 'cwcminratio', 'cwcmaxratio', 'cscminratio', 'cscmaxratio',
       'lwordQual', 'fwordQueal', 'difftokens', 'avgtokens', 'ctcminratio',
       'ctcmaxratio'],
      dtype='object')
```

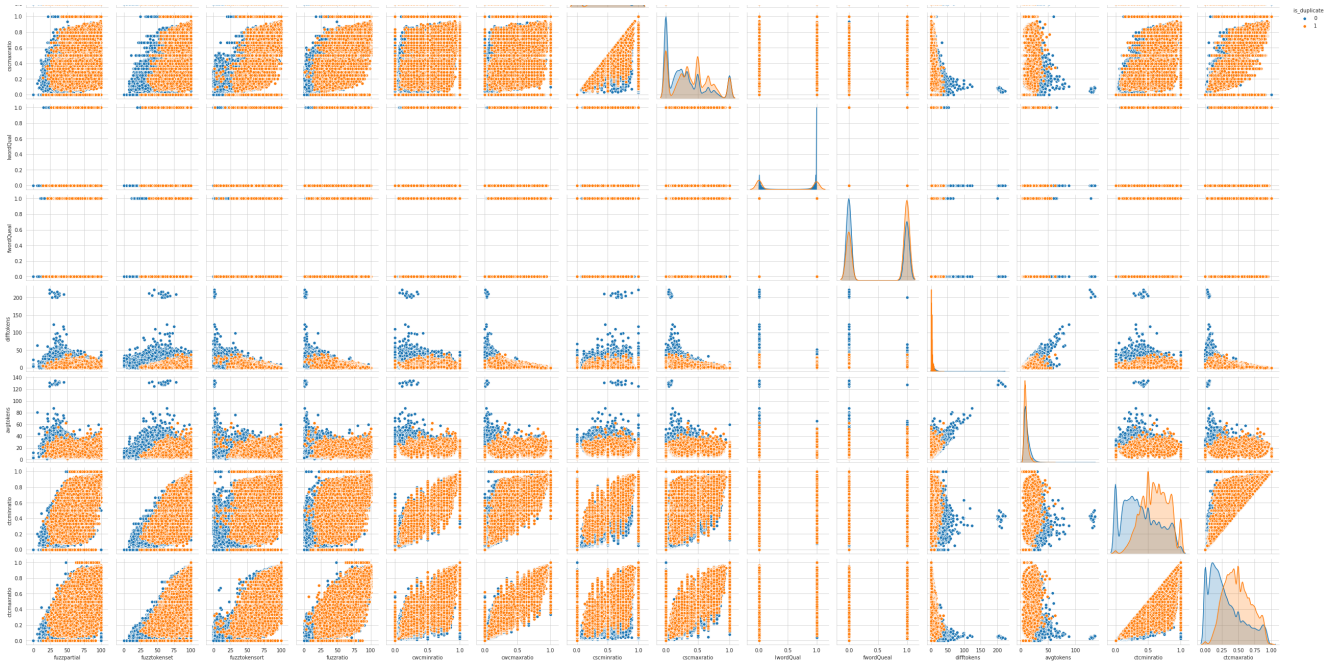
- Lets analyse these features

3.2.3.1 Bi variate analysis

In [0]:

```
sns.pairplot(data=testingfuzzdf2 , hue='is_duplicate')
plt.show()
```





- by looking at above pair plots ctcmin,ctcmax,cwcmax,cwcmin,fuzzratio,fuzzsort,fuzztoken,fuzzpartial are usefull than others in our objective of classification
- by looking at their "scatter and pdf" plots we can see there is some amount of seperation which is an not superb but it is noticable.
- lets Perform TSNE on these all new features

3.2.4 TSNE on all new features

In [0]:

```
tsne_df_withnewfeatures=df[['no_words_in_question1',
       'no_words_in_question2', 'len_of_question1', 'len_of_question2',
       'commonUniqueWords_inBothQuestions', 'frequency_of_question1',
       'frequency_of_question2', 'wordshare', 'fq1+fq2', 'fq1-fq2',
       'total_no_of_words_q1+q2', 'fuzzpartial', 'fuzztokenset',
       'fuzztokensort', 'fuzzratio', 'cwcminratio', 'cwcmaxratio',
       'cscminratio', 'cscmaxratio', 'lwordQual', 'fwordQueal', 'difftokens',
       'avgtokens', 'ctcminratio', 'ctcmaxratio']]
```

In [0]:

```
classLabel=df['is_duplicate']
```

In [0]:

```
# Standardization
standard_scalar=StandardScaler()
```

In [0]:

```
datascaled=standard_scalar.fit_transform(tsne_df_withnewfeatures)
```

In [0]:

```
datascaled.shape
```

Out[0]:

(404290, 25)

(404290, 25)

In [0]:

```
datascaled_1000=datascaled[0:5000 , : ]
```

In [0]:

```
classLabel_1000=classLabel[0:5000]
```

In [0]:

```
tsne=TSNE(n_components=2, perplexity=30.0,  n_iter=1000, init='random', verbose=0, method='barnes_hut', angle=0.5, n_jobs=-1)
```

In [0]:

```
tsnedata=tsne.fit_transform(datascaled_1000)
```

In [0]:

```
tsnedata=tsnedata.T
df_data_tsnedata=np.vstack((tsnedata,classLabel_1000))
```

In [0]:

```
df_data_tsnedata=df_data_tsnedata.T
```

In [0]:

```
df_data_tsnedata.shape
```

Out[0]:

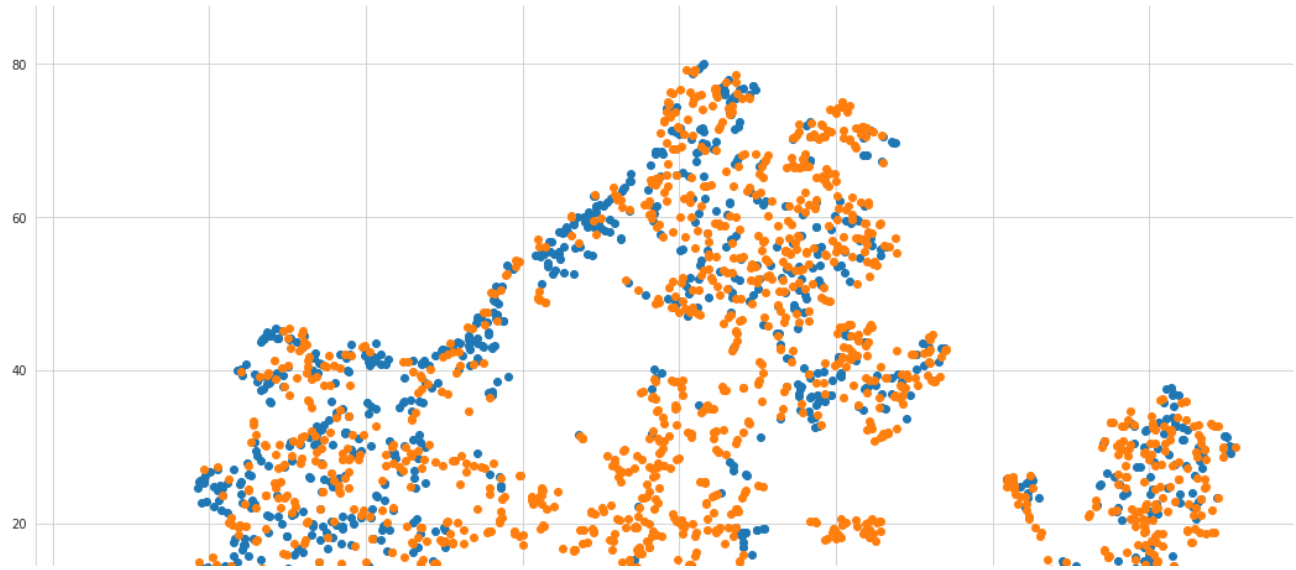
```
(5000, 3)
```

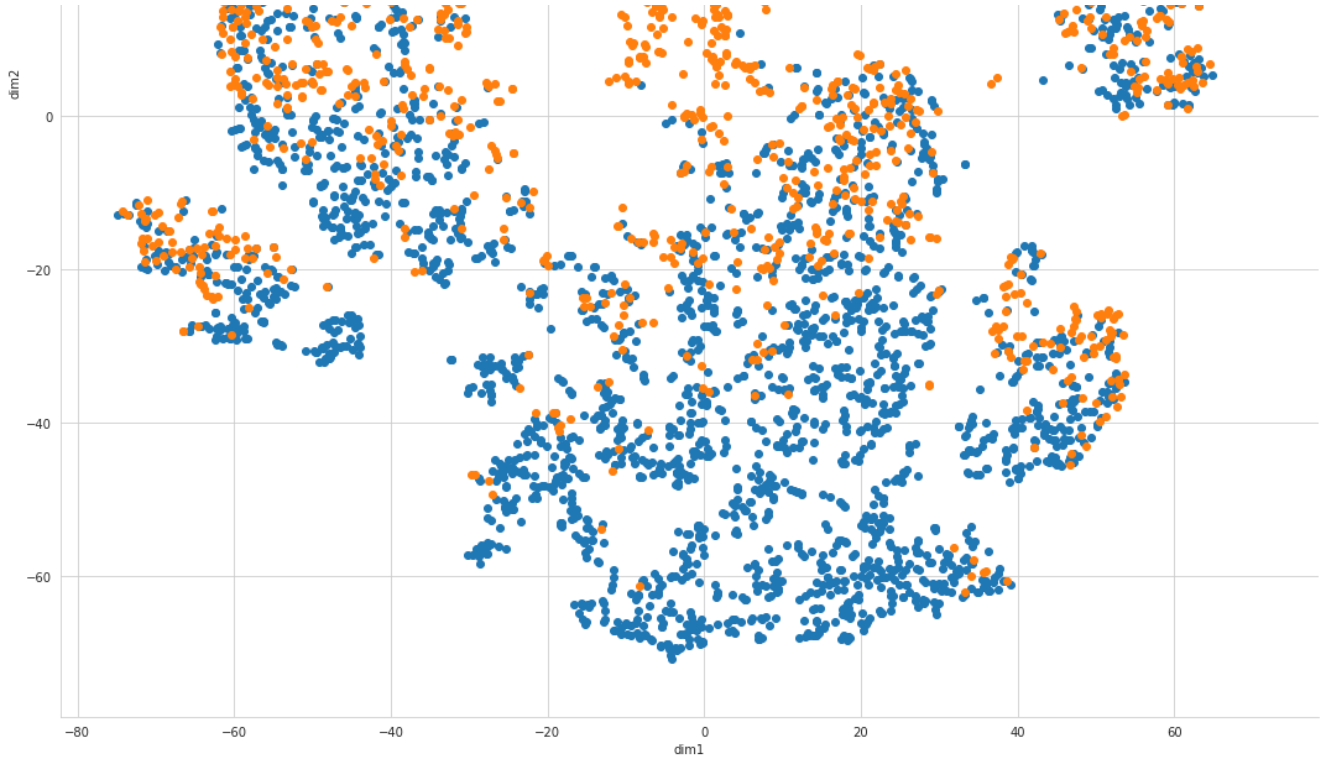
In [0]:

```
df_tsne=pd.DataFrame(df_data_tsnedata , columns=('dim1','dim2','label'))
```

In [0]:

```
sns.FacetGrid(data=df_tsne , hue= 'label' , height = 15)\
    .map(plt.scatter , 'dim1' , 'dim2')

plt.show()
```





- As we can see certainly these features are help ful to some extent in our classification task.
- We are able to distinguish between blue class and orange class by some extent as we took only 5k features.

- lets go to the next phase of data cleaning and converting our text data in to vectors

4. Data Cleaning

In [0]:

```
df.head()
```

Out[0]:

	id	qid1	qid2	question1	question2	is_duplicate	no_words_in_question1	no_words_in_question2	len_of_question1
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	14	12	66
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	8	13	51
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	14	10	73

	id	qid1	qid2	question1	question2	is_duplicate	no_words_in_question1	no_words_in_question2	len_of_question1
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24} [/math] i...	0	11	9	50
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	13	7	76

- If we observe we have questions in text format to be cleaned and should be converted to machine readable form , to create a model.Lets clean the data now.

In [0]:

```
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [0]:

```
cleaned_data_question1=[]

for sentance in df['question1'].values:
  #1.Removing Urls
  sentance=re.sub(r"http\S+" , "" , sentance )
  #2.Removing html tags
  sentance=re.sub(r"<[^<]+?>", "" , sentance )
  #Removing lmxl
  soup = BeautifulSoup(sentance, 'lxml')
  sentance = soup.get_text()
  #3.decontracting phares
  sentance=decontracted(sentance)
  #4.Removing word with numbers
  sentance=re.sub("S*\d\S*" , "" , sentance)
  #5.remove Special charactor punc spaces
  sentance=re.sub(r"\W+", " ", sentance)
  sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in STOPWORDS)
  cleaned_data_question1.append(sentance.strip())
```

```
/usr/local/lib/python3.6/dist-packages/bs4/__init__.py:273: UserWarning:

"b'.'" looks like a filename, not markup. You should probably open this file and pass the filehand
le into Beautiful Soup.
```

In [0]:

```
cleaned_data_question2=[]

for sentance in df['question1'].values:
  #1.Removing Urls
  sentance=re.sub(r"http\S+" , "" , sentance )
  #2.Removing html tags
  sentance=re.sub(r"<[^<]+?>", "" , sentance )
  #3.decontracting phares
```

```
    sentance=decontracted(sentance)
    #4.Removing word with numbers
    sentance=re.sub("S*\d\S*" , "" , sentance)
    #5.remove Special charactor punc spaces
    sentance=re.sub(r"\W+", " ", sentance)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in STOPWORDS)
    cleaned_data_question2.append(sentance.strip())
```

In [0]:

```
df['question1_cleaned']=pd.DataFrame(cleaned_data_question1)
df['question2_cleaned']=pd.DataFrame(cleaned_data_question2)
```

In [0]:

```
df['question2_cleaned'].isna().any()
```

Out[0]:

```
False
```

In [0]:

```
df.isna().any()
```

Out[0]:

```
id                                  False
qid1                                False
qid2                                False
question1                           False
question2                           False
is_duplicate                        False
no_words_in_question1               False
no_words_in_question2               False
len_of_question1                    False
len_of_question2                    False
commonUniqueWords_inBothQuestions   False
frequency_of_question1              False
frequency_of_question2              False
wordshare                           False
fq1+fq2                             False
fq1-fq2                             False
total_no_of_words_q1+q2             False
fuzzpartial                         False
fuzztokenset                        False
fuzztokensort                       False
fuzzratio                           False
cwcminratio                         False
cwcmaxratio                         False
cscminratio                         False
cscmaxratio                         False
lwordQual                           False
fwordQueal                          False
difftokens                          False
avgtokens                           False
ctcminratio                         False
ctcmaxratio                         False
question1_cleaned                   False
question2_cleaned                   False
dtype: bool
```

In [0]:

```
#as we have cleaned questions we can drop these original questions
df=df.drop(columns=['question1','question2'])
```

In [0]:

```
df.isna().any()
```

Out[0]:

```
id                                  False
qid1                                False
qid2                                False
is_duplicate                        False
no_words_in_question1               False
no_words_in_question2               False
len_of_question1                    False
len_of_question2                    False
commonUniqueWords_inBothQuestions   False
frequency_of_question1              False
frequency_of_question2              False
wordshare                           False
fq1+fq2                             False
fq1-fq2                             False
total_no_of_words_q1+q2             False
fuzzpartial                         False
fuzztokenset                        False
fuzztokensort                       False
fuzzratio                           False
cwcminratio                         False
cwcmaxratio                         False
cscminratio                         False
cscmaxratio                         False
lwordQual                           False
fwordQueal                          False
difftokens                          False
avgtokens                           False
ctcminratio                         False
ctcmaxratio                         False
question1_cleaned                   False
question2_cleaned                   False
dtype: bool
```

As we have now cleaned text lets create vectors for it

4.1 Featurization

- taking 75k points due to memory issues.

In [0]:

```
df_75k_datapoints=df.iloc[ 0:75000 , : ]
```

In [0]:

```
df_75k_datapoints.isna().any()
```

Out[0]:

```
id                                  False
qid1                                False
qid2                                False
is_duplicate                        False
no_words_in_question1               False
no_words_in_question2               False
len_of_question1                    False
len_of_question2                    False
commonUniqueWords_inBothQuestions   False
frequency_of_question1              False
frequency_of_question2              False
wordshare                           False
fq1+fq2                             False
fq1-fq2                             False
total_no_of_words_q1+q2             False
fuzzpartial                         False
fuzztokenset                        False
```

```
fuzztokensort                   False
fuzzratio                       False
cwcminratio                     False
cwcmaxratio                     False
cscminratio                     False
cscmaxratio                     False
lwordQual                       False
fwordQueal                      False
difftokens                      False
avgtokens                       False
ctcminratio                     False
ctcmaxratio                     False
question1_cleaned               False
question2_cleaned               False
dtype: bool
```

In [0]:

```
df_75k_datapoints.head()
```

Out[0]:

	id	qid1	qid2	is_duplicate	no_words_in_question1	no_words_in_question2	len_of_question1	len_of_question2	comm
0	0	1	2	0	14	12	66	57	10
1	1	3	4	0	8	13	51	88	4
2	2	5	6	0	14	10	73	59	3
3	3	7	8	0	11	9	50	65	0
4	4	9	10	0	13	7	76	39	2

- Using TFIDF featurization

In [0]:

```
df_tfidf_q1=pd.DataFrame(df_75k_datapoints['question1_cleaned'])
```

In [0]:

```
df_tfidf_q2=pd.DataFrame(df_75k_datapoints['question2_cleaned'])
```

In [0]:

```
df_tfidf_q1[df_tfidf_q1.isna().any(1)]
```

Out[0]:

	question1_cleaned

In [0]:

```
df_tfidf_q2[df_tfidf_q2.isna().any(1)]
```

Out[0]:

question2_cleaned

In [0]:

```
vectorizer=TfidfVectorizer(ngram_range=(1,2), min_df=10 , max_features = 5000 )
```

In [0]:

```
data_Q1_vector=vectorizer.fit_transform(df_tfidf_q1['question1_cleaned'])
```

In [0]:

```
data_narray_1=data_Q1_vector.toarray()
```

In [0]:

```
df_q1_vector_pd=pd.DataFrame(data_narray_1)
```

In [0]:

```
df_q1_vector_pd.to_csv('dataframe_of_q1_vectors_75kand5kFeatures.csv')
```

In [0]:

```
data_Q2_vector=vectorizer.fit_transform(df_tfidf_q2['question2_cleaned'])
```

In [0]:

```
data_narray_2=data_Q2_vector.toarray()
```

In [0]:

```
df_q2_vector_pd=pd.DataFrame(data_narray_2)
```

In [0]:

```
df_q2_vector_pd.to_csv('dataframe_of_q2_vectors_75kand5kFeatures.csv')
```

In [0]:

```
print(df_q2_vector_pd.shape)
print(df_q1_vector_pd.shape)
```

```
(75000, 5000)
(75000, 5000)
```

In [0]:

```
df_q1_vector_pd.head()
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5000 columns

5 rows × 5000 columns

In [0]:

```
df_q2_vector_pd.head()
```

Out[0]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 5000 columns

- Lets combine this dataframes and original data frame.

In [0]:

```
# Reading the file
df_75k_datapoints = pd.read_csv (
'/content/df_100k_datapoints_with_allfeaturesexcptq1andq1tfidf.csv')
```

In [0]:

```
# Reading the file
df_q1_vector_pd = pd.read_csv('/content/dataframe_of_q1_vectors_75kand5kFeatures.csv')
```

In [0]:

```
# Reading the file
df_q2_vector_pd = pd.read_csv('/content/dataframe_of_q1_vectors_75kand5kFeatures.csv')
```

In [0]:

```
combined_dataFrameOf_q1nq2=pd.concat([df_q1_vector_pd,df_q2_vector_pd]  , axis=1)
```

In [0]:

```
# Saving the df to csv file
combined_dataFrameOf_q1nq2.to_csv('combined_df_q1q2_75kand5k.csv')
```

In [0]:

```
combined_dataFrameOf_q1nq2.columns
```

Out[0]:

```
Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
            ...
            4990, 4991, 4992, 4993, 4994, 4995, 4996, 4997, 4998, 4999],
           dtype='int64', length=10000)
```

In [0]:

```
# combining the original df and df of q1 n q2
final_data_frame_with_allFeatures=pd.concat([df_75k_datapoints,combined_dataFrameOf_q1nq2],axis=1)
```

In [0]:

```
# Saving the df to csv file
```

```
# Saving the df to csv file
final_data_frame_with_allFeatures.to_csv('FinalDataFrameWith75kdatapointsand10035.csv')
```

In [0]:

```
final_data_frame_with_allFeatures.shape
```

Out[0]:

```
(75000, 10031)
```

In [0]:

```
# Reading the file
final_data_frame_with_allFeatures=pd.read_csv("/content/FinalDataFrameWith75kdatapointsand10035.csv
)
```

In [0]:

```
final_data_frame_with_allFeatures.columns
```

Out[0]:

```
Index(['Unnamed: 0', 'id', 'qid1', 'qid2', 'is_duplicate',
       'no_words_in_question1', 'no_words_in_question2', 'len_of_question1',
       'len_of_question2', 'commonUniqueWords_inBothQuestions',
       ...
       '4990.1', '4991.1', '4992.1', '4993.1', '4994.1', '4995.1', '4996.1',
       '4997.1', '4998.1', '4999.1'],
      dtype='object', length=10032)
```

In [0]:

```
remove_df=final_data_frame_with_allFeatures
```

In [0]:

```
final_data_75kn5k=final_data_frame_with_allFeatures
```

In [0]:

```
remove_df=remove_df.drop(columns=['0','qid1','qid2','id','0.1','question1_cleaned','question2_clean
ed'])
```

In [0]:

```
remove_df=remove_df.drop(columns='Unnamed: 0' ,axis=0)
```

In [0]:

```
remove_df.head()
```

Out[0]:

	is_duplicate	no_words_in_question1	no_words_in_question2	len_of_question1	len_of_question2	commonUniqueWor
0	0	14	12	66	57	10
1	0	8	13	51	88	4
2	0	14	10	73	59	3
3	0	11	9	50	65	0
4	0	13	7	76	39	2

5 rows × 10024 columns

In [0]:

```
Final_data_frame_Complete=remove_df
```

In [0]:

```
Final_data_frame_Complete.head()
```

Out[0]:

	is_duplicate	no_words_in_question1	no_words_in_question2	len_of_question1	len_of_question2	commonUniqueWor
0	0	14	12	66	57	10
1	0	8	13	51	88	4
2	0	14	10	73	59	3
3	0	11	9	50	65	0
4	0	13	7	76	39	2

5 rows × 10024 columns

In [0]:

```
# writing to csv file
Final_data_frame_Complete.to_csv("completed75kand1024Features.csv")
```

In [0]:

```
Final_data_frame_Complete.shape
```

Out[0]:

```
(75000, 10024)
```

In [0]:

```
import pandas as pd
```

In [0]:

```
#Reading the csv file
Final_data_frame_Complete= pd.read_csv('/content/completed75kand1024Features.csv')
```

In [0]:

```
Final_data_frame_Complete=Final_data_frame_Complete.drop(columns='Unnamed: 0' )
```

In [0]:

```
Final_data_frame_Complete.to_csv('Final.csv')
```

- As we have our final dataframe for modeling lets create models.

4.2 Data Spliting

In [0]:

```
backup_complete=Final_data_frame_Complete
```

In [0]:

```
Final_data_frame_Complete.columns
```

Out[0]:

```
Index(['is_duplicate', 'no_words_in_question1', 'no_words_in_question2',
       'len_of_question1', 'len_of_question2',
       'commonUniqueWords_inBothQuestions', 'frequency_of_question1',
       'frequency_of_question2', 'wordshare', 'fq1+fq2',
       ...
       '4990.1', '4991.1', '4992.1', '4993.1', '4994.1', '4995.1', '4996.1',
       '4997.1', '4998.1', '4999.1'],
      dtype='object', length=10024)
```

In [0]:

```
y=Final_data_frame_Complete['is_duplicate']
```

In [0]:

```
type(y)
```

Out[0]:

```
pandas.core.series.Series
```

In [0]:

```
#DependentVariable
y.shape
```

Out[0]:

```
(75000,)
```

In [0]:

```
#Independent Variable  ( features represented as vectors )
X=backup_complete.drop(columns='is_duplicate')
```

In [0]:

```
X.head()
```

Out[0]:

	no_words_in_question1	no_words_in_question2	len_of_question1	len_of_question2	commonUniqueWords_inBothQue
0	14	12	66	57	10
1	8	13	51	88	4
2	14	10	73	59	3
3	11	9	50	65	0
4	13	7	76	39	2

5 rows × 10023 columns

In [0]:

```
y.head()
```

Out[0]:

```
0    0
```

```
1    0
2    0
3    0
4    0
Name: is_duplicate, dtype: int64
```

- As we have our X and y Lets split them accordingly and create CV test and train datasets

In [0]:

```
X.to_csv('XFinal.csv')
y.to_csv('y(1).csv')
```

In [0]:

```
#Loading the X and y for modeling
```

In [0]:

```
X=pd.read_csv("/content/drive/My Drive/XFinal.csv")
```

In [0]:

```
y=pd.read_csv("/content/y(1).csv")
```

In [0]:

```
y=y['is_duplicate'].values
```

In [0]:

```
X=X.drop(columns='Unnamed: 0')
```

In [0]:

```
X.head()
```

Out[0]:

	no_words_in_question1	no_words_in_question2	len_of_question1	len_of_question2	commonUniqueWords_inBothQue
0	14	12	66	57	10
1	8	13	51	88	4
2	14	10	73	59	3
3	11	9	50	65	0
4	13	7	76	39	2

5 rows × 10023 columns

In [0]:

```
X_train,x_test,y_train,y_test=train_test_split(X,y, stratify=y, test_size=0.2)
```

In [0]:

```
X_train,x_cv,y_train,y_cv=train_test_split(X_train,y_train, stratify=y_train , test_size=0.2)
```

- As we have split the data to for our modelling lets see the size.

In [0]:

```
print ( X_train.shape,y_train.shape)
print( x_cv.shape,y_cv.shape)
print(x_test.shape,y_test.shape)
```

```
(48000, 10023) (48000,)
(12000, 10023) (12000,)
(15000, 10023) (15000,)
```

- Now we have to perform modeling, We can create a dummy model and compare our model metric with its .. and our choosen metric was logloss.

In [0]:

```
# dummy model log loss

length_y=len(y)
```

In [0]:

```
my_array=np.zeros((length_y,2))
print(my_array.shape)
```

```
(75000, 2)
```

In [0]:

```
my_array
```

Out[0]:

```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       ...,
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [0]:

```
for row in range(len(y_test)):
  random_element=np.random.rand(1,2)
  my_array[row] = (random_element/np.sum(random_element))[0]
```

In [0]:

```
predicted_y=(np.argmax(my_array , axis=1))
```

In [0]:

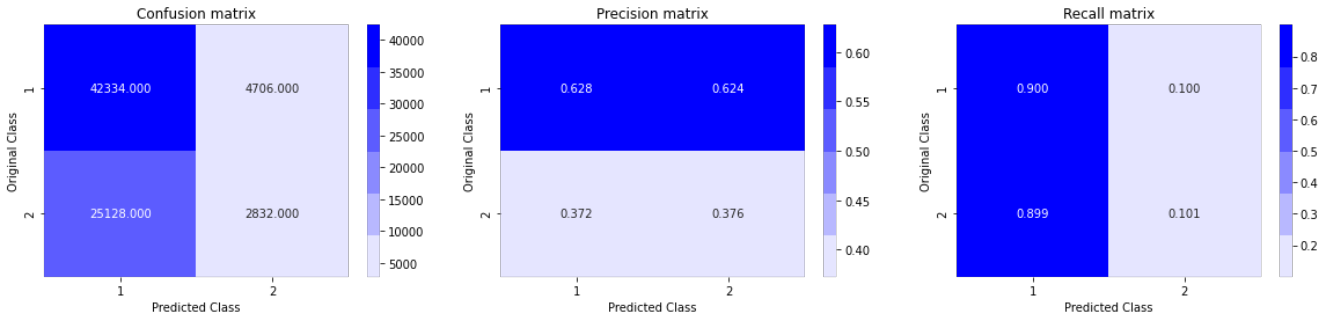
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

    plt.subplot(1, 3, 2)
    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]:

```
print(" the log loss of random model is : {} ".format( log_loss(y,predicted_y)))

print(" the confusion metrix , precission matrix and recall matrix is: " .format(
plot_confusion_matrix(y,predicted_y)))
```

```
 the log loss of random model is : 13.739114904950291
```




```
 the confusion metrix , precission matrix and recall matrix is:
```

- We will take this as as the worst case scenario and build our models such that we get logloss lessthan random model.And good confusion metrics scores.

4.3 Linear SVM algorithm

- As we have data lets do hypertuning to find best parameters

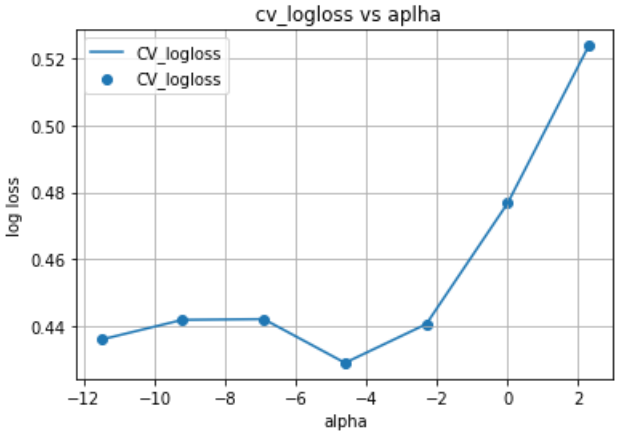
In [0]:

```
alpha= [ 10**x for x in range(-5,2)]
print(alpha)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10]
```

In [0]:

```
logLos=[ ]

for i in  alpha:

  model=SGDClassifier(loss='hinge',penalty='l2',alpha=i, n_jobs=-1 , class_weight = 'balanced')
  sig_clf = CalibratedClassifierCV(model, method="sigmoid")
  sig_clf.fit(X_train, y_train)

  pred_prob=sig_clf.predict_proba(x_cv) [ : , 1]

  logLos.append( log_loss( y_cv , pred_prob) )



plt.plot(np.log(alpha) , logLos  , label = 'CV_logloss')
plt.scatter(np.log(alpha) , logLos , label = 'CV_logloss' )
plt.xlabel('alpha')
plt.ylabel(" log loss ")
plt.grid('white')
plt.legend()
plt.title(" cv_logloss vs aplha")
plt.show()
```



- We can refer that from the figure the log loss is less for aplha = 0.01

In [0]:

```
best_aplha_index=  np.argmin(np.array(logLos))
best_alpha=alpha[best_aplha_index]
```

In [0]:

```
print( " the minimum Logg loss is for aplha {} and its corresponding loss loss is {} :".format( be
st_alpha,min(logLos)))
```

```
 the minimum Logg loss is for aplha 0.01 and its corresponding loss loss is 0.42914430047414576 :
```

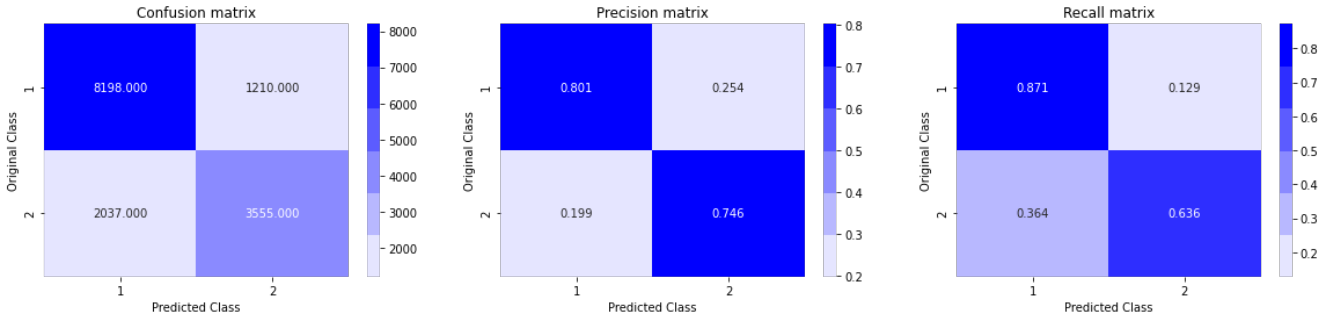
- Lets Test on the test data and plot confusion matrix and log loss and other metrics

In [0]:

```
model=SGDClassifier(loss = 'hinge' , penalty = 'l2' alpha= best_alpha , n_jobs= 1 , class_weight= '
```

```
model=SGDClassifier(loss = 'hinge' , penalty = 'l2',alpha= best_alpha , n_jobs=-1 , class_weight=
balanced')
sig_clf = CalibratedClassifierCV(model, method="sigmoid")
sig_clf.fit(X_train, y_train)
predicted_y= sig_clf.predict_proba(x_test)[: , 1]
print("The log loss for this aplha = 0.01 is {}".format(log_loss(y_test,predicted_y)))
#*************************************************************
print("*************************************************************")
y_predicted_test=sig_clf.predict_proba(x_test)
y_pred_test=np.argmax(y_predicted_test , axis=1)
plot_confusion_matrix(y_test,y_pred_test)
```

```
The log loss for this aplha = 0.01 is 0.43185663791842827
*************************************************************
```



- Observations from the above:-
- Log loss is 0.4318 when compared to random model it is way better
- TNR , TPR , FPR , FNR := 80.1 , 74.7 , 19.7 ,25.1
- Precission and Recall also looking good.

4.3 Logistic Regression Algorithm

- Lets Hyperparameter tune to fine best alpha

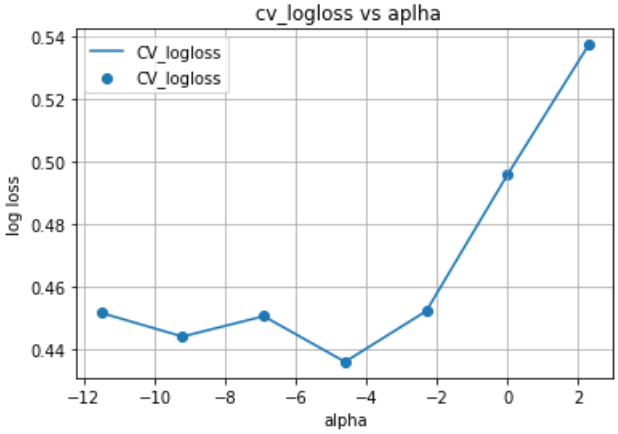
In [0]:

```
alpha= [ 10**x for x in range(-5,2)]
print(alpha)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10]
```

In [0]:

```
logLos=[ ]

for i in  alpha:

  model=SGDClassifier(loss='log',penalty='l2',alpha=i, n_jobs=-1 , class_weight = 'balanced')
  sig_clf = CalibratedClassifierCV(model, method="sigmoid")
  sig_clf.fit(X_train, y_train)

  pred_prob=sig_clf.predict_proba(x_cv) [ : , 1]

  logLos.append( log_loss( y_cv , pred_prob) )



plt.plot(np.log(alpha) , logLos  , label = 'CV_logloss')
plt.scatter(np.log(alpha) , logLos , label = 'CV_logloss' )
plt.xlabel('alpha')
plt.ylabel(" log loss ")
plt.grid('white')
plt.legend()
plt.title(" cv_logloss vs aplha")
plt.show()
```



In [0]:

```
best_aplha_index=  np.argmin(np.array(logLos))
best_alpha=alpha[best_aplha_index]
```

In [0]:

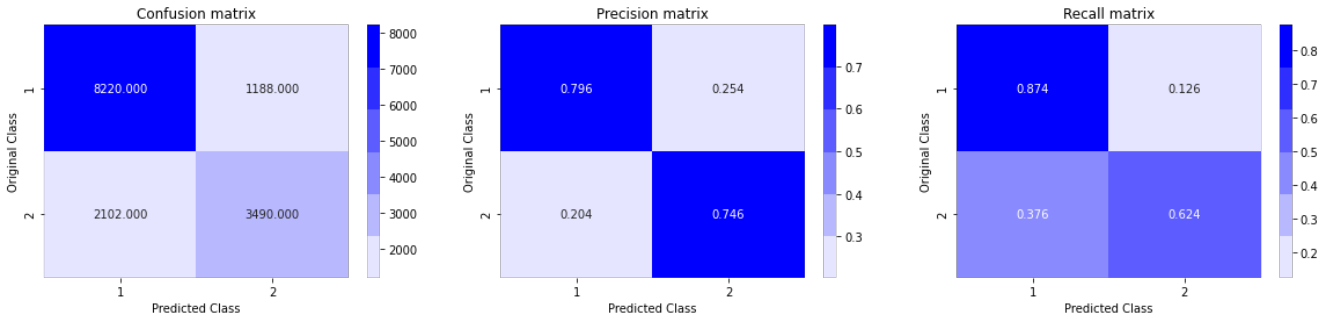
```
print( " the minimum Logg loss is for aplha {} and its corresponding loss loss is {} :".format( best_alpha,min(logLos)))
```

```
 the minimum Logg loss is for aplha 0.01 and its corresponding loss loss is 0.4359580380874459 :
```

In [0]:

```
model=SGDClassifier(loss = 'log' , penalty = 'l2',alpha= best_alpha , n_jobs=-1 , class_weight= 'balanced')
sig_clf = CalibratedClassifierCV(model, method="sigmoid")
sig_clf.fit(X_train, y_train)
predicted_y= sig_clf.predict_proba(x_test)[: , 1]
print("The log loss for this aplha = 0.01 is {}".format(log_loss(y_test,predicted_y)))
#*************************************************************
print("*************************************************************")
y_predicted_test=sig_clf.predict_proba(x_test)
y_pred_test=np.argmax(y_predicted_test , axis=1)
plot_confusion_matrix(y_test,y_pred_test)
```

```
The log loss for this aplha = 0.01 is 0.42867104563430924
*************************************************************
```



- Observations from the above:-
- Log loss is 0.4286 when compared to random model it is way better
- TNR , TPR , FPR , FNR := 79.6 , 74.6 , 20.3 ,25.3
- Precission and Recall also looking good.

5.0 Results

- using pretty table library

In [0]:

```
from prettytable import PrettyTable
table = PrettyTable()

table.field_names = ["Vectorizer","classifier used","Hyper Parameter",  "LogLoss"]
table.add_row(["array","random Model","null",13])
table.add_row(["TFIDF","LogisticRegression",0.01,0.4286])
table.add_row(["TFIDF","Linear SVM",0.01,0.4318])
print(table)
```

```
+------------+--------------------+-----------------+---------+
| Vectorizer |  classifier used   | Hyper Parameter | LogLoss |
+------------+--------------------+-----------------+---------+
|   array    |    random Model    |       null      |    13   |
|   TFIDF    | LogisticRegression |       0.01      |  0.4286 |
|   TFIDF    |     Linear SVM     |       0.01      |  0.4318 |
+------------+--------------------+-----------------+---------+
```

- We can notice logistic regresion performed better than all we can infer from the result table.Linear SVM also performed Good.