

Quora Question pair similarity

July 11, 2018

Quora Question Pairs

1. Business Problem

1.1 Description

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

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

> Credits: Kaggle

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

1.2 Sources/Useful Links

- Source : <https://www.kaggle.com/c/quora-question-pairs> ___ 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/AACZ...>
- 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 Problem

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Learning 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 [1]: 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 warnings
warnings.filterwarnings("ignore")
# This package is used for finding longest common subsequence between two strings
# you can write your own dp code for this
import distance
from nltk.stem import PorterStemmer
```

```

from bs4 import BeautifulSoup
from fuzzywuzzy import fuzz
from sklearn.manifold import TSNE
# Import the Required lib packages for WORD-Cloud generation
# https://stackoverflow.com/questions/45625434/how-to-install-wordcloud-in-python3-6
from wordcloud import WordCloud, STOPWORDS
from os import path
from PIL import Image

```

3.1 Reading data and basic stats

```
In [2]: df = pd.read_csv("train.csv")
```

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

Number of data points: 404290

```
In [3]: df.head()
```

```
Out[3]:
```

	id	qid1	qid2	question1 \
0	0	1	2	What is the step by step guide to invest in sh...
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...
2	2	5	6	How can I increase the speed of my internet co...
3	3	7	8	Why am I mentally very lonely? How can I solve...
4	4	9	10	Which one dissolve in water quickly sugar, salt...

	question2	is_duplicate
0	What is the step by step guide to invest in sh...	0
1	What would happen if the Indian government sto...	0
2	How can Internet speed be increased by hacking...	0
3	Find the remainder when 23^{24} is divided by 1000...	0
4	Which fish would survive in salt water?	0

```
In [4]: 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:

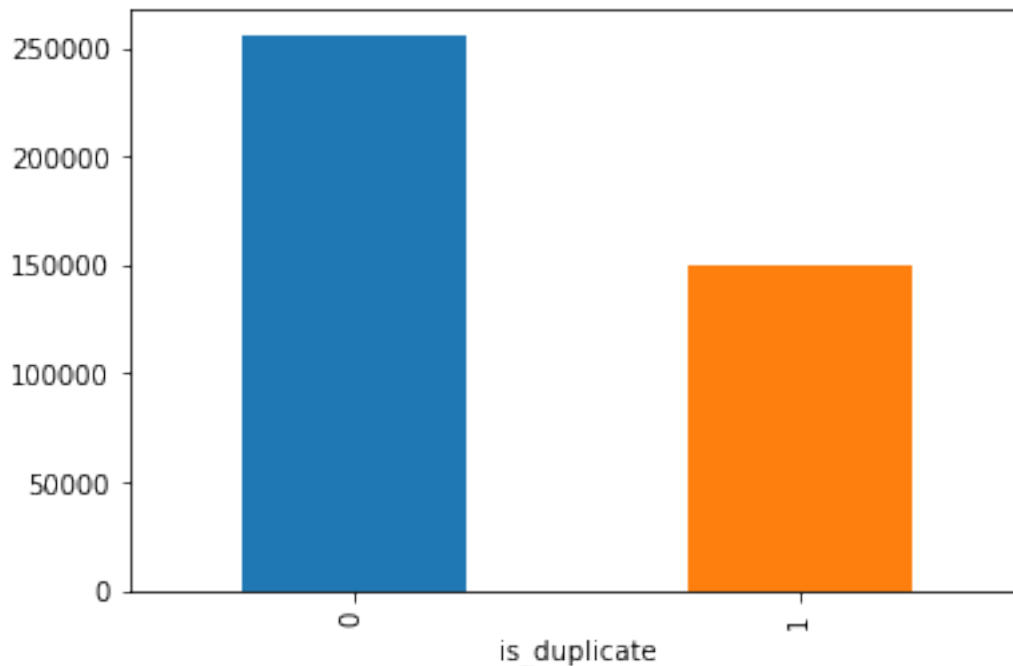
- id: Looks like a simple rowID
- qid{1, 2}: The unique ID of each question in the pair
- question{1, 2}: The actual textual contents of the questions.
- is_duplicate: The label that we are trying to predict - whether the two questions are duplicates of each other.

3.2.1 Distribution of data points among output classes

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

```
In [5]: df.groupby("is_duplicate")["id"].count().plot.bar()
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x151a3cfc04a8>
```



```
In [6]: print('~> Total number of question pairs for training:\n {}'.format(len(df)))
```

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

```
In [7]: print('~> Question pairs are not Similar (is_duplicate = 0):\n {}'.format(100 - round(
        print('\n~> Question pairs are Similar (is_duplicate = 1):\n {}'.format(round(df['is_duplicate'] == 1).count() / len(df) * 100))))
```

```
~> 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 [8]: 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: {}'.format(unique_qs))
        #print len(np.unique(qids))

        print ('Number of unique questions that appear more than one time: {} ({}%)\n'.format(
        print ('Max number of times a single question is repeated: {}'.format(max(qids.value_

        q_vals=qids.value_counts()

        q_vals=q_vals.values
```

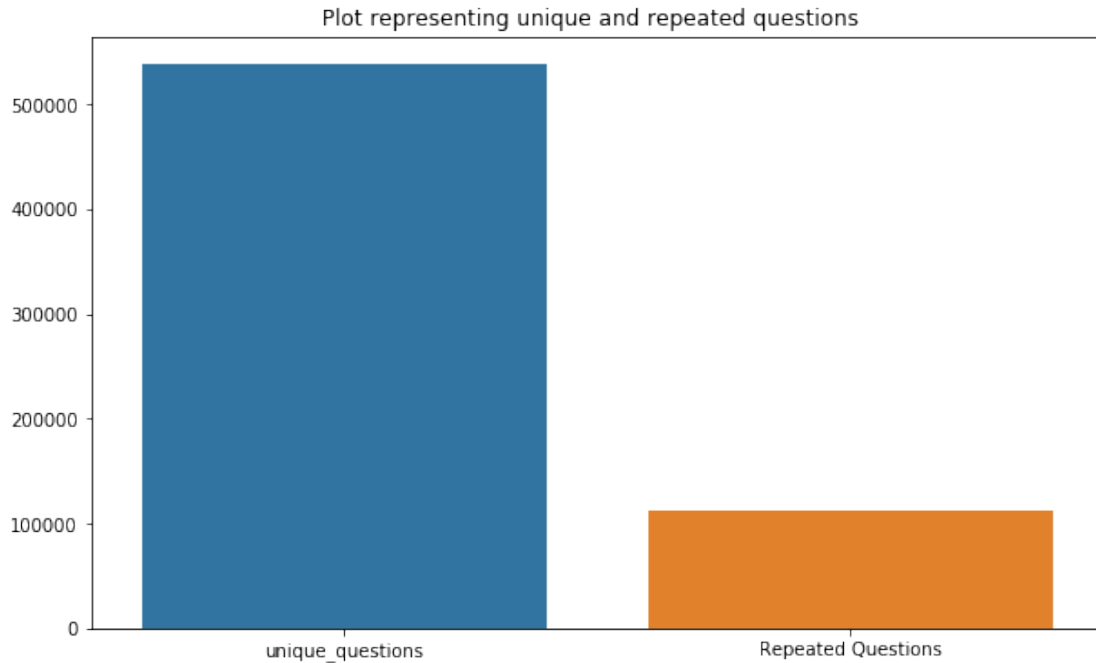
Total number of Unique Questions are: 537933

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

Max number of times a single question is repeated: 157

```
In [9]: 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 [10]: *#checking whether there are any repeated pair of questions*

```
pair_duplicates = df[['qid1', 'qid2', 'is_duplicate']].groupby(['qid1', 'qid2']).count()
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 [11]: `plt.figure(figsize=(20, 10))`

```
plt.hist(qids.value_counts(), bins=160)
```

```
plt.yscale('log', nonposy='clip')
```

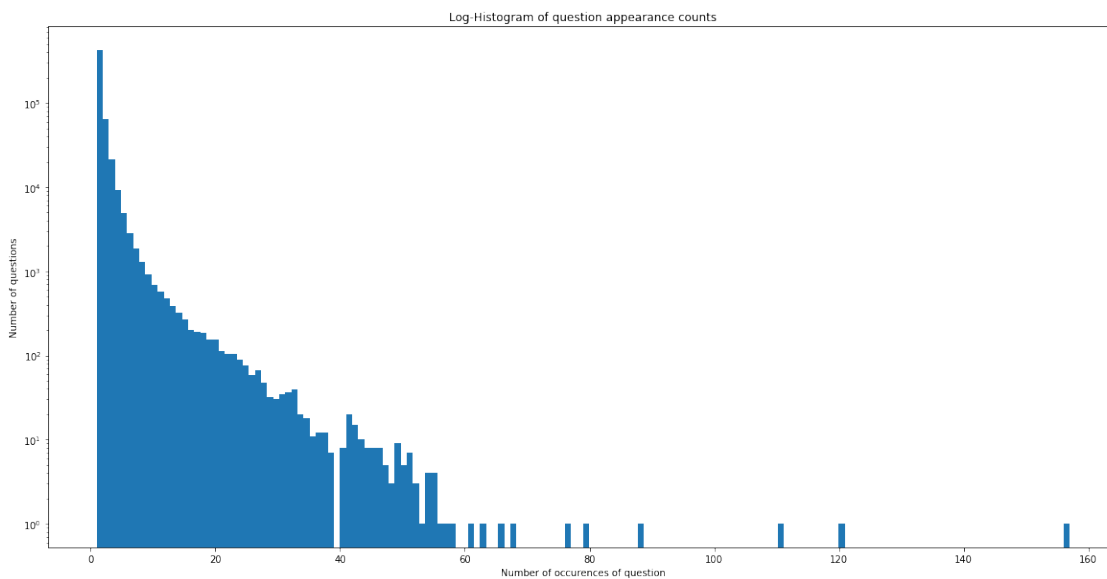
```
plt.title('Log-Histogram of question appearance counts')
```

```
plt.xlabel('Number of occurrences of question')
```

```
plt.ylabel('Number of questions')
```

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

Maximum number of times a single question is repeated: 157



3.2.5 Checking for NULL values

In [12]: *#Checking whether there are any rows with null values*

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

	id	qid1	qid2	question1	\
105780	105780	174363	174364	How can I develop android app?	
201841	201841	303951	174364	How can I create an Android app?	
363362	363362	493340	493341	NaN	

	question2	is_duplicate
105780	NaN	0
201841	NaN	0
363362	My Chinese name is Haichao Yu. What English na...	0

- There are two rows with null values in question2

```
In [13]: # Filling the null values with ' '
```

```
df = df.fillna('')
nan_rows = df[df.isnull().any(1)]
print (nan_rows)
```

Empty DataFrame

```
Columns: [id, gid1, gid2, question1, question2, is_duplicate]
```

Index: []

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like: - `freq_qid1` = Frequency of qid1's - `freq_qid2` = Frequency of qid2's - `q1len` = Length of q1 - `q2len` = Length of q2 - `q1_n_words` = Number of words in Question 1 - `q2_n_words` = Number of words in Question 2 - `word_Common` = (Number of common unique words in Question 1 and Question 2) - `word_Total` = (Total num of words in Question 1 + Total num of words in Question 2) - `word_share` = (word_common)/(word_Total) - `freq_q1+freq_q2` = sum total of frequency of qid1 and qid2 - `freq_q1-freq_q2` = absolute difference of frequency of qid1 and qid2

```
In [14]: if os.path.isfile('df_fe_without_preprocessing_train.csv'):
        df = pd.read_csv("df_fe_without_preprocessing_train.csv", encoding='latin-1')
    else:
        df['freq_qid1'] = df.groupby('qid1')['qid1'].transform('count')
        df['freq_qid2'] = df.groupby('qid2')['qid2'].transform('count')
        df['q1len'] = df['question1'].str.len()
        df['q2len'] = df['question2'].str.len()
        df['q1_n_words'] = df['question1'].apply(lambda row: len(row.split(" ")))
        df['q2_n_words'] = df['question2'].apply(lambda row: len(row.split(" ")))

        def normalized_word_Common(row):
            w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
            w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
            return 1.0 * len(w1 & w2)
        df['word_Common'] = df.apply(normalized_word_Common, axis=1)

        def normalized_word_Total(row):
            w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
            w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
            return 1.0 * (len(w1) + len(w2))
        df['word_Total'] = df.apply(normalized_word_Total, axis=1)

        def normalized_word_share(row):
            w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))
            w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))
            return 1.0 * len(w1 & w2)/(len(w1) + len(w2))
        df['word_share'] = df.apply(normalized_word_share, axis=1)

        df['freq_q1+q2'] = df['freq_qid1']+df['freq_qid2']
        df['freq_q1-q2'] = abs(df['freq_qid1']-df['freq_qid2'])

        df.to_csv("df_fe_without_preprocessing_train.csv", index=False)

    df.head()
```

```
Out[14]:
```

	id	qid1	qid2	question1 \
0	0	1	2	What is the step by step guide to invest in sh...
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...


```

2 2 5 6 How can I increase the speed of my internet co...
3 3 7 8 Why am I mentally very lonely? How can I solve...
4 4 9 10 Which one dissolve in water quickly sugar, salt...

```

	question2	is_duplicate	freq_qid1	\
0	What is the step by step guide to invest in sh...	0	1	
1	What would happen if the Indian government sto...	0	4	
2	How can Internet speed be increased by hacking...	0	1	
3	Find the remainder when 23^{24} i...	0	1	
4	Which fish would survive in salt water?	0	3	

	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	\
0	1	66	57	14	12	10.0	23.0	
1	1	51	88	8	13	4.0	20.0	
2	1	73	59	14	10	4.0	24.0	
3	1	50	65	11	9	0.0	19.0	
4	1	76	39	13	7	2.0	20.0	

	word_share	freq_q1+q2	freq_q1-q2
0	0.434783	2	0
1	0.200000	5	3
2	0.166667	2	0
3	0.000000	2	0
4	0.100000	4	2

3.3.1 Analysis of some of the extracted features

- Here are some questions have only one single words.

```

In [15]: print ("Minimum length of the questions in question1 : " , min(df['q1_n_words']))

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

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

```

```

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

```

3.3.1.1 Feature: word_share

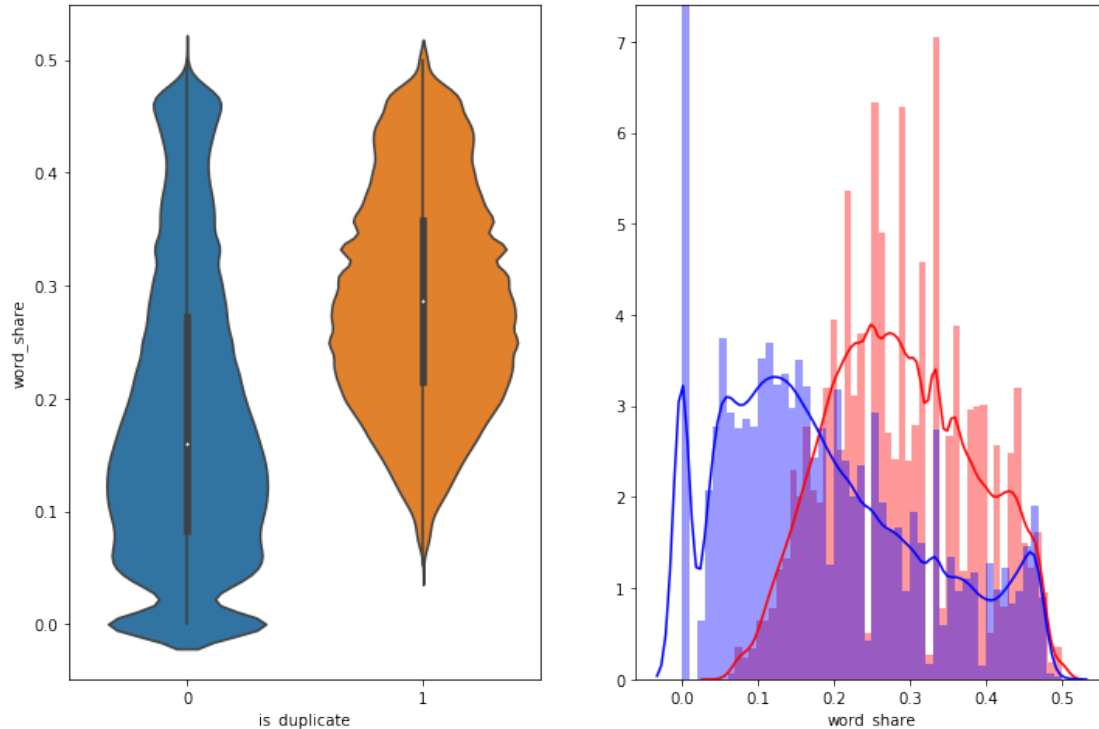
```

In [16]: plt.figure(figsize=(12, 8))

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

```

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



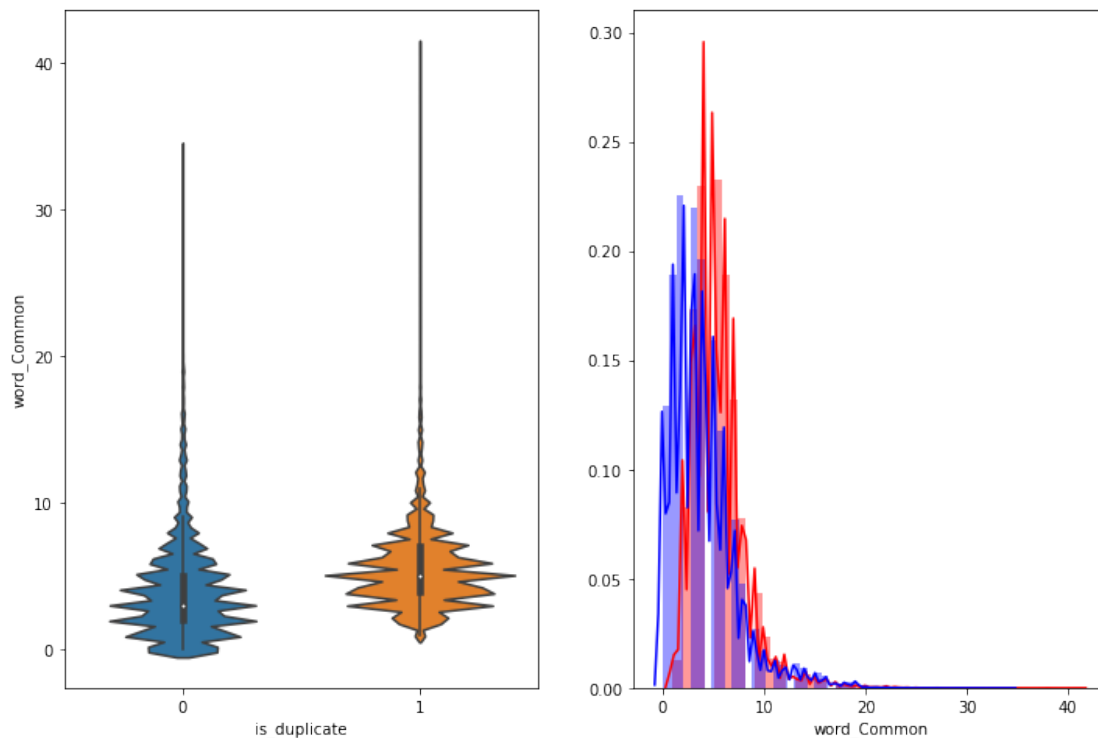
- The distributions for normalized word_share have some overlap on the far right-hand side, i.e., there are quite a lot of questions with high word similarity
- The average word share and Common no. of words of qid1 and qid2 is more when they are duplicate(Similar)

3.3.1.2 Feature: word_Common

```
In [17]: 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 = 'r')
sns.distplot(df[df['is_duplicate'] == 0.0]['word_Common'][0:], label = "0", color = 'b')
plt.show()
```



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

```
In [18]: #https://stackoverflow.com/questions/12468179/unicodedecodeerror-utf8-codec-cant-deco
if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    df = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
    df = df.fillna('')
    df.head()
else:
    print("get df_fe_without_preprocessing_train.csv from drive or run the previous n
```

```
In [19]: df.head(2)
```

```
Out[19]:
```

	id	qid1	qid2	question1	\
0	0	1	2	What is the step by step guide to invest in sh...	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	

		question2	is_duplicate	freq_qid1	\
0	What is the step by step guide to invest in sh...		0	1	
1	What would happen if the Indian government sto...		0	4	

	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	\
0	1	66	57	14	12	10.0	23.0	
1	1	51	88	8	13	4.0	20.0	

	word_share	freq_q1+q2	freq_q1-q2
0	0.434783	2	0
1	0.200000	5	3

3.4 Preprocessing of Text

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

```
In [2]: # To get the results in 4 decemal points
SAFE_DIV = 0.0001
```

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

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

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

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

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

    return x
```

- 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 length 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 length 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 length 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 length 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 length 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 length of token count of Q1 and Q2 $ctc_max = common_token_count / (\max(len(q1_tokens), len(q2_tokens)))$
- **last_word_eq** : Check if Last 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 length of token count of Q1 and Q2 $longest_substr_ratio = len(longest\ common\ substring) / (\min(len(q1_tokens), len(q2_tokens)))$

```
In [21]: def get_token_features(q1, q2):
          token_features = [0.0]*10

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

          if len(q1_tokens) == 0 or len(q2_tokens) == 0:
```

```

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

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

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

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

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

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

    # 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.lcs substrings(a, b))
    if len(strs) == 0:
        return 0
    else:
        return len(strs[0]) / (min(len(a), len(b)) + 1)

def extract_features(df):

```

```

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

print("token features...")

# Merging Features with dataset

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

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

#Computing Fuzzy Features and Merging with Dataset

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

df["token_set_ratio"] = df.apply(lambda x: fuzz.token_set_ratio(x["question1"], x["question2"]), axis=1)
# The token sort approach involves tokenizing the string in question, sorting the tokens
# then joining them back into a string We then compare the transformed strings with the other string
df["token_sort_ratio"] = df.apply(lambda x: fuzz.token_sort_ratio(x["question1"], x["question2"]), axis=1)
df["fuzz_ratio"]       = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), axis=1)
df["fuzz_partial_ratio"] = df.apply(lambda x: fuzz.partial_ratio(x["question1"], x["question2"]), axis=1)
df["longest_substr_ratio"] = df.apply(lambda x: fuzz.longest_common_substring_ratio(x["question1"], x["question2"]), axis=1)
return df

In [27]: if os.path.isfile('nlp_features_train.csv'):
df = pd.read_csv("nlp_features_train.csv", encoding='latin-1')
df.fillna('')
else:
    print("Extracting features for train:")
    df = pd.read_csv("train.csv")
    df = extract_features(df)
    df.to_csv("nlp_features_train.csv", index=False)
df.head(2)

Extracting features for train:
token features...

```

fuzzy features..

```
Out[27]:
```

	id	qid1	qid2	question1	\
0	0	1	2	what is the step by step guide to invest in sh...	
1	1	3	4	what is the story of kohinoor	koh i noor dia...

		question2	is_duplicate	cwc_min	\
0	what is the step by step guide to invest in sh...		0	0.999980	
1	what would happen if the indian government sto...		0	0.799984	

	cwc_max	csc_min	csc_max	...	ctc_max	last_word_eq	\
0	0.833319	0.999983	0.999983	...	0.785709	0.0	
1	0.399996	0.749981	0.599988	...	0.466664	0.0	

	first_word_eq	abs_len_diff	mean_len	token_set_ratio	token_sort_ratio	\
0		1.0	2.0	13.0	100	93
1		1.0	5.0	12.5	86	63

	fuzz_ratio	fuzz_partial_ratio	longest_substr_ratio
0	93		100
1	66		75

[2 rows x 21 columns]

3.5.1 Analysis of extracted features

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

```
In [23]: df_duplicate = df[df['is_duplicate'] == 1]
dfp_nonduplicate = df[df['is_duplicate'] == 0]

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

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

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

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

In [24]: # reading the text files and removing the Stop Words:
d = path.dirname('.')
```



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

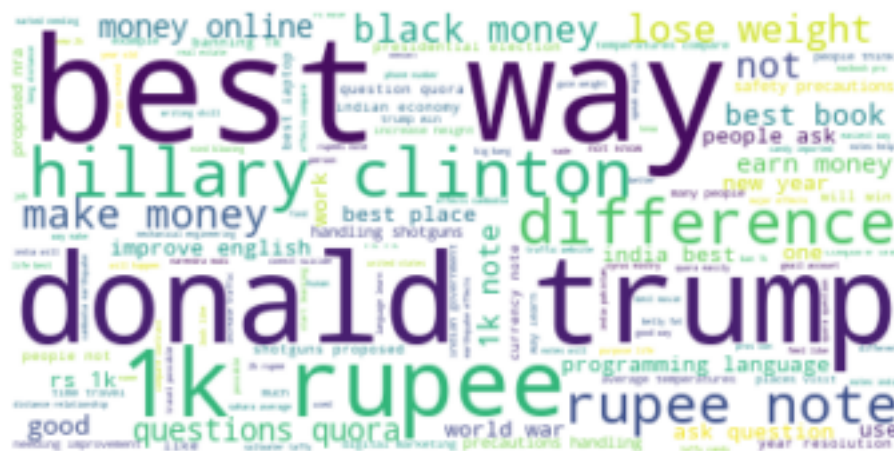
stopwords.remove("no")
#stopwords.remove("good")
#stopwords.remove("love")
stopwords.remove("like")
#stopwords.remove("best")
#stopwords.remove("!")
print ("Total number of words in duplicate pair questions :",len(textp_w))
print ("Total number of words in non duplicate pair questions :",len(textn_w))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

Total number of words in duplicate pair questions : 16110763
Total number of words in non duplicate pair questions : 33201102

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

```
In [25]: 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 [26]: 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 [27]: n = df.shape[0]
sns.pairplot(df[['ctc_min', 'cwc_min', 'csc_min', 'token_sort_ratio', 'is_duplicate']])
plt.show()
```



In [28]: # Distribution of the token_sort_ratio

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

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

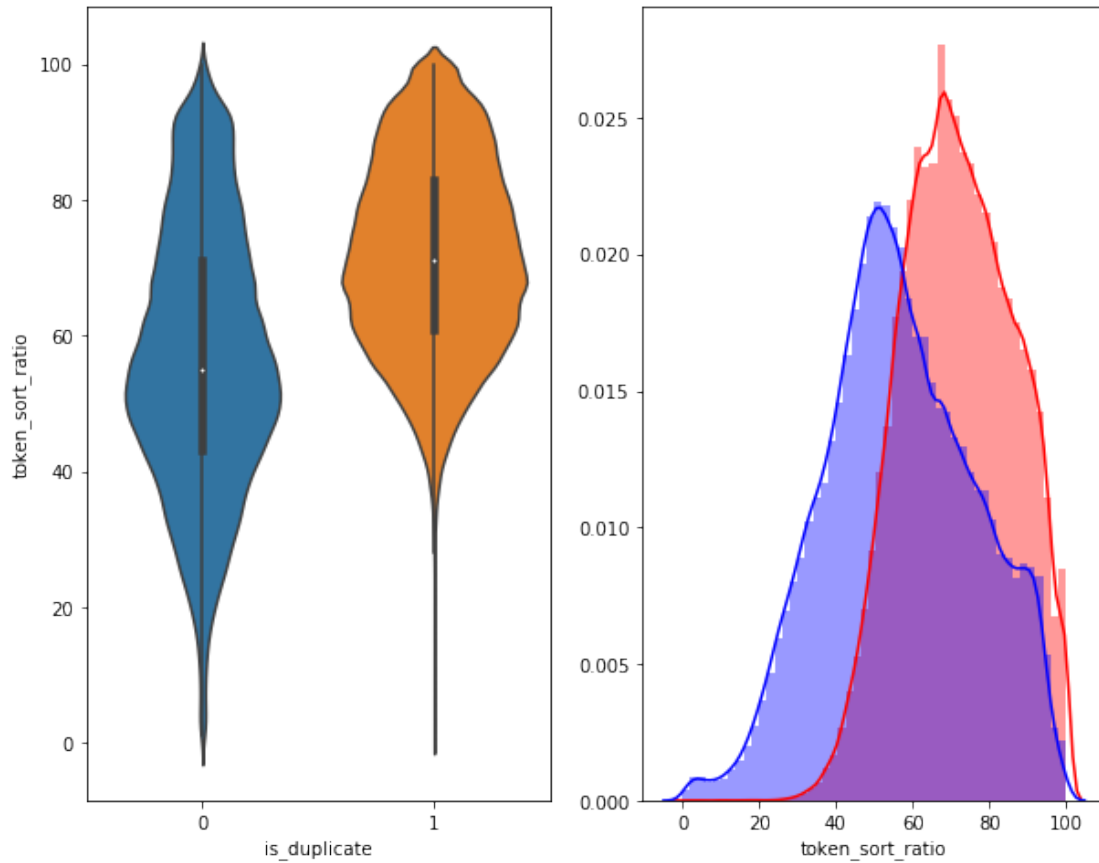
```
sns.violinplot(x = 'is_duplicate', y = 'token_sort_ratio', data = df[0:] , )
```

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

```
sns.distplot(df[df['is_duplicate'] == 1.0]['token_sort_ratio'][0:] , label = "1", col
```

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

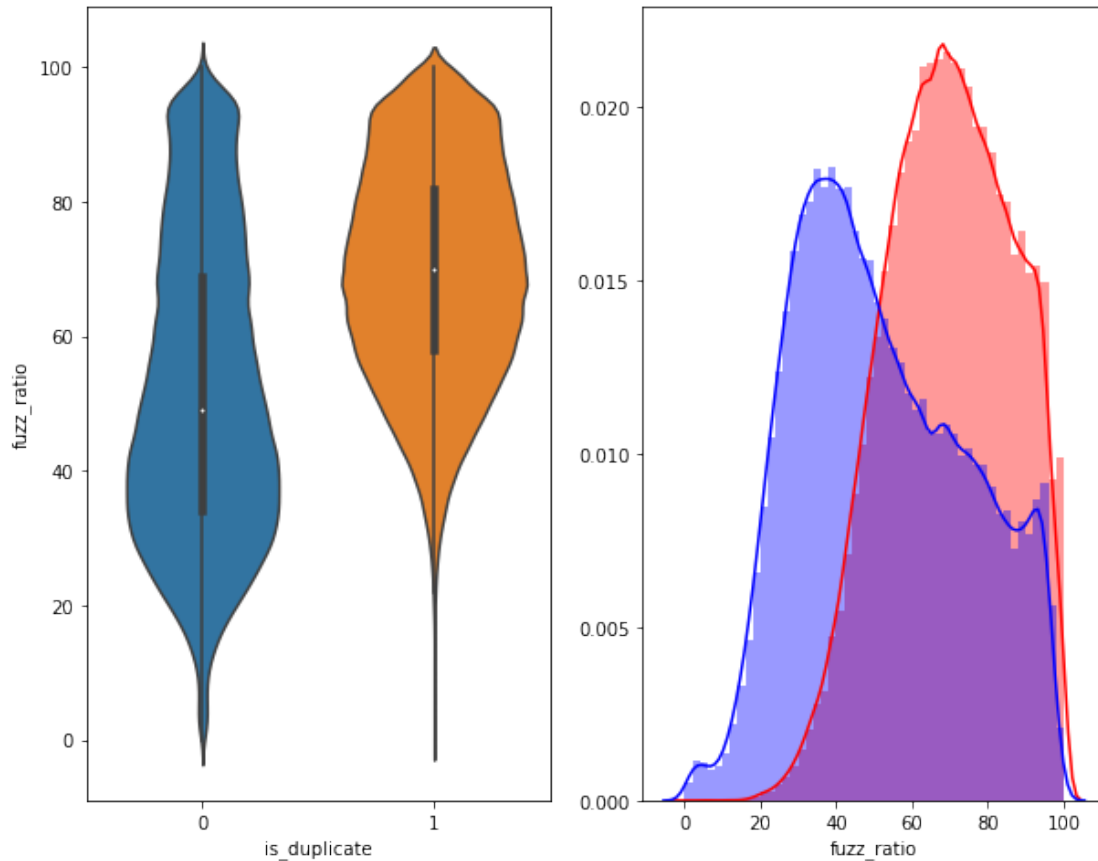
```
plt.show()
```



```
In [29]: plt.figure(figsize=(10, 8))
```

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

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



3.5.2 Visualization

In [30]: *# Using TSNE for Dimentionality reduction for 15 Features(Generated after cleaning the*

```
from sklearn.preprocessing import MinMaxScaler
```

```
dfp_subsampled = df[0:5000]
```

```
X = MinMaxScaler().fit_transform(dfp_subsampled[['cwc_min', 'cwc_max', 'csc_min', 'csc_max']])
```

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

```
In [31]: tsne2d = TSNE(
    n_components=2,
    init='random', # pca
    random_state=101,
    method='barnes_hut',
    n_iter=1000,
    verbose=2,
    angle=0.5
).fit_transform(X)
```

[t-SNE] Computing 91 nearest neighbors...

[t-SNE] Indexed 5000 samples in 0.013s...

```

[t-SNE] Computed neighbors for 5000 samples in 0.304s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilities in 0.310s
[t-SNE] Iteration 50: error = 81.2897949, gradient norm = 0.0455700 (50 iterations in 3.856s)
[t-SNE] Iteration 100: error = 70.6164398, gradient norm = 0.0095177 (50 iterations in 3.056s)
[t-SNE] Iteration 150: error = 68.9172134, gradient norm = 0.0056736 (50 iterations in 2.977s)
[t-SNE] Iteration 200: error = 68.1004639, gradient norm = 0.0049672 (50 iterations in 3.045s)
[t-SNE] Iteration 250: error = 67.5914536, gradient norm = 0.0039700 (50 iterations in 3.111s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.591454
[t-SNE] Iteration 300: error = 1.7926962, gradient norm = 0.0011878 (50 iterations in 3.302s)
[t-SNE] Iteration 350: error = 1.3936826, gradient norm = 0.0004807 (50 iterations in 3.242s)
[t-SNE] Iteration 400: error = 1.2281071, gradient norm = 0.0002778 (50 iterations in 3.251s)
[t-SNE] Iteration 450: error = 1.1385784, gradient norm = 0.0001864 (50 iterations in 3.258s)
[t-SNE] Iteration 500: error = 1.0835493, gradient norm = 0.0001437 (50 iterations in 3.262s)
[t-SNE] Iteration 550: error = 1.0471643, gradient norm = 0.0001152 (50 iterations in 3.271s)
[t-SNE] Iteration 600: error = 1.0231258, gradient norm = 0.0001007 (50 iterations in 3.286s)
[t-SNE] Iteration 650: error = 1.0069925, gradient norm = 0.0000892 (50 iterations in 3.295s)
[t-SNE] Iteration 700: error = 0.9953420, gradient norm = 0.0000804 (50 iterations in 3.318s)
[t-SNE] Iteration 750: error = 0.9866475, gradient norm = 0.0000728 (50 iterations in 3.328s)
[t-SNE] Iteration 800: error = 0.9796536, gradient norm = 0.0000658 (50 iterations in 3.319s)
[t-SNE] Iteration 850: error = 0.9737327, gradient norm = 0.0000618 (50 iterations in 3.311s)
[t-SNE] Iteration 900: error = 0.9688665, gradient norm = 0.0000594 (50 iterations in 3.318s)
[t-SNE] Iteration 950: error = 0.9644679, gradient norm = 0.0000589 (50 iterations in 3.323s)
[t-SNE] Iteration 1000: error = 0.9610358, gradient norm = 0.0000559 (50 iterations in 3.321s)
[t-SNE] Error after 1000 iterations: 0.961036

```

```

In [32]: df = pd.DataFrame({'x':tsne2d[:,0], 'y':tsne2d[:,1] , 'label':y})

```

```

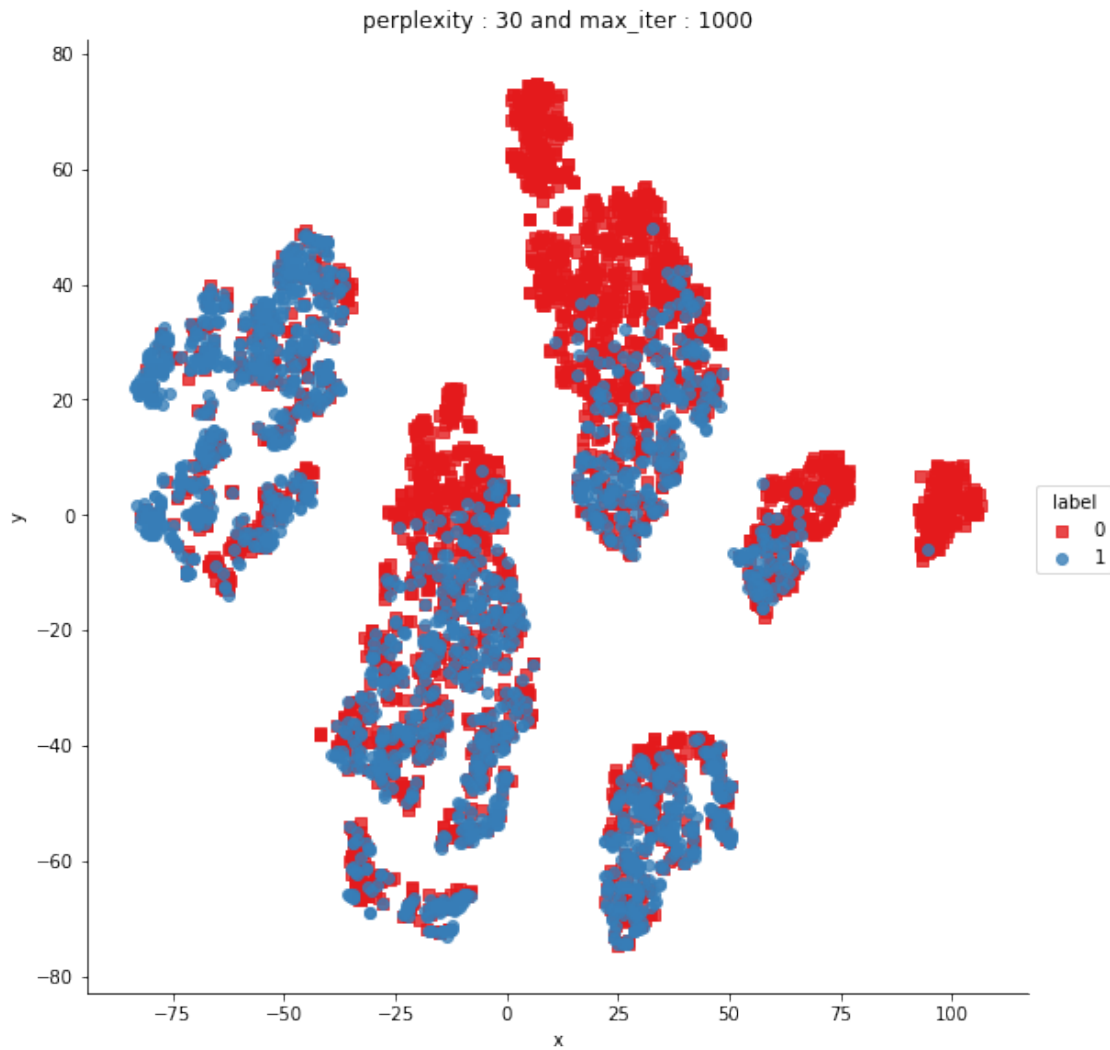
# draw the plot in appropriate place in the grid

```

```

sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,palette="Set1",m
plt.title("perplexity : {} and max_iter : {}".format(30, 1000))
plt.show()

```



```
In [33]: 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  
)
```

```
[t-SNE] Computing 91 nearest neighbors...
```

```
[t-SNE] Indexed 5000 samples in 0.005s...
```

```
[t-SNE] Computed neighbors for 5000 samples in 0.298s...
```

```
[t-SNE] Computed conditional probabilities for sample 1000 / 5000
```



```

[t-SNE] Computed conditional probabilities for sample 2000 / 5000
[t-SNE] Computed conditional probabilities for sample 3000 / 5000
[t-SNE] Computed conditional probabilities for sample 4000 / 5000
[t-SNE] Computed conditional probabilities for sample 5000 / 5000
[t-SNE] Mean sigma: 0.130446
[t-SNE] Computed conditional probabilities in 0.307s
[t-SNE] Iteration 50: error = 80.5298615, gradient norm = 0.0306586 (50 iterations in 13.377s)
[t-SNE] Iteration 100: error = 69.3777008, gradient norm = 0.0037944 (50 iterations in 7.474s)
[t-SNE] Iteration 150: error = 67.9726028, gradient norm = 0.0017517 (50 iterations in 7.136s)
[t-SNE] Iteration 200: error = 67.4098892, gradient norm = 0.0013384 (50 iterations in 7.196s)
[t-SNE] Iteration 250: error = 67.0977859, gradient norm = 0.0009594 (50 iterations in 7.179s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 67.097786
[t-SNE] Iteration 300: error = 1.5276405, gradient norm = 0.0007237 (50 iterations in 8.832s)
[t-SNE] Iteration 350: error = 1.1820400, gradient norm = 0.0002119 (50 iterations in 10.675s)
[t-SNE] Iteration 400: error = 1.0407882, gradient norm = 0.0001023 (50 iterations in 10.452s)
[t-SNE] Iteration 450: error = 0.9688321, gradient norm = 0.0000652 (50 iterations in 10.217s)
[t-SNE] Iteration 500: error = 0.9303923, gradient norm = 0.0000554 (50 iterations in 10.104s)
[t-SNE] Iteration 550: error = 0.9110239, gradient norm = 0.0000524 (50 iterations in 10.026s)
[t-SNE] Iteration 600: error = 0.9016075, gradient norm = 0.0000421 (50 iterations in 10.119s)
[t-SNE] Iteration 650: error = 0.8924681, gradient norm = 0.0000360 (50 iterations in 10.179s)
[t-SNE] Iteration 700: error = 0.8837291, gradient norm = 0.0000353 (50 iterations in 10.207s)
[t-SNE] Iteration 750: error = 0.8771634, gradient norm = 0.0000316 (50 iterations in 10.173s)
[t-SNE] Iteration 800: error = 0.8718039, gradient norm = 0.0000295 (50 iterations in 10.140s)
[t-SNE] Iteration 850: error = 0.8669323, gradient norm = 0.0000276 (50 iterations in 10.153s)
[t-SNE] Iteration 900: error = 0.8628623, gradient norm = 0.0000262 (50 iterations in 10.178s)
[t-SNE] Iteration 950: error = 0.8591092, gradient norm = 0.0000241 (50 iterations in 10.159s)
[t-SNE] Iteration 1000: error = 0.8553245, gradient norm = 0.0000220 (50 iterations in 10.166s)
[t-SNE] Error after 1000 iterations: 0.855325

```

```

In [34]: trace1 = go.Scatter3d(
    x=tsne3d[:,0],
    y=tsne3d[:,1],
    z=tsne3d[:,2],
    mode='markers',
    marker=dict(
        sizemode='diameter',
        color = y,
        colorscale = 'Portland',
        colorbar = dict(title = 'duplicate'),
        line=dict(color='rgb(255, 255, 255)'),
        opacity=0.75
    )
)

data=[trace1]
layout=dict(height=800, width=800, title='3d embedding with engineered features')
fig=dict(data=data, layout=layout)

```



```
py.ipplot(fig, filename='3DBubble')
```

3.6 Featurizing text data with tfidf weighted word-vectors

```
In [3]: import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

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

In [3]: # avoid decoding problems
df = pd.read_csv("train.csv")

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----- python 2 -----
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x), "utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x), "utf-8"))
# ----- python 3 -----
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))

In [4]: df.head()

Out[4]:
```

	id	qid1	qid2	question1	\
0	0	1	2	What is the step by step guide to invest in sh...	
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	
2	2	5	6	How can I increase the speed of my internet co...	
3	3	7	8	Why am I mentally very lonely? How can I solve...	
4	4	9	10	Which one dissolve in water quickly sugar, salt...	

	question2	is_duplicate
0	What is the step by step guide to invest in sh...	0

1	What would happen if the Indian government sto...	0
2	How can Internet speed be increased by hacking...	0
3	Find the remainder when 23^{24} is divided by 1000...	0
4	Which fish would survive in salt water?	0

```
In [5]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.feature_extraction.text import CountVectorizer
        # merge texts
        questions = list(df['question1']) + list(df['question2'])
```

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

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

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

```
In [4]: import en_core_web_sm
```

```
In [7]: # en_vectors_web_lg, which includes over 1 million unique vectors.
        nlp = en_core_web_sm.load()
```

```
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(list(df['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), 384])
    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 [51:03<00:00, 131.95it/s]
```

```
In [8]: vecs2 = []
        for qu2 in tqdm(list(df['question2'])):
            doc2 = nlp(qu2)
            mean_vec2 = np.zeros([len(doc2), 384])
            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 [51:38<00:00, 130.49it/s]

```
In [9]: #prepro_features_train.csv (Simple Preprocessing Feartures)
        #nlp_features_train.csv (NLP Features)
        if os.path.isfile('nlp_features_train.csv'):
            dfnlp = pd.read_csv("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("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 [10]: df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
          df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
          df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
          df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)
          df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
```

```
In [11]: # dataframe of nlp features
          df1.head()
```

```
Out[11]:
```

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	\
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	

	ctc_max	last_word_eq	first_word_eq	abs_len_diff	mean_len	\
0	0.785709	0.0	1.0	2.0	13.0	
1	0.466664	0.0	1.0	5.0	12.5	
2	0.285712	0.0	1.0	4.0	12.0	
3	0.000000	0.0	0.0	2.0	12.0	
4	0.307690	0.0	1.0	6.0	10.0	

	token_set_ratio	token_sort_ratio	fuzz_ratio	fuzz_partial_ratio	\
0	100	93	93	100	
1	86	63	66	75	
2	63	63	43	47	
3	28	24	9	14	
4	67	47	35	56	

	longest_substr_ratio
0	0.982759
1	0.596154
2	0.166667
3	0.039216
4	0.175000

In [12]: # data before preprocessing
df2.head()

Out[12]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	\
0	0	1	1	66	57	14	12	
1	1	4	1	51	88	8	13	
2	2	1	1	73	59	14	10	
3	3	1	1	50	65	11	9	
4	4	3	1	76	39	13	7	

	word_Common	word_Total	word_share	freq_q1+q2	freq_q1-q2
0	10.0	23.0	0.434783	2	0
1	4.0	20.0	0.200000	5	3
2	4.0	24.0	0.166667	2	0
3	0.0	19.0	0.000000	2	0
4	2.0	20.0	0.100000	4	2

In [13]: # Questions 1 tfidf weighted word2vec
df3_q1.head()

Out[13]:

	0	1	2	3	4	5	\
0	121.929942	100.083880	72.497911	115.641811	-48.370869	34.619061	
1	-78.070951	54.843758	82.738470	98.191843	-51.234829	55.013499	
2	-5.355038	73.671822	14.376389	104.130229	1.433505	35.229101	
3	5.778357	-34.712029	48.999641	59.699237	40.661264	-41.658736	
4	51.138244	38.587245	123.639505	53.333045	-47.062794	37.356188	

	6	7	8	9	...	374	\
0	-172.057791	-92.502620	113.223269	50.562425	...	12.397645	
1	-39.140743	-82.692363	45.161478	-9.556312	...	-21.987076	
2	-148.519386	-97.124609	41.972183	50.948724	...	3.027701	
3	-36.808583	24.170647	0.235591	-29.407297	...	13.100011	
4	-298.722757	-106.421101	106.248917	65.880708	...	13.906532	

	375	376	377	378	379	380	381	\
0	40.909527	8.150259	-15.170695	18.007704	6.167002	-30.124162	3.700891	
1	-12.389276	20.667988	2.202712	-17.142450	-5.880969	-10.123960	-4.890663	
2	14.025776	-2.960310	-3.206542	4.355143	2.936156	-20.199560	9.816350	
3	1.405662	-1.891074	-7.882639	18.000562	12.106919	-10.507836	5.243826	
4	43.461717	11.519202	-22.468288	45.431128	8.161224	-35.373911	7.728860	

	382	383
0	-1.757701	-1.818054
1	-13.018387	-5.219299
2	11.894365	-8.798817
3	10.158344	5.886345
4	9.592854	5.447332

[5 rows x 384 columns]

In [14]: # Questions 2 tfidf weighted word2vec
df3_q2.head()

Out [14]:

	0	1	2	3	4	5	\
0	125.983298	95.636470	42.114726	95.450003	-37.386298	39.400067	
1	-106.871918	80.290394	79.066295	59.302086	-42.175396	117.616721	
2	7.072902	15.513379	1.846908	85.937593	-33.808806	94.702355	
3	39.421524	44.136999	-24.010940	85.265890	-0.339027	-9.323140	
4	31.950129	62.854121	1.778174	36.218745	-45.130847	66.674900	

	6	7	8	9	...	374	\
0	-148.116056	-87.851470	110.371952	62.272808	...	16.165598	
1	-144.364294	-127.131529	22.962535	25.397595	...	-4.901131	
2	-122.256852	-114.009528	53.922329	60.131812	...	8.359975	
3	-60.499645	-37.044788	49.407829	-23.350167	...	3.311411	
4	-106.342323	-22.901015	59.835930	62.663936	...	-2.403874	

	375	376	377	378	379	380	\
0	33.030675	7.019995	-14.793956	15.437508	8.199661	-25.070837	
1	-4.565384	41.520752	-0.727562	-16.413774	-7.373776	2.638878	
2	-2.165974	10.936577	-16.531654	14.681221	15.633755	-1.210893	
3	3.788880	13.398604	-6.592597	6.437358	5.993291	2.732391	
4	11.991198	8.088481	-15.090199	8.375162	1.727222	-6.601128	

	381	382	383
0			
1			
2			
3			
4			

0	1.571609	1.603732	0.305657
1	-7.403461	2.703065	0.408052
2	14.183818	11.703130	10.148080
3	-3.727645	5.614124	6.023692
4	11.317407	11.544598	2.478690

[5 rows x 384 columns]

```
In [15]: print("Number of features in nlp dataframe :", df1.shape[1])
         print("Number of features in preprocessed dataframe :", df2.shape[1])
         print("Number of features in question1 w2v dataframe :", df3_q1.shape[1])
         print("Number of features in question2 w2v dataframe :", df3_q2.shape[1])
         print("Number of features in final dataframe :", df1.shape[1]+df2.shape[1]+df3_q1.sh
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v dataframe : 384
Number of features in question2 w2v dataframe : 384
Number of features in final dataframe : 797
```

```
In [16]: # 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='left')
             result = df1.merge(df2, on='id',how='left')
             result.to_csv('final_features.csv')
```

```
In [4]: 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.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier

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

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

```

4. Machine Learning Models

4.1 Reading data from file and storing into sql table

```

In [20]: #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('final_features.csv', names=['Unnamed: 0', 'id', 'is_duplicate']):
                 df.index += index_start
                 j+=1
                 print('{} rows'.format(j*chunksize))
                 df.to_sql('data', disk_engine, if_exists='append')
                 index_start = df.index[-1] + 1

```

180000 rows

360000 rows

540000 rows

```
In [5]: #http://www.sqlitetutorial.net/sqlite-python/create-tables/
```

```
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None

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

```
In [7]: read_db = 'train.db'
        conn_r = create_connection(read_db)
        checkTableExists(conn_r)
        conn_r.close()
```

Tables in the databse:
data

```
In [6]: # 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 ;", conn_r)
                conn_r.commit()
                conn_r.close()
```



```
In [7]: # remove the first row
data.drop(data.index[0], inplace=True)
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id', 'index', 'is_duplicate'], axis=1, inplace=True)
```

```
In [8]: data.head()
```

```
Out [8]:
```

	cwc_min	cwc_max	csc_min	\
1	0.999980000399992	0.8333194446759221	0.9999833336111064	
2	0.7999840003199936	0.3999960000399996	0.7499812504687383	
3	0.3999920001599968	0.3333277778703688	0.3999920001599968	
4	0.0	0.0	0.0	
5	0.3999920001599968	0.19999800001999984	0.9999500024998748	

	csc_max	ctc_min	ctc_max	last_word_eq	\
1	0.9999833336111064	0.9166590278414348	0.7857086735094749	0.0	
2	0.5999880002399952	0.6999930000699993	0.4666635555762962	0.0	
3	0.24999687503906198	0.3999960000399996	0.28571224491253633	0.0	
4	0.0	0.0	0.0	0.0	
5	0.6666444451851604	0.5714204082798817	0.3076899408466089	0.0	

	first_word_eq	abs_len_diff	mean_len	...	\
1	1.0	2.0	13.0	...	
2	1.0	5.0	12.5	...	
3	1.0	4.0	12.0	...	
4	0.0	2.0	12.0	...	
5	1.0	6.0	10.0	...	

	374_y	375_y	376_y	\
1	16.165598386898637	33.03067463636398	7.019995227456093	
2	-4.901130557060242	-4.565384194254875	41.5207524523139	
3	8.359974771738052	-2.165974423289299	10.936577022075653	
4	3.31141060590744	3.788880407810211	13.398604452610016	
5	-2.4038737677037716	11.99119820445776	8.088481079787016	

	377_y	378_y	379_y	\
1	-14.793955877423286	15.437508314847946	8.19966059923172	
2	-0.7275624666363001	-16.41377378255129	-7.373775810003281	
3	-16.53165421076119	14.681220807135105	15.63375510275364	
4	-6.592596508562565	6.437358126044273	5.993290975689888	
5	-15.09019909799099	8.375162452459335	1.7272223234176636	

	380_y	381_y	382_y	\
1	-25.070836670696735	1.5716093145310879	1.6037320122122765	
2	2.638877835124731	-7.403460711240768	2.7030646055936813	
3	-1.2108925580978394	14.183817744255066	11.703129768371582	
4	2.7323912382125854	-3.7276453971862793	5.614123735576868	
5	-6.601127505302429	11.317407250404358	11.544598042964935	

```

          383_y
1  0.3056571036577225
2  0.4080522954463959
3  10.148079725913703
4   6.023691833019257
5  2.4786900877952576

```

```
[5 rows x 794 columns]
```

```
In [9]: data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 404290 entries, 1 to 404290
Columns: 794 entries, cwc_min to 383_y
dtypes: object(794)
memory usage: 2.4+ GB

```

```

In [10]: # after we read from sql table each entry was read it as a string
         # we convert all the features into numeric before we apply any model
         cols = list(data.columns)
         data = pd.DataFrame(np.array(data.values,dtype=np.float64),columns=cols)

```

```
In [11]: y_true = list(map(int, y_true.values))
```

4.3 Random train test split(70:30)

```
In [12]: X_train,X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)
```

```

In [13]: 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 : (283003, 794)
```

```
Number of data points in test data : (121287, 794)
```

```

In [14]: 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 test data", "-"*10)
         test_distr = Counter(y_test)
         test_len = len(y_test)
         print("Class 0: ",int(test_distr[0])/test_len, "Class 1: ",int(test_distr[1])/test_len)

```

```

----- Distribution of output variable in train data -----
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
----- Distribution of output variable in test data -----
Class 0:  0.3691986775169639 Class 1:  0.3691986775169639

```

```

In [5]: # 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 as 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 corresponds to columns and axis=1 corresponds to rows in
    # C.sum(axis=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 corresponds to columns and axis=1 corresponds to rows in
    # C.sum(axis=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)

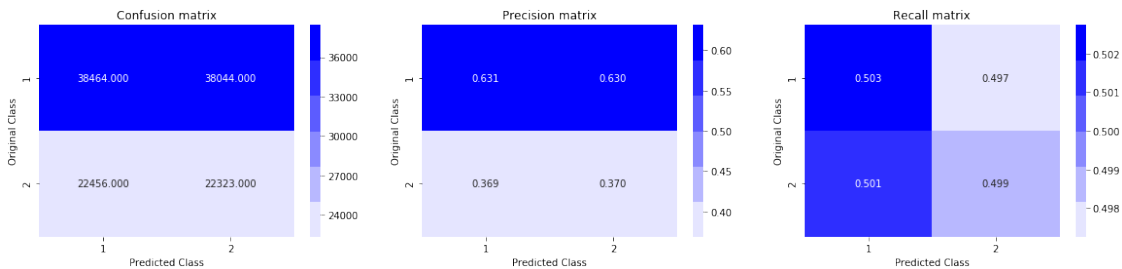
```

In [33]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs))))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-16))

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

```

Log loss on Test Data using Random Model 0.8876992330072402



4.4 Logistic Regression with hyperparameter tuning

```

In [16]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ]) # Fit linear model with Stochastic Gradient Descent

```

```

# predict(X)          Predict class labels for samples in X.

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

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

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

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

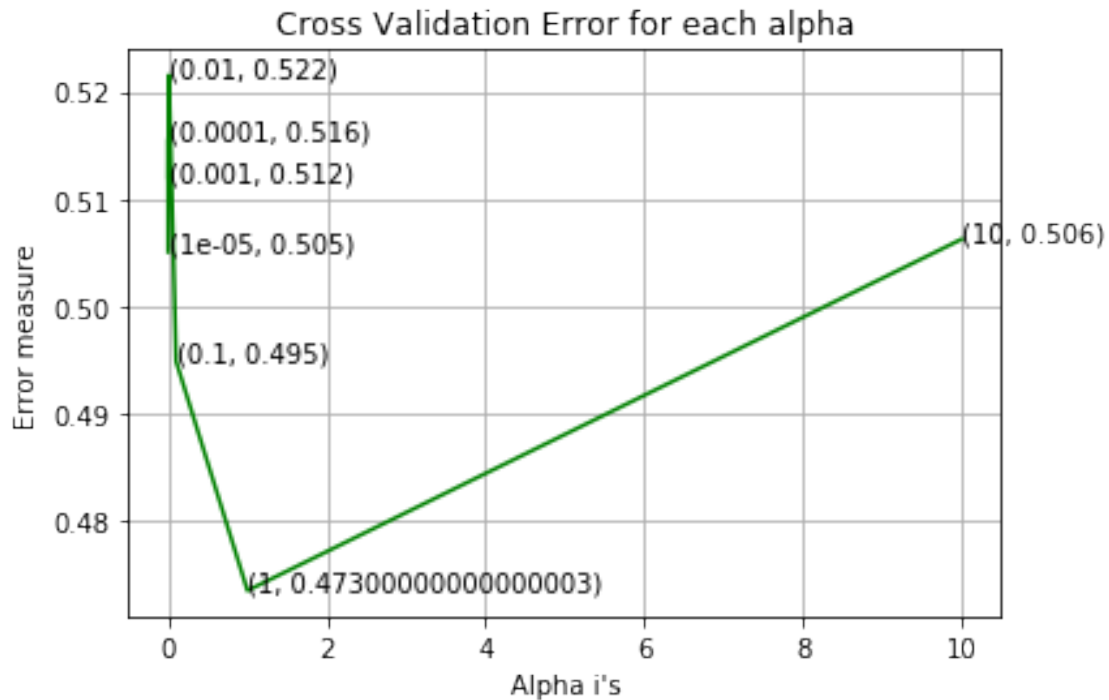
predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

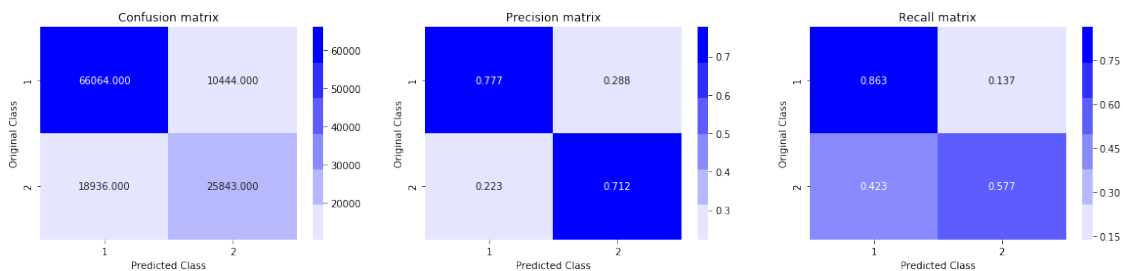
```

For values of alpha = 1e-05 The log loss is: 0.5050464987596055
For values of alpha = 0.0001 The log loss is: 0.5156064161083074
For values of alpha = 0.001 The log loss is: 0.5118124206289113
For values of alpha = 0.01 The log loss is: 0.5215407246451201
For values of alpha = 0.1 The log loss is: 0.49485550469453093
For values of alpha = 1 The log loss is: 0.47346295882439915
For values of alpha = 10 The log loss is: 0.5062282696956512

```



For values of best alpha = 1 The train log loss is: 0.4712558809818194
 For values of best alpha = 1 The test log loss is: 0.47346295882439915
 Total number of data points : 121287



SGD is sensitive to feature scaling, so did scaling and tried.

```
In [17]: from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X_train_sc = scale.fit_transform(X_train)
X_test_sc = scale.transform(X_test)
```

```
In [39]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
```

```

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

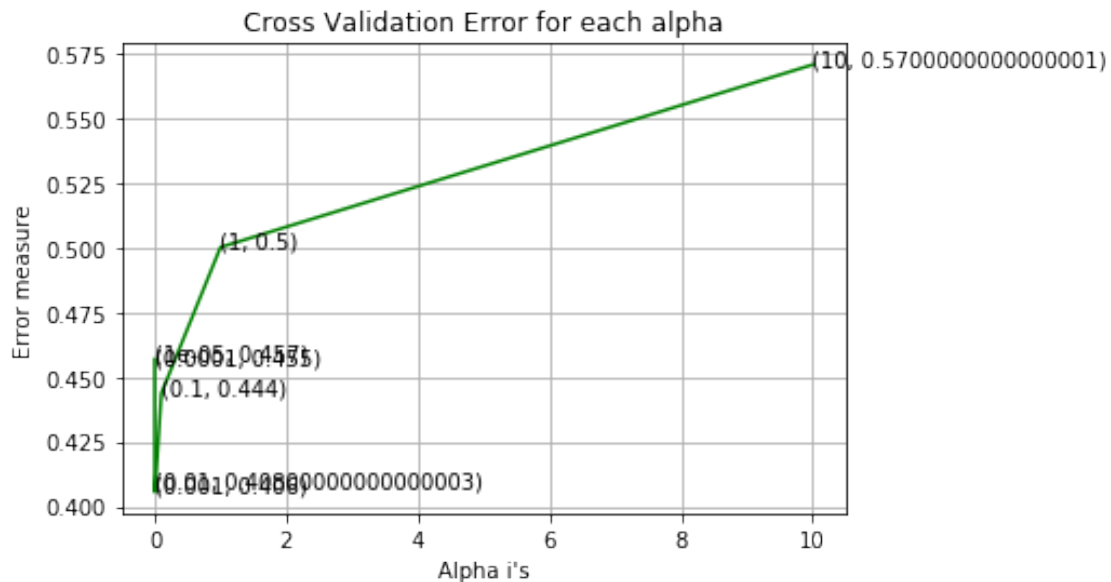
fig, ax = plt.subplots()
ax.plot(alpha, log_error_array, c='g')
for i, txt in enumerate(np.round(log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

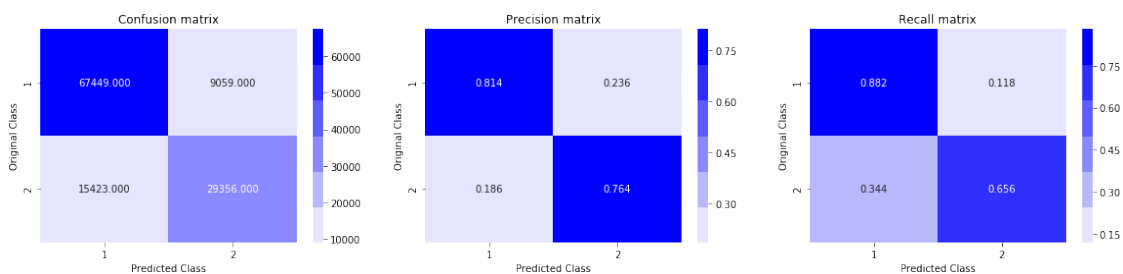
predict_y = sig_clf.predict_proba(X_train_sc)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test_sc)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, 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: 0.45683456744359
 For values of alpha = 0.0001 The log loss is: 0.45492081462801304
 For values of alpha = 0.001 The log loss is: 0.40559361222633294
 For values of alpha = 0.01 The log loss is: 0.40761367674333254
 For values of alpha = 0.1 The log loss is: 0.44362792720222327
 For values of alpha = 1 The log loss is: 0.5000580130072036
 For values of alpha = 10 The log loss is: 0.5704629435817549



For values of best alpha = 0.001 The train log loss is: 0.4031443854177573
 For values of best alpha = 0.001 The test log loss is: 0.40559361222633294
 Total number of data points : 121287



In [51]: `#alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.`

`# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/`


```

# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

np.random.seed(45)
alpha = np.random.uniform(0.0006,0.006,14)
alpha = np.round(alpha,6)
alpha.sort()
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_sc, y_train)
    predict_y = sig_clf.predict_proba(X_test_sc)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, eps=1e-15))

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

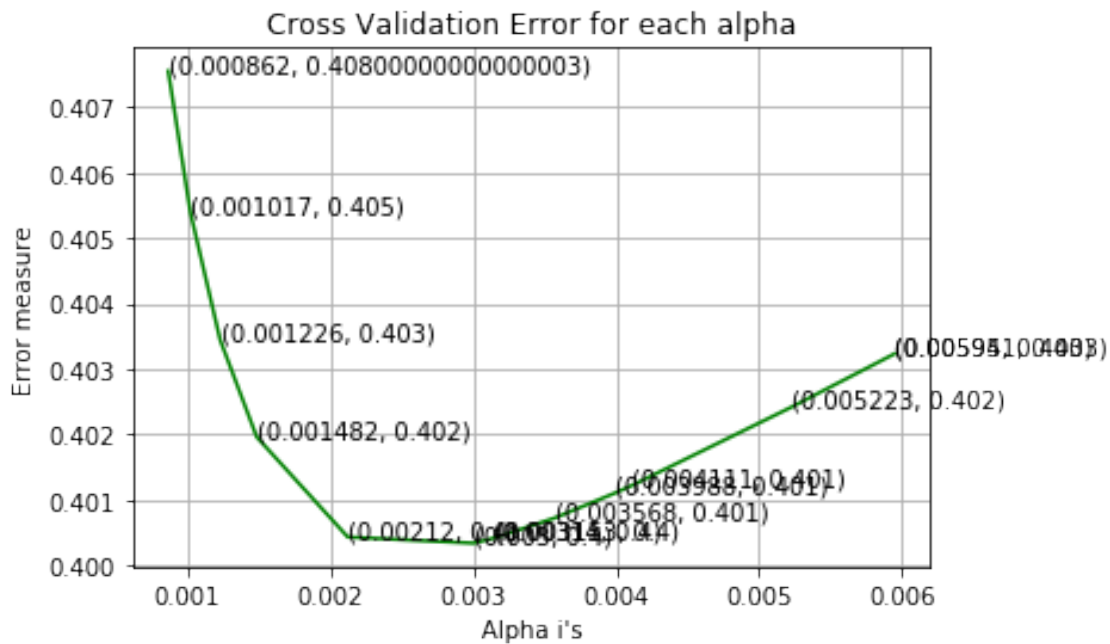
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_sc, y_train)

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

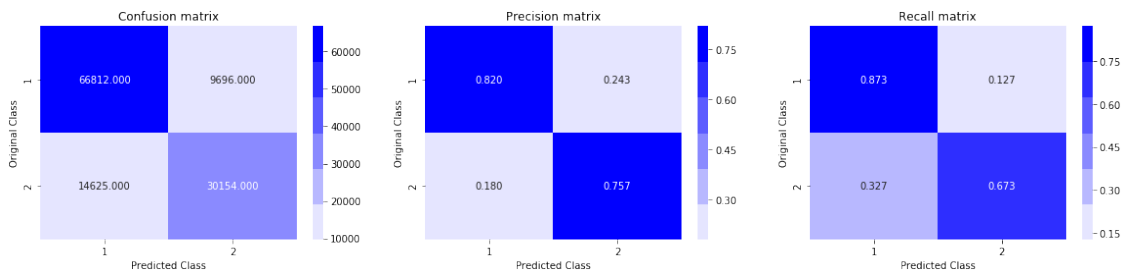
```

```
plot_confusion_matrix(y_test, predicted_y)
```

For values of alpha = 0.000862 The log loss is: 0.4075498397239791
 For values of alpha = 0.001017 The log loss is: 0.40539250011035743
 For values of alpha = 0.001226 The log loss is: 0.4034425566208746
 For values of alpha = 0.001482 The log loss is: 0.4019625978302733
 For values of alpha = 0.00212 The log loss is: 0.40043503086001114
 For values of alpha = 0.003 The log loss is: 0.40034152313699967
 For values of alpha = 0.00314 The log loss is: 0.4004166666193004
 For values of alpha = 0.003153 The log loss is: 0.4004249565995252
 For values of alpha = 0.003568 The log loss is: 0.4007241237416872
 For values of alpha = 0.003988 The log loss is: 0.40110250715655493
 For values of alpha = 0.004111 The log loss is: 0.4012226624599859
 For values of alpha = 0.005223 The log loss is: 0.4024120323437416
 For values of alpha = 0.005941 The log loss is: 0.40322094282826937
 For values of alpha = 0.00595 The log loss is: 0.4032311500519025



For values of best alpha = 0.003 The train log loss is: 0.39784383489862885
 For values of best alpha = 0.003 The test log loss is: 0.40034152313699967
 Total number of data points : 121287



4.5 Linear SVM with hyperparameter tuning

In [56]: `alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.`

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

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

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

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

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

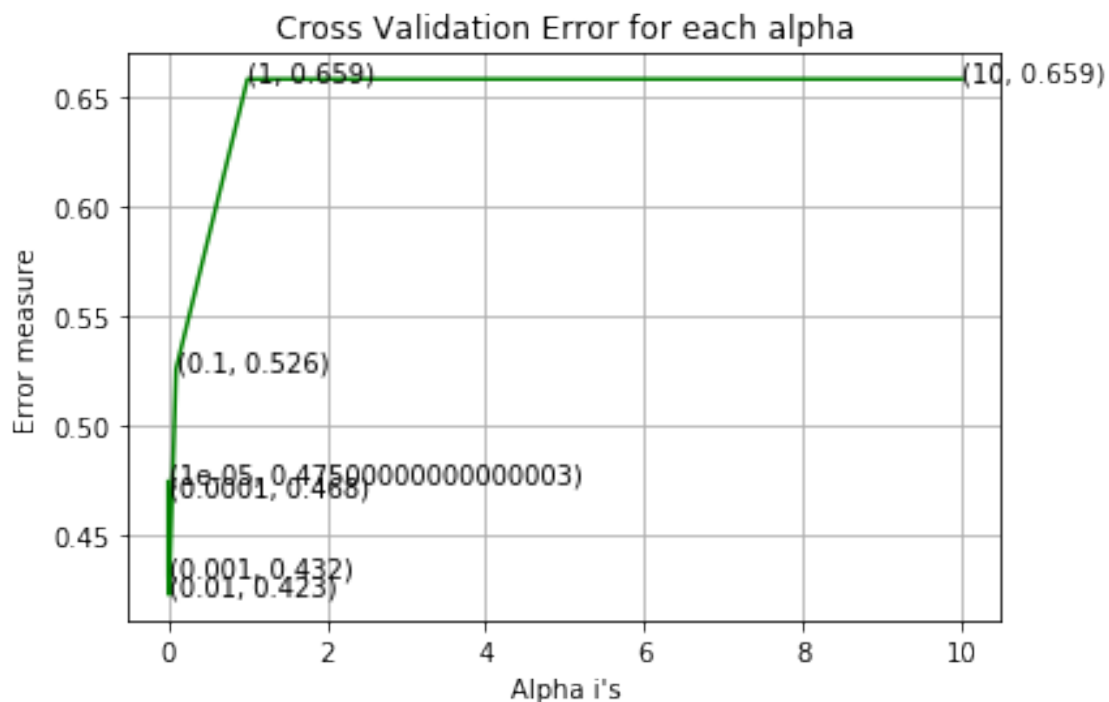
```

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

predict_y = sig_clf.predict_proba(X_train_sc)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(X_test_sc)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
predicted_y = np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

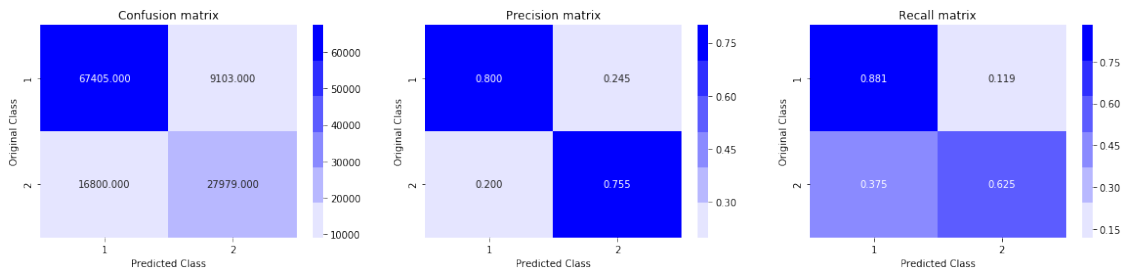
```

For values of alpha = 1e-05 The log loss is: 0.47459282302213923
 For values of alpha = 0.0001 The log loss is: 0.46819163734685043
 For values of alpha = 0.001 The log loss is: 0.43159379565435446
 For values of alpha = 0.01 The log loss is: 0.42292019437911754
 For values of alpha = 0.1 The log loss is: 0.5260749681006414
 For values of alpha = 1 The log loss is: 0.6585278256322723
 For values of alpha = 10 The log loss is: 0.6585278256322611



For values of best alpha = 0.01 The train log loss is: 0.4223724082952659
 For values of best alpha = 0.01 The test log loss is: 0.42292019437911754

Total number of data points : 121287



In [57]: `alpha = [10 ** x for x in range(-5, 2)]` # hyperparam for SGD classifier.

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

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

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

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

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

```

plt.ylabel("Error measure")
plt.show()

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

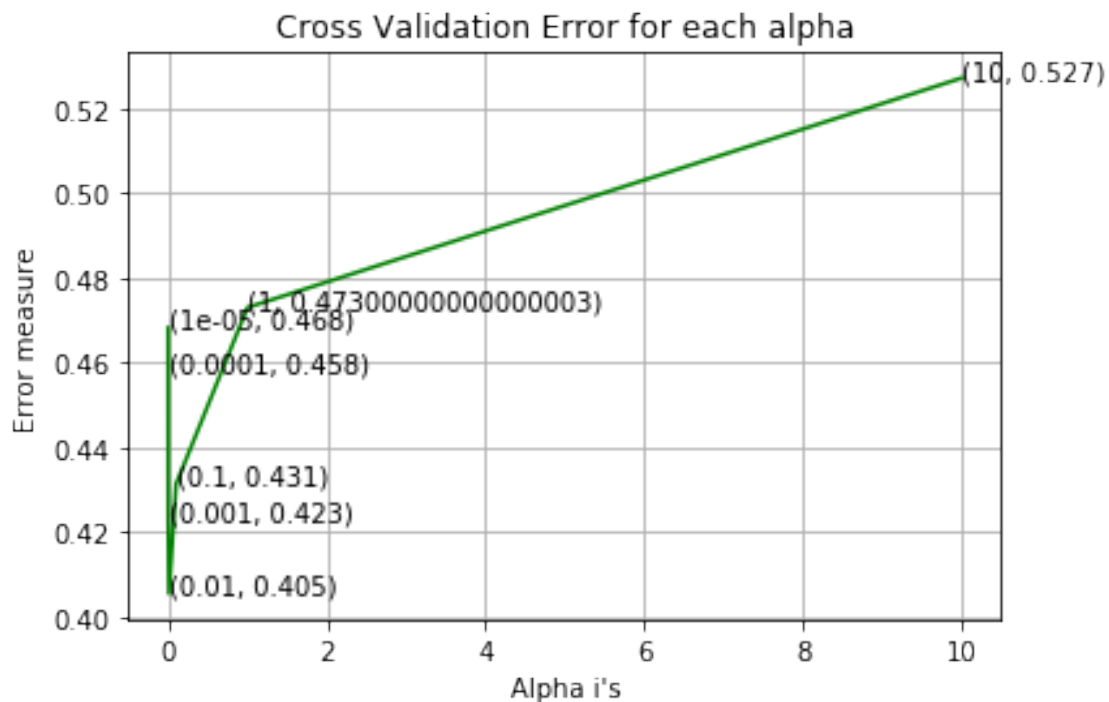
predict_y = sig_clf.predict_proba(X_train_sc)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_
predict_y = sig_clf.predict_proba(X_test_sc)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_
predicted_y = np.argmax(predict_y, axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

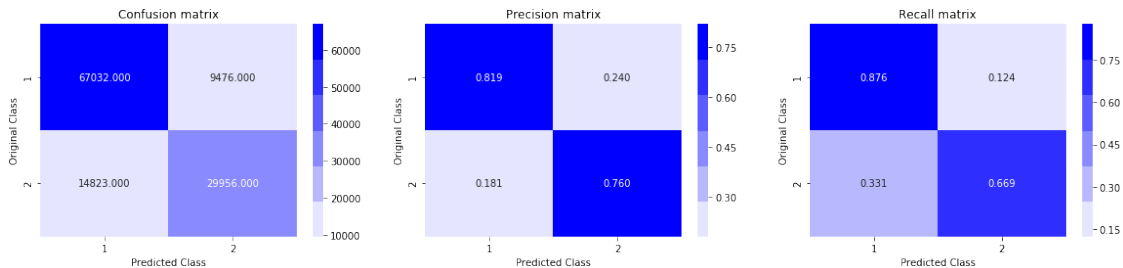
```

For values of alpha = 1e-05 The log loss is: 0.46827517367892096
For values of alpha = 0.0001 The log loss is: 0.45775339387339004
For values of alpha = 0.001 The log loss is: 0.42313495536418577
For values of alpha = 0.01 The log loss is: 0.40541772273229554
For values of alpha = 0.1 The log loss is: 0.4313071929003579
For values of alpha = 1 The log loss is: 0.4730453418766842
For values of alpha = 10 The log loss is: 0.5272789313684569

```



For values of best alpha = 0.01 The train log loss is: 0.4027358236207044
 For values of best alpha = 0.01 The test log loss is: 0.40541772273229554
 Total number of data points : 121287



In [63]: `#alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.`

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

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

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

```
np.random.seed(25)
alpha = np.random.uniform(0.002,0.03,14)
alpha = np.round(alpha,5)
alpha.sort()
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_sc, y_train)
    predict_y = sig_clf.predict_proba(X_test_sc)
    log_error_array.append(log_loss(y_test, predict_y,eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y,eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
```

```

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

best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=0)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train_sc, y_train)

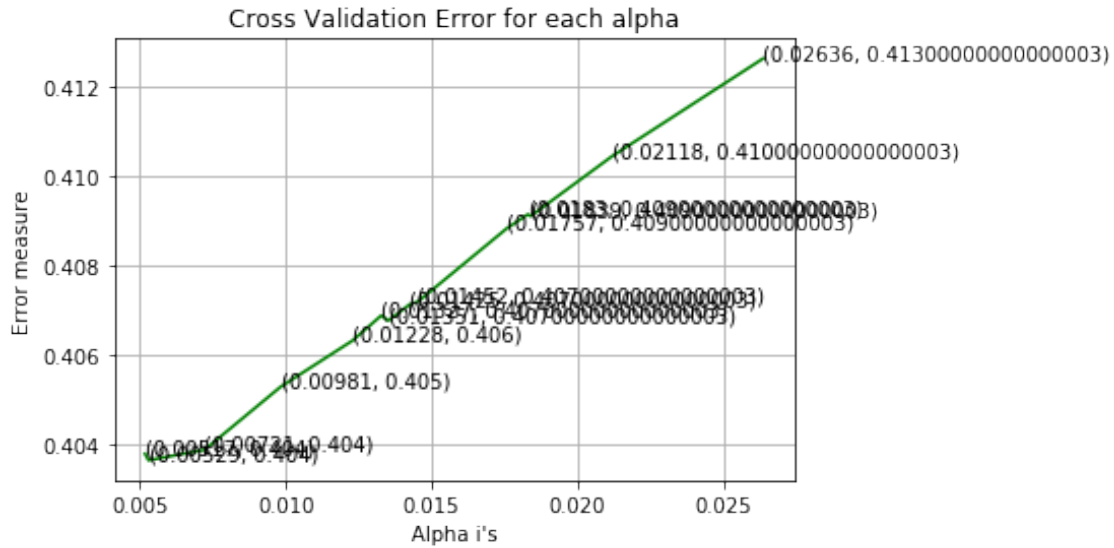
predict_y = sig_clf.predict_proba(X_train_sc)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(X_train_sc, y_train, predict_y))
predict_y = sig_clf.predict_proba(X_test_sc)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(X_test_sc, y_test, predict_y))
predicted_y = np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

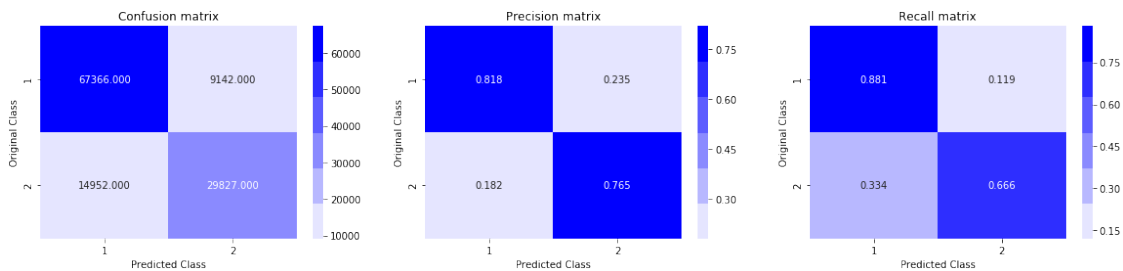
```

For values of alpha = 0.00517 The log loss is: 0.4037706351092166
For values of alpha = 0.00529 The log loss is: 0.4036245330070321
For values of alpha = 0.00721 The log loss is: 0.4038591515869226
For values of alpha = 0.00981 The log loss is: 0.4052693810491646
For values of alpha = 0.01228 The log loss is: 0.40630937344731693
For values of alpha = 0.01327 The log loss is: 0.4068733679114762
For values of alpha = 0.01351 The log loss is: 0.4067584415367667
For values of alpha = 0.01425 The log loss is: 0.40712629206352663
For values of alpha = 0.01452 The log loss is: 0.40718224970845057
For values of alpha = 0.01757 The log loss is: 0.40882766620678973
For values of alpha = 0.0183 The log loss is: 0.4091457931830299
For values of alpha = 0.01839 The log loss is: 0.4091050200862671
For values of alpha = 0.02118 The log loss is: 0.41044869825018293
For values of alpha = 0.02636 The log loss is: 0.4126434814409032

```

For values of best alpha = 0.00529 The train log loss is: 0.4008580635552486
 For values of best alpha = 0.00529 The test log loss is: 0.4036245330070321
 Total number of data points : 121287



4.6 Random Forest

```
In [16]: from sklearn.ensemble import RandomForestClassifier as RFC

In [19]: estimators = [100,150,200,300,400,600,800]
test_scores = []
train_scores = []
for i in estimators:
    clf = RFC(n_estimators=i,n_jobs=-1)
    clf.fit(X_train,y_train)
    predict_y = clf.predict_proba(X_train)
    log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
    train_scores.append(log_loss_train)
```

```

predict_y = clf.predict_proba(X_test)
log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
test_scores.append(log_loss_test)
print('estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(estimators,train_scores,label='Train Log Loss')
plt.plot(estimators,test_scores,label='Test Log Loss')
plt.xlabel('estimators')
plt.ylabel('Log Loss')

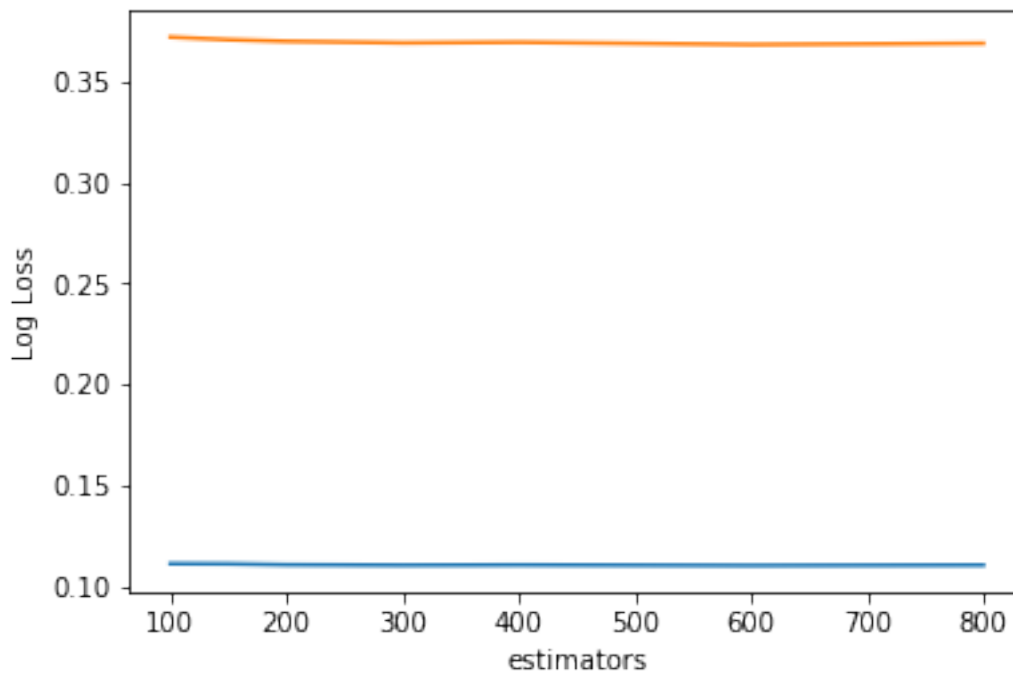
```

```

estimators = 100 Train Log Loss 0.11109459647005955 Test Log Loss 0.37192988174912683
estimators = 150 Train Log Loss 0.11100847787606712 Test Log Loss 0.37065163655861083
estimators = 200 Train Log Loss 0.11056444275163889 Test Log Loss 0.3698413672093776
estimators = 300 Train Log Loss 0.11035449276121001 Test Log Loss 0.3690804893102336
estimators = 400 Train Log Loss 0.11047904985310375 Test Log Loss 0.3693838339921741
estimators = 600 Train Log Loss 0.11021203125673375 Test Log Loss 0.36823350864534526
estimators = 800 Train Log Loss 0.11039528102117314 Test Log Loss 0.3688582117860808

```

Out[19]: Text(0,0.5,'Log Loss')



```

In [19]: Depth = [5,10,12,15,20,25,50]
test_scores = []
train_scores = []
for i in Depth:
    clf = RFC(n_estimators=100,max_depth=i,n_jobs=-1)

```

```

clf.fit(X_train,y_train)
predict_y = clf.predict_proba(X_train)
log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
train_scores.append(log_loss_train)
predict_y = clf.predict_proba(X_test)
log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
test_scores.append(log_loss_test)
print('Depth = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(Depth,train_scores,label='Train Log Loss')
plt.plot(Depth,test_scores,label='Test Log Loss')
plt.xlabel('Depth')
plt.ylabel('Log Loss')

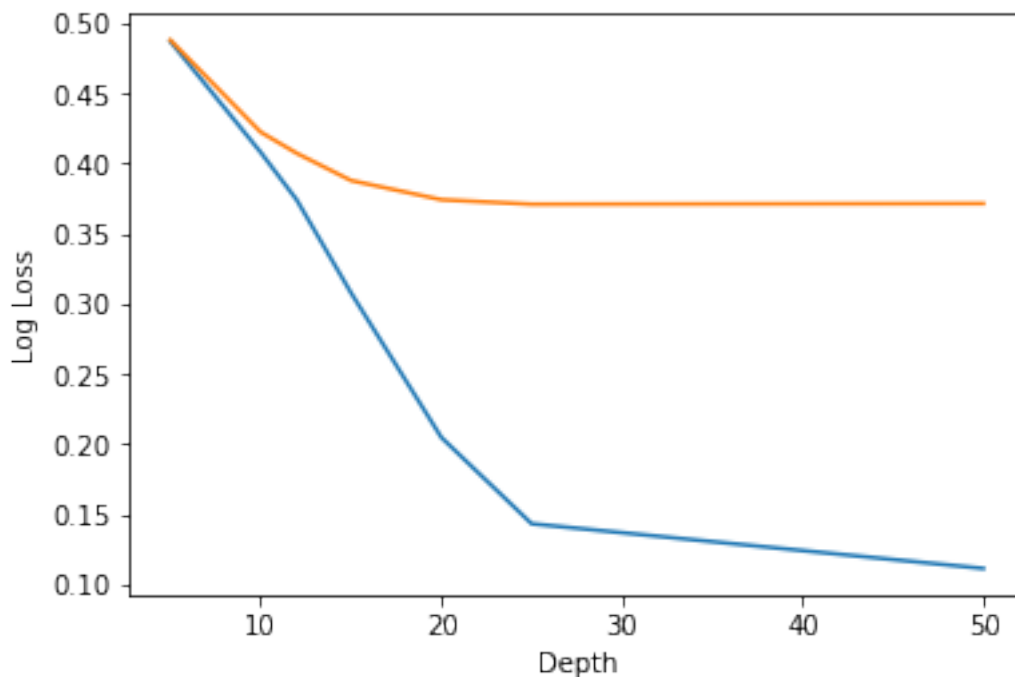
```

```

Depth = 5 Train Log Loss 0.48722729508327145 Test Log Loss 0.48813291311027934
Depth = 10 Train Log Loss 0.4081238487571798 Test Log Loss 0.4225459782185362
Depth = 12 Train Log Loss 0.37408662763549844 Test Log Loss 0.4074103602751899
Depth = 15 Train Log Loss 0.3079064316463802 Test Log Loss 0.38802805760307507
Depth = 20 Train Log Loss 0.20481718757448902 Test Log Loss 0.3743196152422552
Depth = 25 Train Log Loss 0.14326963278706298 Test Log Loss 0.37110604544161335
Depth = 50 Train Log Loss 0.11139277276272488 Test Log Loss 0.3715810469258812

```

Out[19]: Text(0,0.5, 'Log Loss')



```

In [20]: estimators = [100,150,200,300,400,600,800]
         test_scores = []

```

```

train_scores = []
for i in estimators:
    clf = RFC(n_estimators=i,max_depth=11,n_jobs=-1)
    clf.fit(X_train,y_train)
    predict_y = clf.predict_proba(X_train)
    log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
    train_scores.append(log_loss_train)
    predict_y = clf.predict_proba(X_test)
    log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
    test_scores.append(log_loss_test)
    print('estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(estimators,train_scores,label='Train Log Loss')
plt.plot(estimators,test_scores,label='Test Log Loss')
plt.xlabel('estimators')
plt.ylabel('Log Loss')

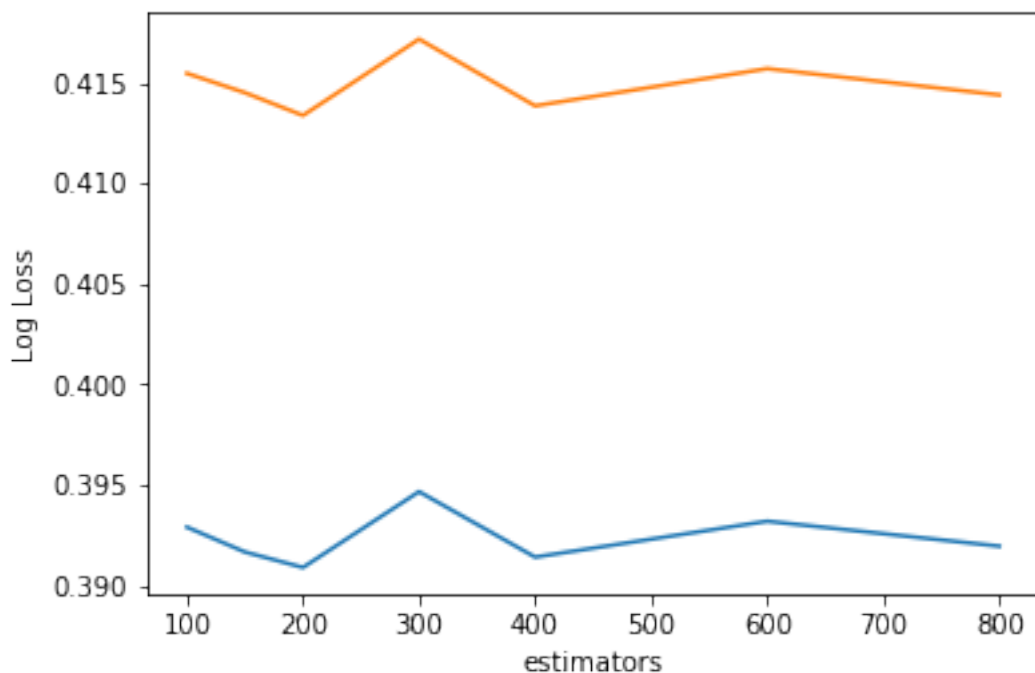
```

```

estimators = 100 Train Log Loss 0.39289057215728057 Test Log Loss 0.4154641445550544
estimators = 150 Train Log Loss 0.3916522623500148 Test Log Loss 0.4145023288788135
estimators = 200 Train Log Loss 0.39088408416850556 Test Log Loss 0.413370659547225
estimators = 300 Train Log Loss 0.39465656231621055 Test Log Loss 0.41716846990811174
estimators = 400 Train Log Loss 0.3913998286278825 Test Log Loss 0.4138491796713054
estimators = 600 Train Log Loss 0.39318544549179174 Test Log Loss 0.4156994049185631
estimators = 800 Train Log Loss 0.3919437224404921 Test Log Loss 0.4143914497614524

```

Out[20]: Text(0,0.5,'Log Loss')



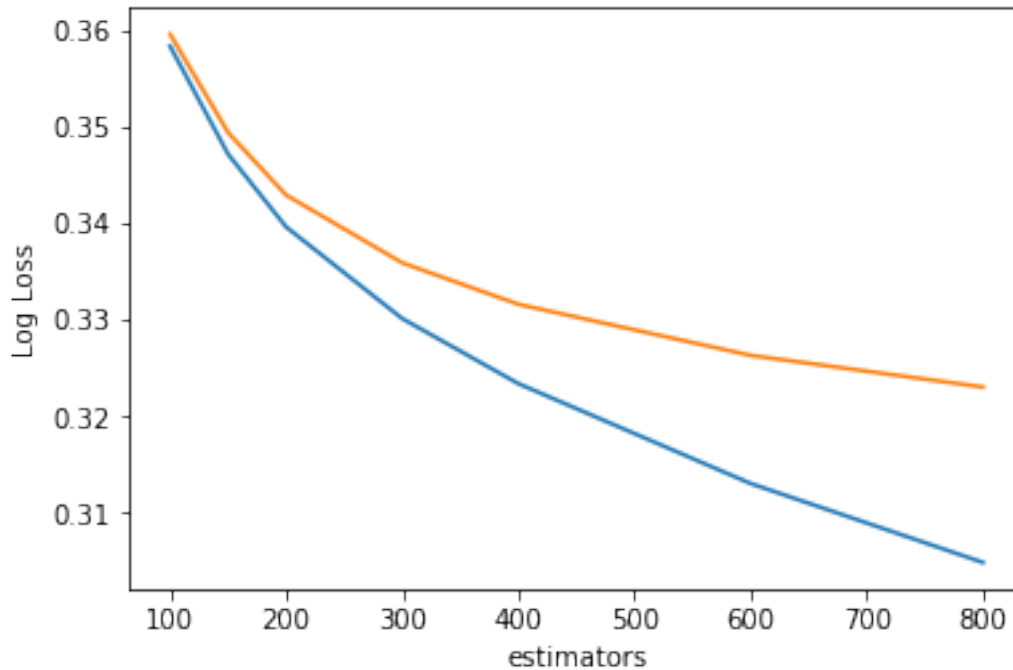
4.7 XGBoost

```
In [6]: import xgboost as xgb
```

```
In [21]: estimators = [100,150,200,300,400,600,800]
         test_scores = []
         train_scores = []
         for i in depths:
             clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.1,n_estimators=i,n_jobs=-1)
             clf.fit(X_train,y_train)
             predict_y = clf.predict_proba(X_train)
             log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
             train_scores.append(log_loss_train)
             predict_y = clf.predict_proba(X_test)
             log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
             test_scores.append(log_loss_test)
             print('estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
         plt.plot(estimators,train_scores,label='Train Log Loss')
         plt.plot(estimators,test_scores,label='Test Log Loss')
         plt.xlabel('estimators')
         plt.ylabel('Log Loss')
```

```
estimators = 100 Train Log Loss 0.3583641938531458 Test Log Loss 0.35961819366299197
estimators = 150 Train Log Loss 0.34716821707657336 Test Log Loss 0.3493991268629394
estimators = 200 Train Log Loss 0.339620979084566 Test Log Loss 0.34292007832953664
estimators = 300 Train Log Loss 0.3301188189311435 Test Log Loss 0.3358950724784673
estimators = 400 Train Log Loss 0.32337384734119523 Test Log Loss 0.33159850014983644
estimators = 600 Train Log Loss 0.3130198865512778 Test Log Loss 0.32630453308630486
estimators = 800 Train Log Loss 0.30483396799846996 Test Log Loss 0.3230216993976186
```

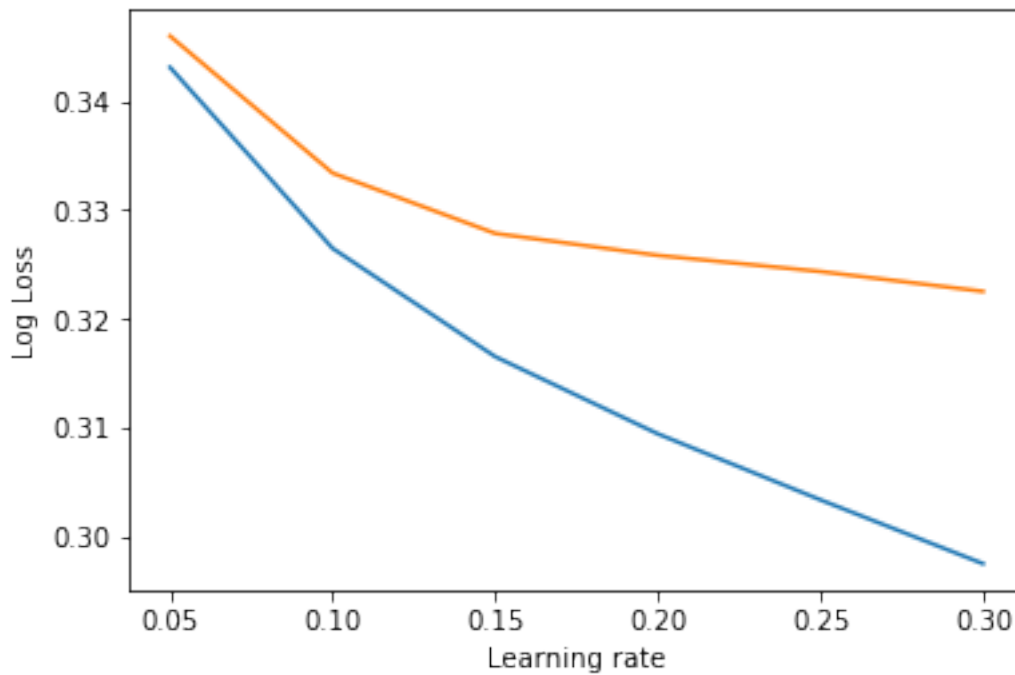
```
Out[21]: Text(0,0.5,'Log Loss')
```



```
In [47]: test_scores = []
train_scores = []
etas = [0.05,0.1,0.15,0.2,0.25,0.3]
for i in etas:
    clf = xgb.XGBClassifier(max_depth=3,learning_rate=i,n_estimators=350,n_jobs=-1)
    clf.fit(X_train,y_train)
    predict_y = clf.predict_proba(X_train)
    log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
    train_scores.append(log_loss_train)
    predict_y = clf.predict_proba(X_test)
    log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
    test_scores.append(log_loss_test)
    print('Learning Rate = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_
plt.plot(etas,train_scores,label='Train Log Loss')
plt.plot(etas,test_scores,label='Test Log Loss')
plt.xlabel('Learning rate')
plt.ylabel('Log Loss')
```

```
Learning Rate = 0.05 Train Log Loss 0.3431145884649425 Test Log Loss 0.34599386091965273
Learning Rate = 0.1 Train Log Loss 0.3264772174456944 Test Log Loss 0.3333956069010313
Learning Rate = 0.15 Train Log Loss 0.31652482578279084 Test Log Loss 0.3278496386770277
Learning Rate = 0.2 Train Log Loss 0.3094281555236832 Test Log Loss 0.32583108629355173
Learning Rate = 0.25 Train Log Loss 0.30336320048274407 Test Log Loss 0.3243467612783949
Learning Rate = 0.3 Train Log Loss 0.2974955443733094 Test Log Loss 0.3225233032247901
```

```
Out[47]: Text(0,0.5,'Log Loss')
```

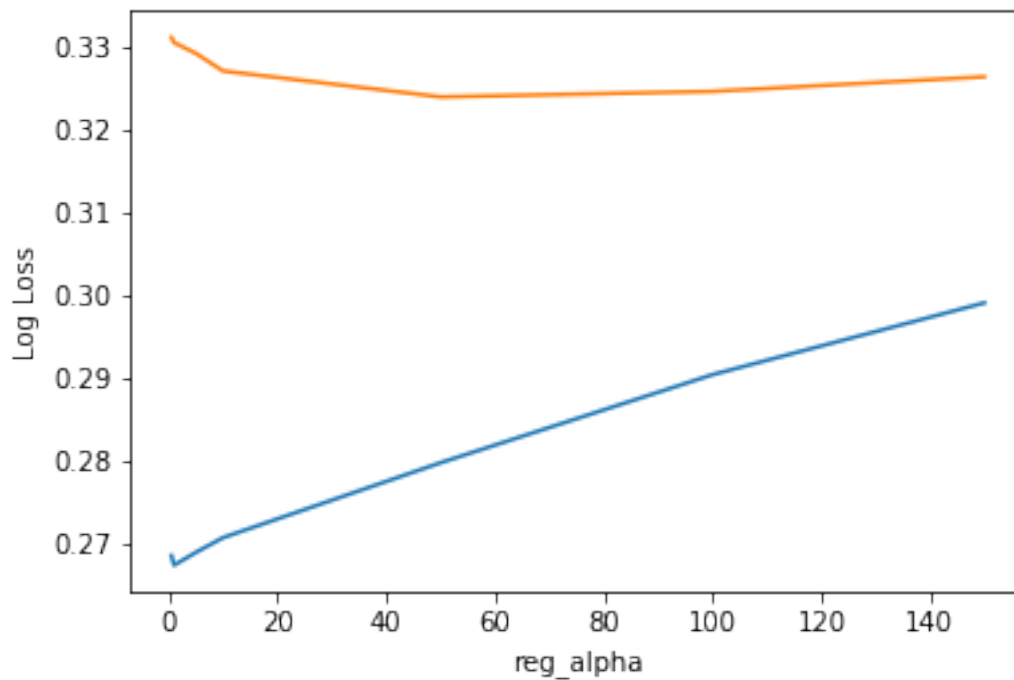


```
In [19]: test_scores = []
train_scores = []
alpha = [0.5,1,5,10,50,100,150]
for i in alpha:
    clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.65,n_estimators=370,reg_alpha=
    clf.fit(X_train,y_train)
    predict_y = clf.predict_proba(X_train)
    log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
    train_scores.append(log_loss_train)
    predict_y = clf.predict_proba(X_test)
    log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
    test_scores.append(log_loss_test)
    print('reg_alpha = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss
plt.plot(alpha,train_scores,label='Train Log Loss')
plt.plot(alpha,test_scores,label='Test Log Loss')
plt.xlabel('reg_alpha')
plt.ylabel('Log Loss')
```

```
reg_alpha = 0.5 Train Log Loss 0.26852223284338356 Test Log Loss 0.3312034872387728
reg_alpha = 1 Train Log Loss 0.26735883582307046 Test Log Loss 0.33056092032224055
reg_alpha = 5 Train Log Loss 0.2689006319588199 Test Log Loss 0.3292621067563762
reg_alpha = 10 Train Log Loss 0.27068857032690075 Test Log Loss 0.327132773020531
reg_alpha = 50 Train Log Loss 0.27974802262650905 Test Log Loss 0.3239947646462191
```

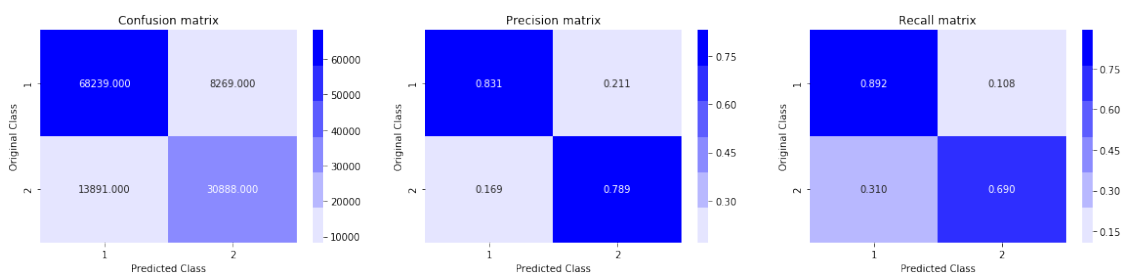
```
reg_alpha = 100 Train Log Loss 0.29039765847245264 Test Log Loss 0.32466225063664067
reg_alpha = 150 Train Log Loss 0.29908412709272575 Test Log Loss 0.3264270224443026
```

Out[19]: Text(0,0.5,'Log Loss')



```
In [18]: import xgboost as xgb
clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.02,n_estimators=400,n_jobs=-1)
clf.fit(X_train,y_train)
predict_y = clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
predicted_y = np.argmax(predict_y,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

The test log loss is: 0.3663235114698256



Sampled data and did further hyperparam tuning because of time constraints. random search for 15 models took around one day and then my system is not responding.

```
In [7]: # 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 LIMIT 100001;", conn_r)
                conn_r.commit()
                conn_r.close()

In [8]: # remove the first row
        data.drop(data.index[0], inplace=True)
        y_true = data['is_duplicate']
        data.drop(['Unnamed: 0', 'id', 'index', 'is_duplicate'], axis=1, inplace=True)

In [9]: # after we read from sql table each entry was read it as a string
        # we convert all the features into numeric before we apply any model
        cols = list(data.columns)
        data = pd.DataFrame(np.array(data.values, dtype=np.float64), columns=cols)

In [10]: y_true = list(map(int, y_true.values))

In [11]: X_train, X_test, y_train, y_test = train_test_split(data, y_true, stratify=y_true, test_size=0.3)

In [12]: print("Number of data points in train data :", X_train.shape)
        print("Number of data points in test data :", X_test.shape)

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

In [25]: estimators = [100, 150, 200, 300, 400, 600, 800]
        test_scores = []
        train_scores = []
        for i in estimators:
            clf = xgb.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=i, n_jobs=-1)
            clf.fit(X_train, y_train)
            predict_y = clf.predict_proba(X_train)
            log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
            train_scores.append(log_loss_train)
            predict_y = clf.predict_proba(X_test)
            log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
```

```

test_scores.append(log_loss_test)
print('estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(estimators,train_scores,label='Train Log Loss')
plt.plot(estimators,test_scores,label='Test Log Loss')
plt.xlabel('estimators')
plt.ylabel('Log Loss')

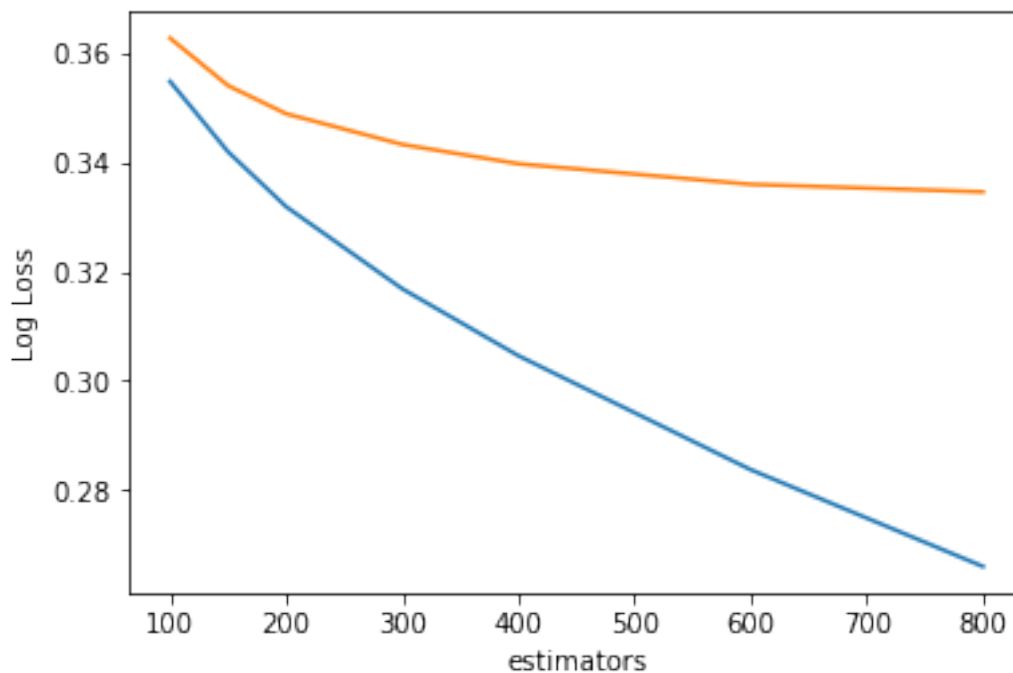
```

```

estimators = 100 Train Log Loss 0.3548561992467385 Test Log Loss 0.3627878957743776
estimators = 150 Train Log Loss 0.3419396997944057 Test Log Loss 0.35407105466366096
estimators = 200 Train Log Loss 0.3318797377522653 Test Log Loss 0.34891807315988954
estimators = 300 Train Log Loss 0.3168242151591801 Test Log Loss 0.343288478096738
estimators = 400 Train Log Loss 0.30455667545383736 Test Log Loss 0.33974266566115263
estimators = 600 Train Log Loss 0.283669436630139 Test Log Loss 0.3359795288017877
estimators = 800 Train Log Loss 0.26592577978587245 Test Log Loss 0.3345965160762109

```

Out[25]: Text(0,0.5,'Log Loss')



```

In [26]: test_scores = []
train_scores = []
etas = [0.05,0.1,0.15,0.2,0.25,0.3]
for i in etas:
    clf = xgb.XGBClassifier(max_depth=3,learning_rate=i,n_estimators=250,n_jobs=-1)
    clf.fit(X_train,y_train)
    predict_y = clf.predict_proba(X_train)

```

```

log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
train_scores.append(log_loss_train)
predict_y = clf.predict_proba(X_test)
log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
test_scores.append(log_loss_test)
print('Learning Rate = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_
plt.plot(etas,train_scores,label='Train Log Loss')
plt.plot(etas,test_scores,label='Test Log Loss')
plt.xlabel('Learning rate')
plt.ylabel('Log Loss')

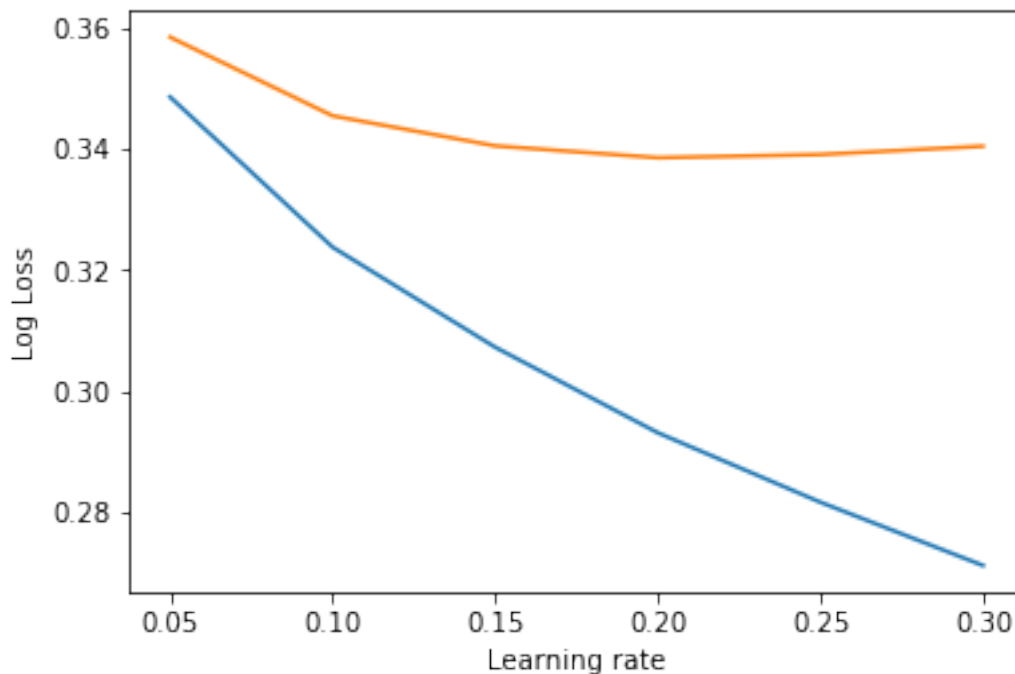
```

```

Learning Rate = 0.05 Train Log Loss 0.3486359981449816 Test Log Loss 0.3585428590734305
Learning Rate = 0.1 Train Log Loss 0.323761992687137 Test Log Loss 0.34551035487138326
Learning Rate = 0.15 Train Log Loss 0.30719720409078277 Test Log Loss 0.3405323742802464
Learning Rate = 0.2 Train Log Loss 0.2930345805370733 Test Log Loss 0.3385715798192348
Learning Rate = 0.25 Train Log Loss 0.2815342555628434 Test Log Loss 0.33910280387596653
Learning Rate = 0.3 Train Log Loss 0.27105970680499414 Test Log Loss 0.34047511766321725

```

Out[26]: Text(0,0.5,'Log Loss')



```

In [27]: test_scores = []
         train_scores = []
         alpha = [0.5,1,5,10,50,100,150]
         for i in alpha:

```

```

clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.95,n_estimators=250,reg_alpha=0.5)
clf.fit(X_train,y_train)
predict_y = clf.predict_proba(X_train)
log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
train_scores.append(log_loss_train)
predict_y = clf.predict_proba(X_test)
log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
test_scores.append(log_loss_test)
print('reg_alpha = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(alpha,train_scores,label='Train Log Loss')
plt.plot(alpha,test_scores,label='Test Log Loss')
plt.xlabel('reg_alpha')
plt.ylabel('Log Loss')

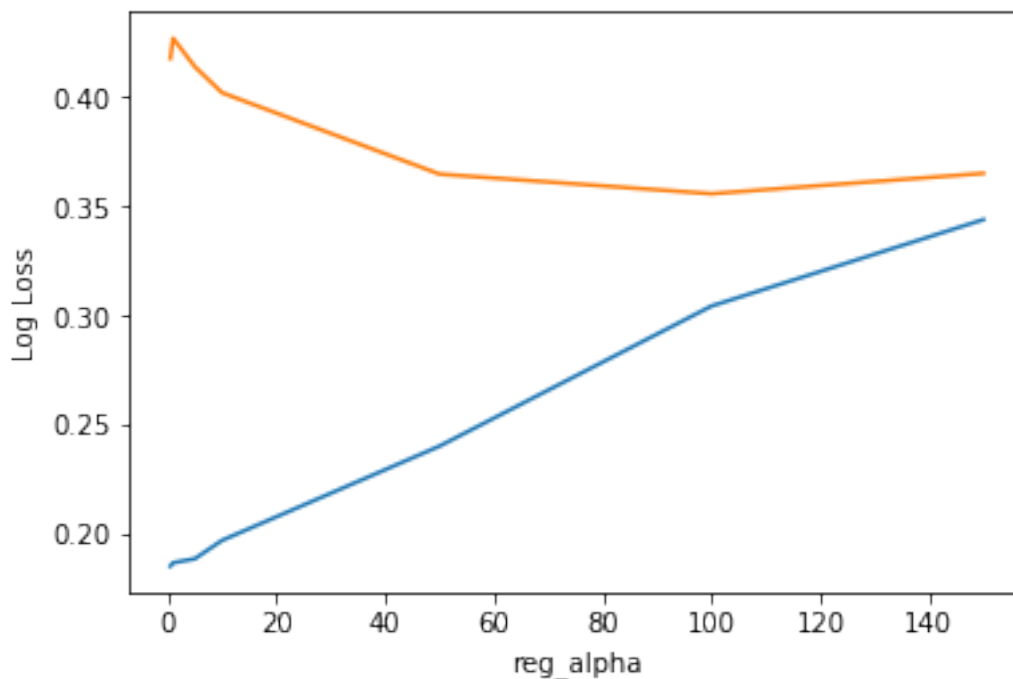
```

```

reg_alpha = 0.5 Train Log Loss 0.18552754110016914 Test Log Loss 0.41771510824008634
reg_alpha = 1 Train Log Loss 0.18708247507165138 Test Log Loss 0.42690521142320964
reg_alpha = 5 Train Log Loss 0.1889097248930446 Test Log Loss 0.4139238847946861
reg_alpha = 10 Train Log Loss 0.19732091498184656 Test Log Loss 0.4020169662291672
reg_alpha = 50 Train Log Loss 0.2404195343979278 Test Log Loss 0.364811240860891
reg_alpha = 100 Train Log Loss 0.3044636385472295 Test Log Loss 0.35583897909062384
reg_alpha = 150 Train Log Loss 0.3439979601721529 Test Log Loss 0.36513114237933575

```

Out[27]: Text(0,0.5,'Log Loss')



```

In [16]: param_dist = {"max_depth": sp_randint(2,5),
                        "learning_rate":uniform(0,0.2),
                        "n_estimators":sp_randint(200,350),
                        "min_child_weight": sp_randint(2, 8),
                        "gamma": uniform(0,4),
                        "subsample":uniform(0.7,0.3),
                        "colsample_bytree": uniform(0.7,0.3),
                        "reg_alpha":uniform(100,300),
                        "reg_lambda":uniform(100,300)}

model_rs_xgb = RandomizedSearchCV(xgb.XGBClassifier(n_jobs=-1,random_state=25), param_
                                n_iter=30,scoring='neg_log_loss',cv=5,n_jobs=-1)

model_rs_xgb.fit(X_train,y_train)
pickle.dump(model_rs_xgb,open('model_rs_xgb.p','wb'))

In [23]: dict_scores = []
        idx = 0
        for i in model_rs_xgb.grid_scores_:
            dict_score = []
            dict_score.append(i[0]['n_estimators'])
            dict_score.append(i[0]['max_depth'])
            dict_score.append(i[0]['subsample'])
            dict_score.append(i[0]['min_child_weight'])
            dict_score.append(i[0]['learning_rate'])
            dict_score.append(i[0]['reg_alpha'])
            dict_score.append(i[0]['reg_lambda'])
            dict_score.append(i[0]['gamma'])
            dict_score.append(i[0]['colsample_bytree'])
            dict_score.append(-i[1])
            dict_score.append((np.abs(i[2]).std()))
            dict_score.append(-model_rs_xgb.cv_results_['mean_train_score'][idx])
            dict_scores.append(dict_score)
            idx = idx + 1
        scores_df = pd.DataFrame(dict_scores,columns=['n_estimators','depth','subsample','min_
                                                    'learning_rate','reg_alpha','reg_lambda',
                                                    'colsample_bytree','Test_score',
                                                    'Test_std','Train_score'])

In [24]: scores_df.sort_values('Test_score').head(10)

Out[24]:
```

	n_estimators	depth	subsample	min_child_weight	learning_rate	\
21	312	4	0.862849	7	0.078998	
6	237	3	0.918557	7	0.093173	
2	252	4	0.861955	5	0.147171	
3	260	2	0.883727	5	0.162472	
1	263	3	0.989867	6	0.134493	
10	287	4	0.727813	3	0.083195	
0	335	4	0.836968	3	0.069301	

28	324	3	0.982237	3	0.060558
26	257	3	0.737091	3	0.094935
13	211	3	0.743472	7	0.082667

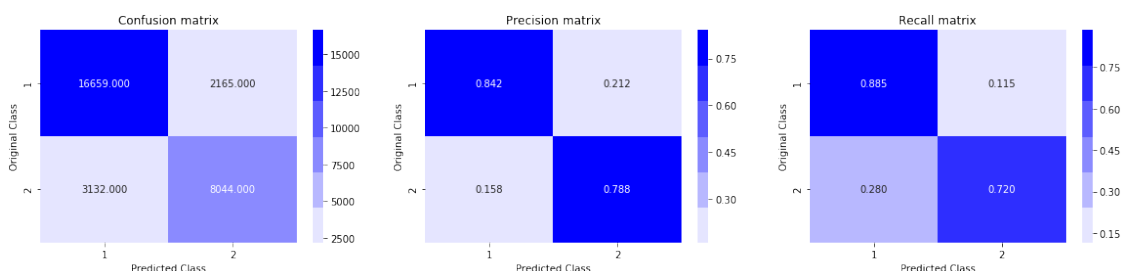
	reg_alpha	reg_lambda	gamma	colsample_bytree	Test_score	Test_std	\
21	151.329530	368.527334	2.475153	0.889555	0.365552	0.004887	
6	141.951036	247.381210	2.147542	0.959726	0.370005	0.005169	
2	211.574781	237.988483	1.297998	0.914696	0.371073	0.005258	
3	140.330037	125.075348	0.928819	0.723344	0.371593	0.004502	
1	201.904989	291.399625	1.215903	0.976700	0.374411	0.005102	
10	194.466489	393.672716	1.808000	0.963584	0.374655	0.005214	
0	215.509247	341.789604	2.932693	0.905324	0.375133	0.004916	
28	205.901274	176.926146	3.485774	0.748520	0.376651	0.004448	
26	183.424671	264.413431	1.303816	0.824543	0.377248	0.005051	
13	159.094695	283.155601	3.633829	0.982659	0.379051	0.004702	

	Train_score
21	0.355803
6	0.363034
2	0.363501
3	0.365183
1	0.368895
10	0.368341
0	0.369223
28	0.371700
26	0.372042
13	0.374420

```
In [27]: import xgboost as xgb
clf = xgb.XGBClassifier(max_depth=4,learning_rate=0.078998,n_estimators=312,
                        min_child_weight=7,subsample=0.862849,
                        reg_alpha=151.329530,reg_lambda=368.527334,
                        colsample_bytree=0.889555,gamma=2.475153,n_jobs=-1)

clf.fit(X_train,y_train)
predict_y = clf.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

The test log loss is: 0.362546103674608



0.0.1 With Tf-Idf features

```
In [22]: data.columns[0:26]
```

```
Out[22]: Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',  
               'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',  
               'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',  
               'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid1', 'freq_qid2',  
               'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',  
               'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2'],  
              dtype='object')
```

```
In [23]: #prepro_features_train.csv (Simple Preprocessing Feartures)
```

```
#nlp_features_train.csv (NLP Features)
```

```
if os.path.isfile('nlp_features_train.csv'):
```

```
    dfnlp = pd.read_csv("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("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 [24]: 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 [25]: df1.columns
```

```
Out[25]: Index(['id', 'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',  
               'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',  
               'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',  
               'fuzz_partial_ratio', 'longest_substr_ratio'],  
              dtype='object')
```

```
In [26]: df2.columns
```

```
Out[26]: Index(['id', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',  
               'q2_n_words', 'word_Common', 'word_Total', 'word_share', 'freq_q1+q2',  
               'freq_q1-q2'],  
              dtype='object')
```

```
In [27]: df3.columns
```

```
Out[27]: Index(['id', 'question1', 'question2'], dtype='object')
```

so for Tf-Idf Features i am combining question1 and question2, then getting Tf-Idf for for Train and transforming test.

```
In [28]: df3 = df3.fillna(' ')
df4 = pd.DataFrame()
df4['Text'] = df3.question1 + ' ' + df3.question2
df4['id'] = df3.id
```

Combining question1 and question2, then getting Tf-Idf

```
In [29]: df2['id']=df1['id']
df4['id']=df1['id']
df5 = df1.merge(df2, on='id',how='left')
final = df5.merge(df4, on='id',how='left')
```

```
In [30]: final.columns
```

```
Out[30]: Index(['id', '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', 'Text'],
              dtype='object')
```

```
In [31]: final = final.drop('id',axis=1)
```

```
In [32]: X_train_tf,X_test_tf, y_train_tf, y_test_tf = train_test_split(final,duplicate, strat.
```

```
In [33]: tfidf_vect = TfidfVectorizer(ngram_range=(1,3),max_features=200000,min_df=0.000032)
train_tfidf = tfidf_vect.fit_transform(X_train_tf.Text)
test_tfidf = tfidf_vect.transform(X_test_tf.Text)
print('No of Tfidf features',len(tfidf_vect.get_feature_names()))
```

No of Tfidf features 122967

```
In [34]: X_train_tf = X_train_tf.drop('Text',axis=1)
X_test_tf = X_test_tf.drop('Text',axis=1)
```

```
In [35]: from scipy.sparse import hstack
X_train1 = hstack((X_train_tf.values,train_tfidf))
X_test1 = hstack((X_test_tf.values,test_tfidf))
```

```
In [36]: X_train1
```

```
Out[36]: <283003x122993 sparse matrix of type '<class 'numpy.float64'>'
         with 16070821 stored elements in COOrdinate format>
```



```

In [37]: scale = StandardScaler(with_mean=False)
        X_train_sc = scale.fit_transform(X_train1)
        X_test_sc = scale.transform(X_test1)

In [180]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generators.html
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

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

predict_y = sig_clf.predict_proba(X_train_sc)

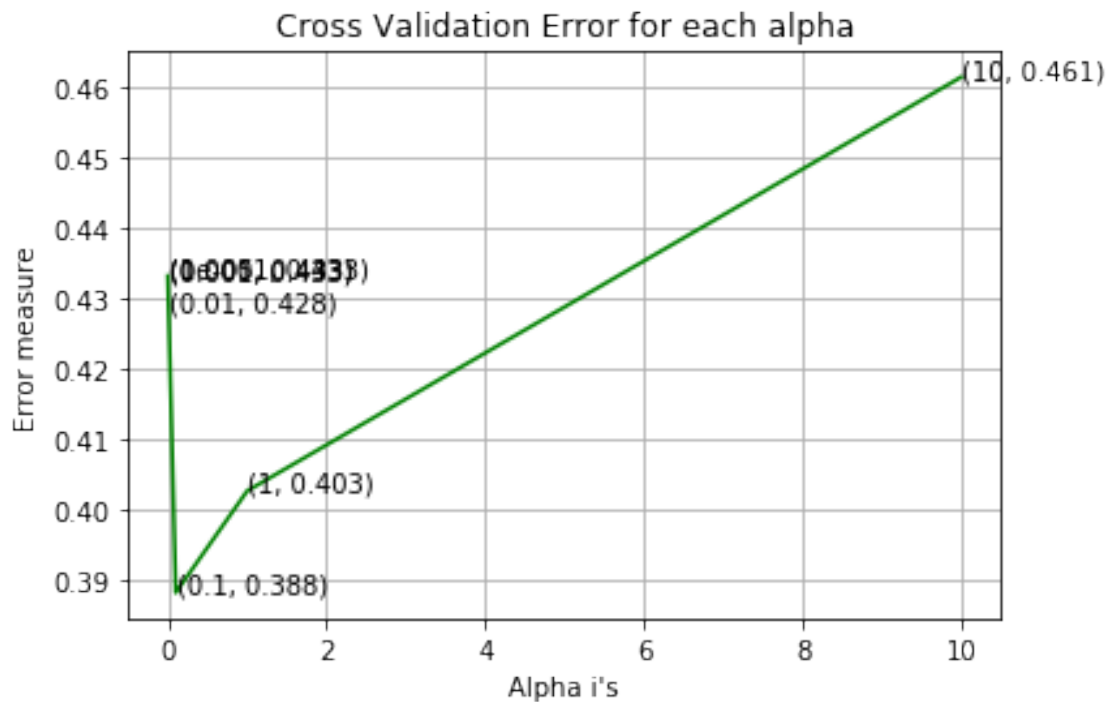
```

```

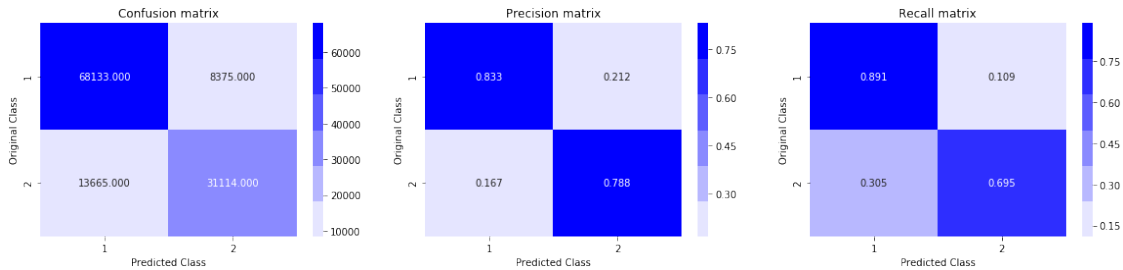
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(X_test_sc)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

For values of alpha = 1e-05 The log loss is: 0.43314008705729606
 For values of alpha = 0.0001 The log loss is: 0.4331272796939656
 For values of alpha = 0.001 The log loss is: 0.4327302472283095
 For values of alpha = 0.01 The log loss is: 0.4283776670722629
 For values of alpha = 0.1 The log loss is: 0.3881322803856421
 For values of alpha = 1 The log loss is: 0.40257550672626424
 For values of alpha = 10 The log loss is: 0.4613941629266127



For values of best alpha = 0.1 The train log loss is: 0.26985162323505474
 For values of best alpha = 0.1 The test log loss is: 0.3881322803856421
 Total number of data points : 121287



```
In [182]: #alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
```

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

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

```
#-----
# video link:
#-----
alpha = np.random.uniform(0.05,0.5,14)
alpha = np.round(alpha,4)
alpha.sort()
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_sc, y_train)
    predict_y = sig_clf.predict_proba(X_test_sc)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y,
```

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

```

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

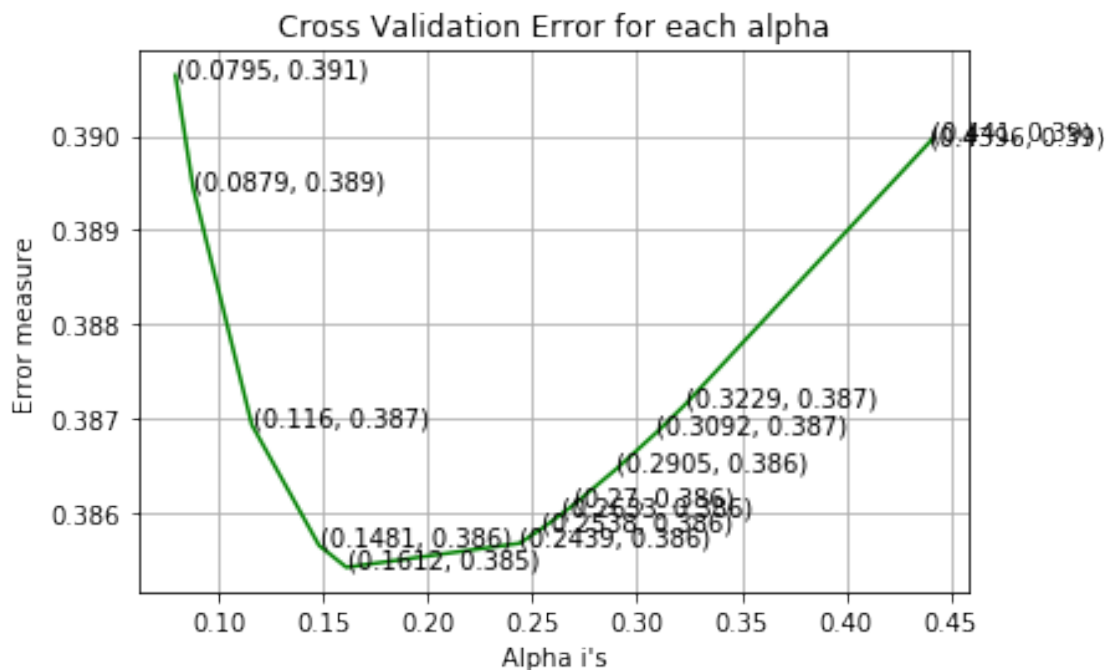
predict_y = sig_clf.predict_proba(X_train_sc)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(X_test_sc)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

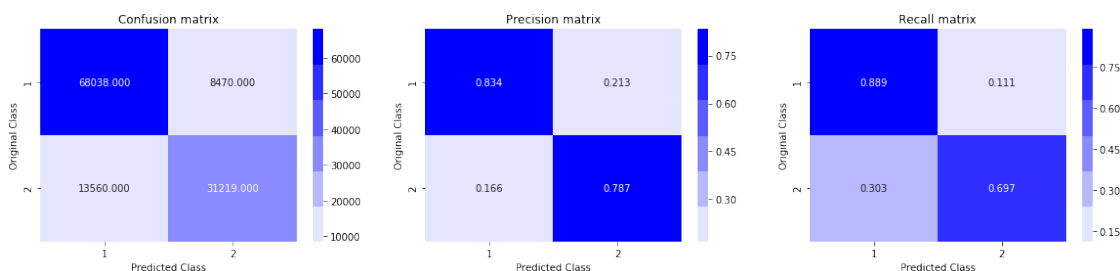
```

For values of alpha = 0.0795 The log loss is: 0.39064417583295863
For values of alpha = 0.0879 The log loss is: 0.3894558074210688
For values of alpha = 0.116 The log loss is: 0.3869327250039135
For values of alpha = 0.1481 The log loss is: 0.38565275085466144
For values of alpha = 0.1612 The log loss is: 0.3854195068372494
For values of alpha = 0.2439 The log loss is: 0.3856745506873943
For values of alpha = 0.2538 The log loss is: 0.38581966177962984
For values of alpha = 0.2633 The log loss is: 0.3859735181868948
For values of alpha = 0.27 The log loss is: 0.38608826320451
For values of alpha = 0.2905 The log loss is: 0.38646998649269443
For values of alpha = 0.3092 The log loss is: 0.3868477094820089
For values of alpha = 0.3229 The log loss is: 0.387144248260163
For values of alpha = 0.4396 The log loss is: 0.389928810977827
For values of alpha = 0.441 The log loss is: 0.389963612599141

```



For values of best alpha = 0.1612 The train log loss is: 0.2749225188463317
 For values of best alpha = 0.1612 The test log loss is: 0.3854195068372494
 Total number of data points : 121287



it was giving some good scores but it seems to be some overfitting in this.
 i think it may be because of feature scaling for all the data including tfidf values so i tried without feature scaling

```
In [183]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss=‘hinge’, penalty=‘l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True)
```

```

# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=0.1
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train1, y_train)
    predict_y = sig_clf.predict_proba(X_test1)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_test, predict_y))

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

best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train1, y_train)

predict_y = sig_clf.predict_proba(X_train1)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_test1)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
predicted_y = np.argmax(predict_y, axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

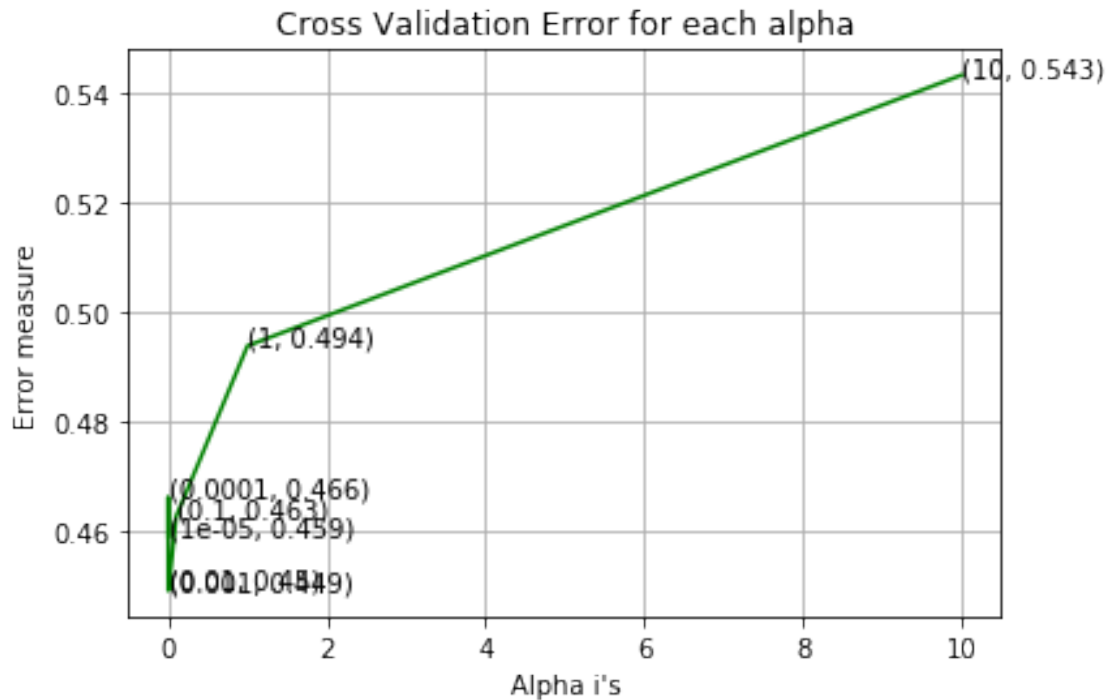
```

```

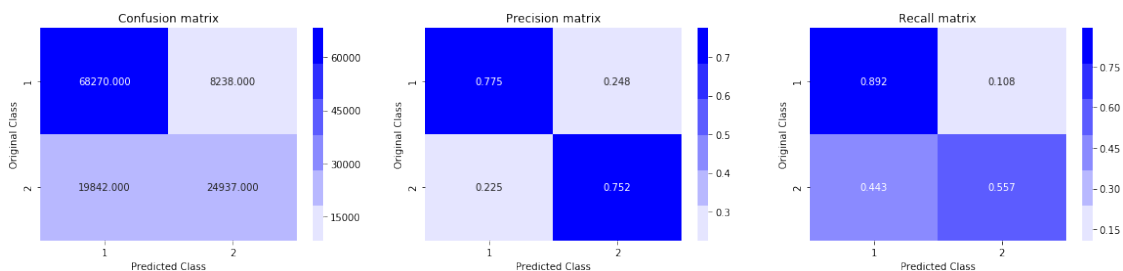
For values of alpha = 1e-05 The log loss is: 0.45919822041900693
For values of alpha = 0.0001 The log loss is: 0.4663672071140698
For values of alpha = 0.001 The log loss is: 0.4492402172589541

```

For values of alpha = 0.01 The log loss is: 0.44994060193898233
 For values of alpha = 0.1 The log loss is: 0.46275295853045256
 For values of alpha = 1 The log loss is: 0.4939560730721335
 For values of alpha = 10 The log loss is: 0.5434258589612511



For values of best alpha = 0.001 The train log loss is: 0.4489866787327462
 For values of best alpha = 0.001 The test log loss is: 0.4492402172589541
 Total number of data points : 121287



i decreased overfittig but bias increased. Tought like i will scale features otherthan Tf-Idf and Tf-Idf was already coming with l2 normalization. so tried with this format below

```

In [38]: scale = StandardScaler()
        X_train_some = scale.fit_transform(X_train_tf)
        X_test_some = scale.transform(X_test_tf)

In [39]: from scipy.sparse import hstack
        X_train2 = hstack((X_train_some, train_tfidf))
        X_test2 = hstack((X_test_some, test_tfidf))

In [193]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train2, y_train)
    predict_y = sig_clf.predict_proba(X_test2)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_test, predict_y))

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

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

```



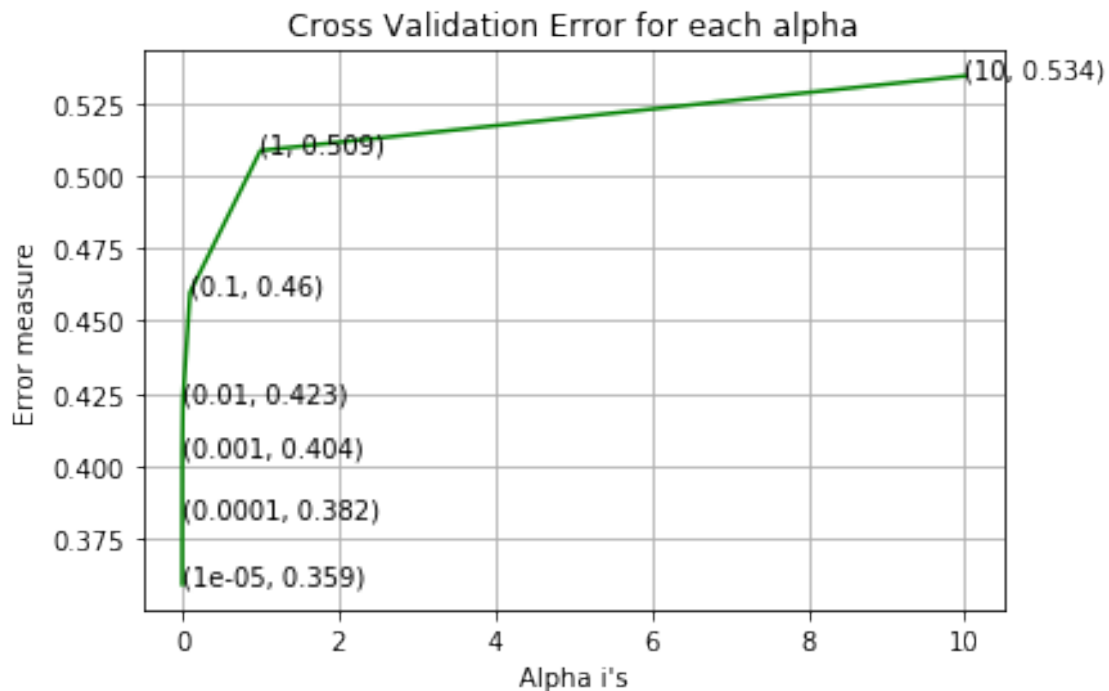
```

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train2, y_train)

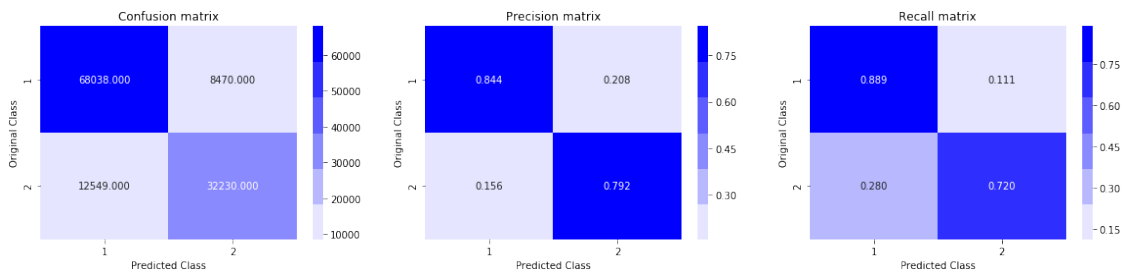
predict_y = sig_clf.predict_proba(X_train2)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(predict_y, y_train))
predict_y = sig_clf.predict_proba(X_test2)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(predict_y, y_test))
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: 0.3592485420041076
 For values of alpha = 0.0001 The log loss is: 0.38220592755894905
 For values of alpha = 0.001 The log loss is: 0.4040174130069092
 For values of alpha = 0.01 The log loss is: 0.4228198041131542
 For values of alpha = 0.1 The log loss is: 0.4598042497583823
 For values of alpha = 1 The log loss is: 0.5086852562507405
 For values of alpha = 10 The log loss is: 0.534388446481064



For values of best alpha = 1e-05 The train log loss is: 0.3395431324943519
 For values of best alpha = 1e-05 The test log loss is: 0.3592485420041076
 Total number of data points : 121287



Now it seems to be good

In [40]: `#alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.`

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

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

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

alpha = np.random.uniform(0.000002,0.00003,14)
alpha = np.round(alpha,8)
alpha.sort()

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train2, y_train)
    predict_y = sig_clf.predict_proba(X_test2)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, eps=1e-15))

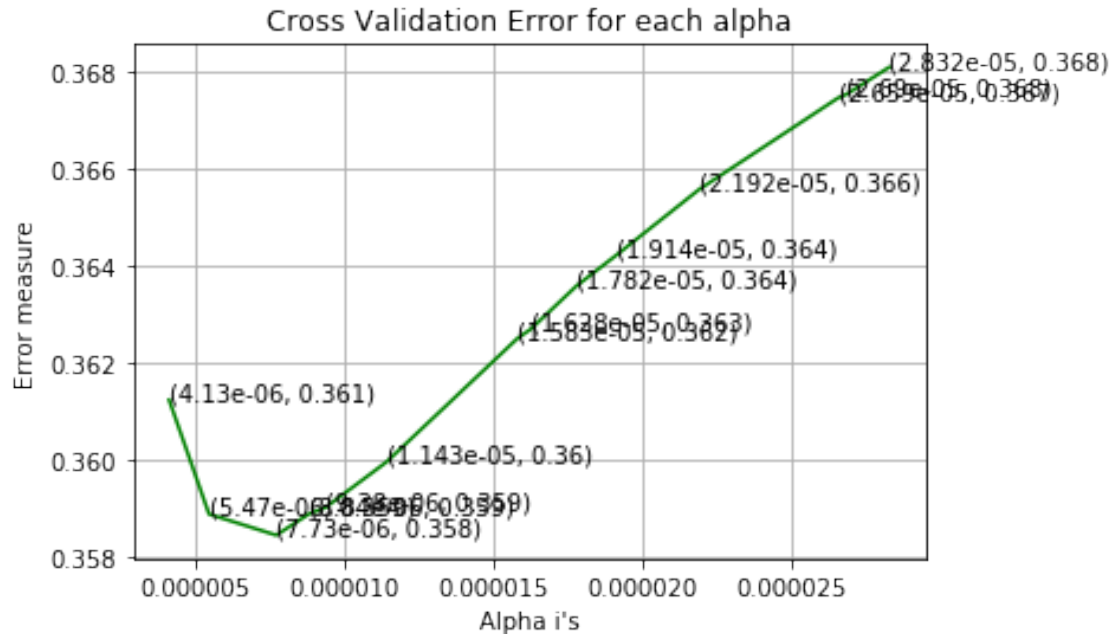
fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
```

```
plt.ylabel("Error measure")
plt.show()
```

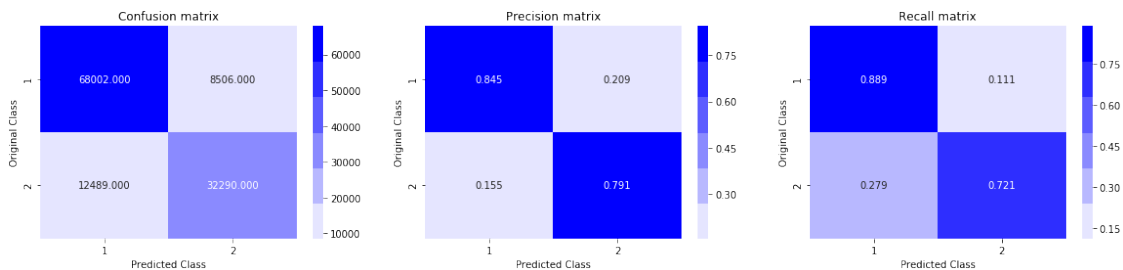
```
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train2, y_train)
```

```
predict_y = sig_clf.predict_proba(X_train2)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_test2)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
predicted_y = np.argmax(predict_y, axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha = 4.13e-06 The log loss is: 0.36122866843967544
For values of alpha = 5.47e-06 The log loss is: 0.3588790251668011
For values of alpha = 7.73e-06 The log loss is: 0.3584450374617226
For values of alpha = 8.84e-06 The log loss is: 0.3588864116809375
For values of alpha = 9.38e-06 The log loss is: 0.3590125699678223
For values of alpha = 1.143e-05 The log loss is: 0.35994931400868724
For values of alpha = 1.583e-05 The log loss is: 0.3624907902604967
For values of alpha = 1.628e-05 The log loss is: 0.3626844556970233
For values of alpha = 1.782e-05 The log loss is: 0.36357454612123535
For values of alpha = 1.914e-05 The log loss is: 0.3641980673263202
For values of alpha = 2.192e-05 The log loss is: 0.36554921230584847
For values of alpha = 2.659e-05 The log loss is: 0.3674344684859557
For values of alpha = 2.69e-05 The log loss is: 0.3675203002315432
For values of alpha = 2.832e-05 The log loss is: 0.3680707978055651
```



For values of best alpha = 7.73e-06 The train log loss is: 0.3368825243663587
 For values of best alpha = 7.73e-06 The test log loss is: 0.3584450374617226
 Total number of data points : 121287



Linear SVM with Hyperparameter tuning:

In [41]: `alpha = [10 ** x for x in range(-5, 2)]` # hyperparam for SGD classifier.

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

```

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

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

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train2, y_train)
    predict_y = sig_clf.predict_proba(X_test2)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_test, predict_y, eps=1e-15))

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

best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train2, y_train)

predict_y = sig_clf.predict_proba(X_train2)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test2)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, 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)

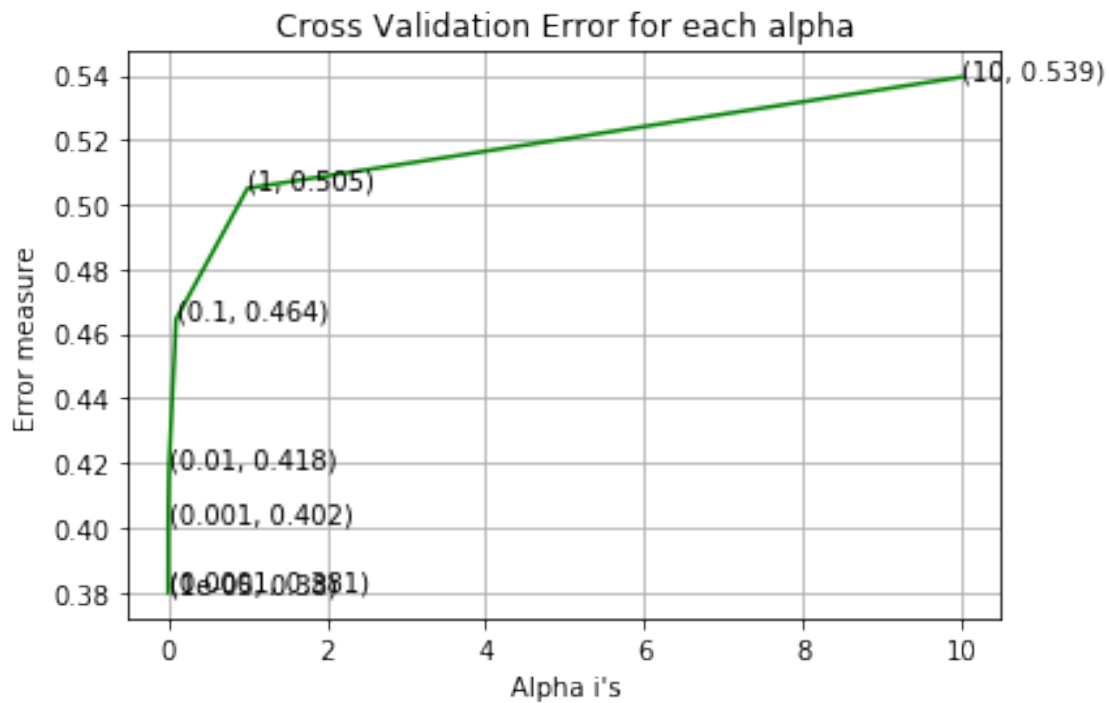
```

```

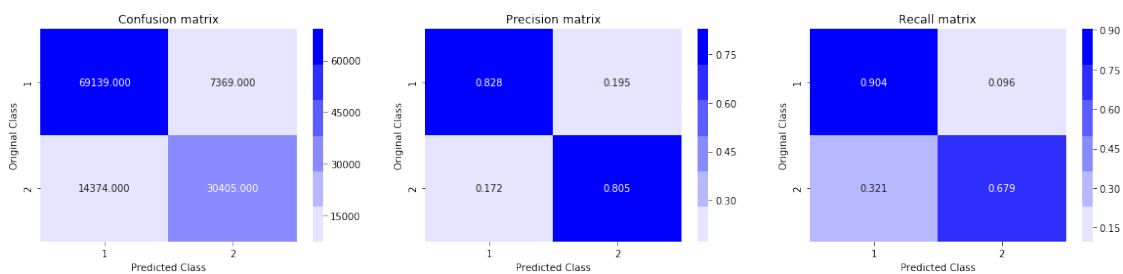
For values of alpha = 1e-05 The log loss is: 0.3798066793143051
For values of alpha = 0.0001 The log loss is: 0.38116712269147246
For values of alpha = 0.001 The log loss is: 0.40184830286142403
For values of alpha = 0.01 The log loss is: 0.41845776341903373
For values of alpha = 0.1 The log loss is: 0.4644497805778554

```

For values of alpha = 1 The log loss is: 0.5049977600164108
 For values of alpha = 10 The log loss is: 0.5394307111628995



For values of best alpha = 1e-05 The train log loss is: 0.3607393522040741
 For values of best alpha = 1e-05 The test log loss is: 0.3798066793143051
 Total number of data points : 121287



In [42]: `#alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.`

`# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated,
 # -----`

```

# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

alpha = np.random.uniform(0.000002,0.00003,14)
alpha = np.round(alpha,8)
alpha.sort()

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train2, y_train)
    predict_y = sig_clf.predict_proba(X_test2)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, eps=1e-15))

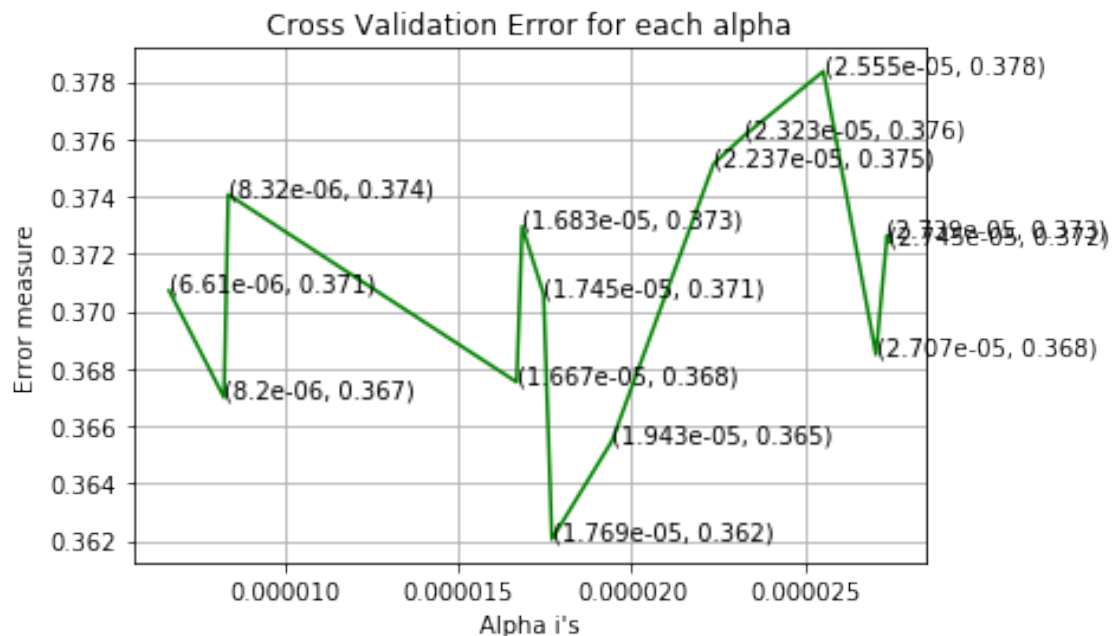
fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train2, y_train)

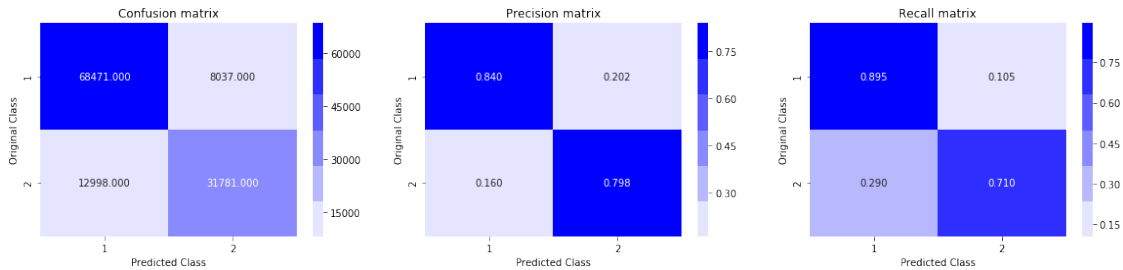
predict_y = sig_clf.predict_proba(X_train2)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test2)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, 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 = $6.61e-06$ The log loss is: 0.3707146846644485
 For values of alpha = $8.2e-06$ The log loss is: 0.36699067890864273
 For values of alpha = $8.32e-06$ The log loss is: 0.3740658586177854
 For values of alpha = $1.667e-05$ The log loss is: 0.36752693609047854
 For values of alpha = $1.683e-05$ The log loss is: 0.3729555362549245
 For values of alpha = $1.745e-05$ The log loss is: 0.3705911233029048
 For values of alpha = $1.769e-05$ The log loss is: 0.36204932154873615
 For values of alpha = $1.943e-05$ The log loss is: 0.36547605004579575
 For values of alpha = $2.237e-05$ The log loss is: 0.3751313041730113
 For values of alpha = $2.323e-05$ The log loss is: 0.3760910650204779
 For values of alpha = $2.555e-05$ The log loss is: 0.37836011165769023
 For values of alpha = $2.707e-05$ The log loss is: 0.3684866481434041
 For values of alpha = $2.739e-05$ The log loss is: 0.37263448797034704
 For values of alpha = $2.745e-05$ The log loss is: 0.3724051898117524



For values of best alpha = $1.769e-05$ The train log loss is: 0.343773971608721
 For values of best alpha = $1.769e-05$ The test log loss is: 0.36204932154873615
 Total number of data points : 121287



With some others features:

```
In [6]: #prepro_features_train.csv (Simple Preprocessing Feartures)
        #nlp_features_train.csv (NLP Features)
        if os.path.isfile('nlp_features_train.csv'):
            dfnlp = pd.read_csv("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("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 [7]: def remove_stop(sent):
        sent = str(sent)
        if sent == None:
            return ' '
        if sent==np.nan:
            return ' '
        if sent == 'NaN':
            return ' '
        z = [i for i in sent.split() if i not in STOP_WORDS]
        return ' '.join(z)

In [8]: dfnlp['question1'] = dfnlp.question1.apply(remove_stop)

In [9]: dfnlp['question2'] = dfnlp.question2.apply(remove_stop)

In [10]: from gensim.models import Word2Vec
         from gensim.models import KeyedVectors
         import pickle
         import gensim

In [28]: !wget http://nlp.stanford.edu/data/glove.840B.300d.zip

--2018-07-10 11:38:55-- http://nlp.stanford.edu/data/glove.840B.300d.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
```

```

Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.840B.300d.zip [following]
--2018-07-10 11:38:55-- https://nlp.stanford.edu/data/glove.840B.300d.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2176768927 (2.0G) [application/zip]
Saving to: glove.840B.300d.zip

```

```

100%[=====>] 2,176,768,927 6.10MB/s in 9m 16s

```

```

2018-07-10 11:48:11 (3.74 MB/s) - glove.840B.300d.zip saved [2176768927/2176768927]

```

```

In [36]: !unzip glove.840B.300d.zip

```

```

Archive: glove.840B.300d.zip
  inflating: glove.840B.300d.txt

```

```

In [37]: from gensim.scripts.glove2word2vec import glove2word2vec
         glove2word2vec(glove_input_file="glove.840B.300d.txt", word2vec_output_file="glove_vectors.txt")

```

```

Out[37]: (2196017, 300)

```

```

In [11]: from gensim.models.keyedvectors import KeyedVectors
         glove_model = KeyedVectors.load_word2vec_format("glove_vectors.txt", binary=False)

```

```

In [12]: def wmd(s1, s2, model):
         s1 = str(s1)
         s2 = str(s2)
         s1 = s1.split()
         s2 = s2.split()
         return model.wmdistance(s1, s2)

```

<http://proceedings.mlr.press/v37/kusnerb15.pdf> i read about word mover distance and after that i calculated some distances from avg word vectors as below

```

In [13]: dfnlp['Word_Mover_Dist'] = dfnlp.apply(lambda x: wmd(x['question1'], x['question2']), g

```

```

In [14]: # the avg-w2v for each sentence/review is stored in this list
         def avg_w2v(list_of_sent, model, d):
             """
             Returns average of word vectors for
             each sentence with dimension of model given
             """
             sent_vectors = []
             for sent in list_of_sent: # for each review/sentence

```

```

        doc = [word for word in sent if word in model.wv.vocab]
        if doc:
            sent_vec = np.mean(model.wv[doc],axis=0)
        else:
            sent_vec = np.zeros(d)
        sent_vectors.append(sent_vec)
    return sent_vectors

In [15]: #converting into lists
list_of_question1=[]
for sent in dfnlp.question1.values:
    list_of_question1.append(sent.split())
list_of_question2=[]
for sent in dfnlp.question2.values:
    list_of_question2.append(sent.split())

In [17]: #avg word 2 vec
avgw2v_q1 = avg_w2v(list_of_question1,glove_model,300)
avgw2v_q2 = avg_w2v(list_of_question2,glove_model,300)

In [25]: #converting as df
df_avgw2v = pd.DataFrame()
df_avgw2v['q1_vec'] = list(avgw2v_q1)
df_avgw2v['q2_vec'] = list(avgw2v_q2)
df_q1 = pd.DataFrame(df_avgw2v.q1_vec.values.tolist())
df_q2 = pd.DataFrame(df_avgw2v.q2_vec.values.tolist())

In [28]: #importing soma distances and calculating
from scipy.stats import skew, kurtosis
from scipy.spatial.distance import cosine, cityblock, canberra, euclidean, minkowski
dfnlp['dist_cosine'] = [cosine(x, y) for (x, y) in zip(avgw2v_q1, avgw2v_q2)]
dfnlp['dist_cityblock'] = [cityblock(x, y) for (x, y) in zip(avgw2v_q1, avgw2v_q2)]
dfnlp['dist_canberra'] = [canberra(x, y) for (x, y) in zip(avgw2v_q1, avgw2v_q2)]
dfnlp['dist_euclidean'] = [euclidean(x, y) for (x, y) in zip(avgw2v_q1, avgw2v_q2)]
dfnlp['dist_minkowski'] = [minkowski(x, y) for (x, y) in zip(avgw2v_q1, avgw2v_q2)]

In [42]: #filling na values with 0 for cosine distance
dfnlp.dist_cosine = dfnlp.dist_cosine.fillna(0)

In [44]: #merzing all df
df_q1.reset_index(inplace=True)
df_q2.reset_index(inplace=True)
df_q1['index'] = df_q2['index']
df_avgw2v_final = df_q1.merge(df_q2, on='index', how='left')

In [51]: #for final df
df1 = dfnlp.drop(['qid1', 'qid2', 'question1', 'question2'], axis=1)
df2 = dfppro.drop(['qid1', 'qid2', 'question1', 'question2', 'is_duplicate'], axis=1)

```

```

In [52]: ##merging all
         df1.id = df_avgw2v.index
         df2.id = df_avgw2v.index
         df_temp = df1.merge(df2,on='id',how='left')
         df_final = df_temp.merge(df_avgw2v_final,left_on='id',right_on='index',how='left')

In [56]: #saving to disk
         df_final.to_csv('df_final_avg.csv',index=False)

In [45]: df_final = pd.read_csv('df_final_avg.csv')

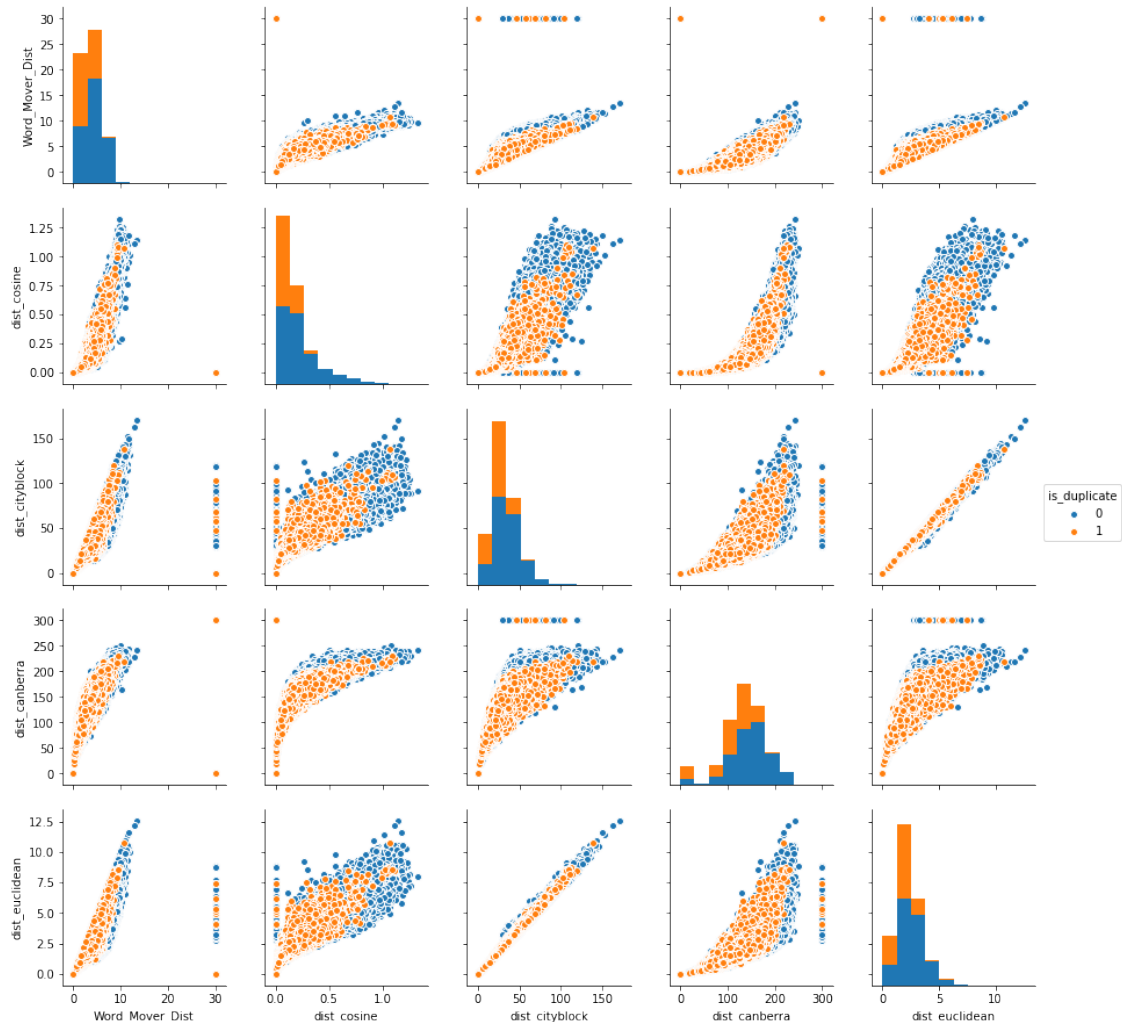
In [47]: ## max no after inf is 13.45 so imputed infinity with 30
         df_final.Word_Mover_Dist = df_final.Word_Mover_Dist.apply(lambda x: 30 if x == np.inf

In [10]: #set of values
         np.sort(list(set(df_final.Word_Mover_Dist.values)))

Out[10]: array([ 0.          ,  0.10251455,  0.11491957, ..., 12.86403772,
                13.45192544, 30.          ])

In [48]: n = df_final.shape[0]
         sns.pairplot(df_final[['Word_Mover_Dist', 'dist_cosine', 'dist_cityblock',
                                'dist_canberra','dist_euclidean', 'is_duplicate']][0:n],
                        hue='is_duplicate', vars=['Word_Mover_Dist', 'dist_cosine', 'dist_citybl
plt.show()

```



```
In [8]: ##removing dependent variable
duplicate = df_final.is_duplicate
df_final = df_final.drop(['id', 'is_duplicate', 'index'],axis=1)

In [11]: X_train,X_test, y_train, y_test = train_test_split(df_final,duplicate, stratify=duplicate)

In [12]: 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[0])/test_len, "Class 1: ",int(test_distr[1])/test_len)

----- Distribution of output variable in train data -----
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
```

```
----- Distribution of output variable in train data -----
Class 0: 0.3691986775169639 Class 1: 0.3691986775169639
```

```
In [13]: scale_col = ['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max', 'last_w',
                    'first_word_eq', 'abs_len_diff', 'mean_len', 'token_set_ratio', 'token_s',
                    'fuzz_ratio', 'fuzz_partial_ratio', 'longest_substr_ratio', 'Word_Mover_L',
                    'dist_cosine', 'dist_cityblock', 'dist_canberra', 'dist_euclidean',
                    'dist_minkowski', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
                    'q2_n_words', 'word_Common', 'word_Total', 'word_share', 'freq_q1+q2', '']
```

```
In [14]: X_train_scale = X_train[scale_col]
        X_test_scale = X_test[scale_col]
```

```
In [15]: X_train_w2v = X_train.drop(scale_col,axis=1)
        X_test_w2v = X_test.drop(scale_col,axis=1)
```

```
In [16]: from sklearn.preprocessing import StandardScaler
        scale = StandardScaler()
        X_train_sc = scale.fit_transform(X_train_scale)
        X_test_sc = scale.transform(X_test_scale)
        X_train_sc = pd.DataFrame(X_train_sc,columns=X_train_scale.columns)
        X_test_sc = pd.DataFrame(X_test_sc,columns=X_test_scale.columns)
```

```
In [17]: ## Final train and test vectors after scaling of normal features
        X_train-fi = pd.DataFrame(np.hstack((X_train_sc.values,X_train_w2v.values)),columns=df_f
        X_test-fi = pd.DataFrame(np.hstack((X_test_sc.values,X_test_w2v.values)),columns=df_f
```

0.0.2 Logistic Regression:

```
In [103]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

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

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

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

        log_error_array=[]
```

```

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

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

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

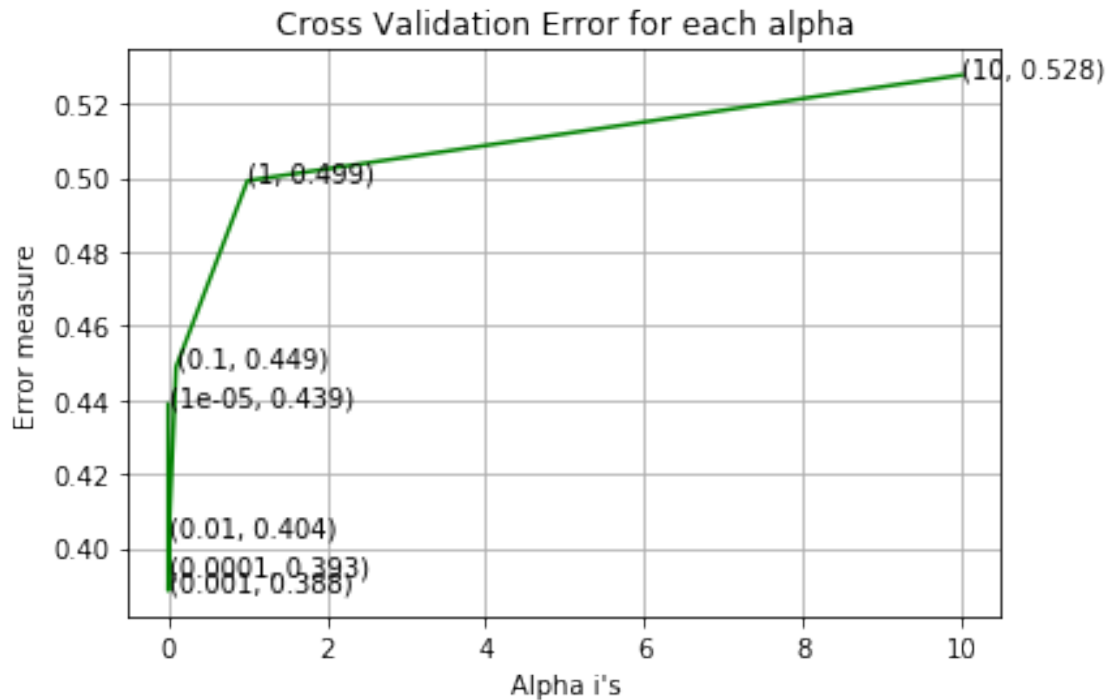
predict_y = sig_clf.predict_proba(X_train_fi)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log
predict_y = sig_clf.predict_proba(X_test_fi)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log
predicted_y = np.argmax(predict_y, axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

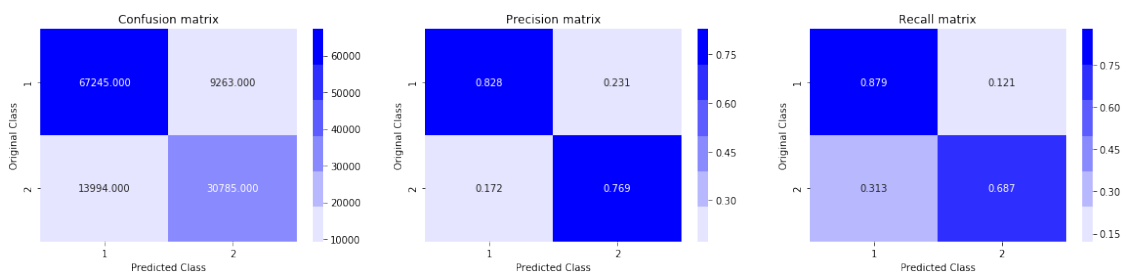
```

For values of alpha = 1e-05 The log loss is: 0.4388024128087525
For values of alpha = 0.0001 The log loss is: 0.39289765345196564
For values of alpha = 0.001 The log loss is: 0.3884671083090384
For values of alpha = 0.01 The log loss is: 0.4036735395413484
For values of alpha = 0.1 The log loss is: 0.4488595248965405
For values of alpha = 1 The log loss is: 0.4991565448915637
For values of alpha = 10 The log loss is: 0.5276589765281985

```



For values of best alpha = 0.001 The train log loss is: 0.38681517329784765
 For values of best alpha = 0.001 The test log loss is: 0.3884671083090384
 Total number of data points : 121287



In [104]: `#alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.`

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



```

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

#-----
# video link:
#-----
alpha = np.random.uniform(0.0005,0.005,14)
alpha = np.round(alpha,6)
alpha.sort()
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train_fi, y_train)
    predict_y = sig_clf.predict_proba(X_test_fi)
    log_error_array.append(log_loss(y_test, predict_y, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y))

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

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

predict_y = sig_clf.predict_proba(X_train_fi)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_test_fi)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

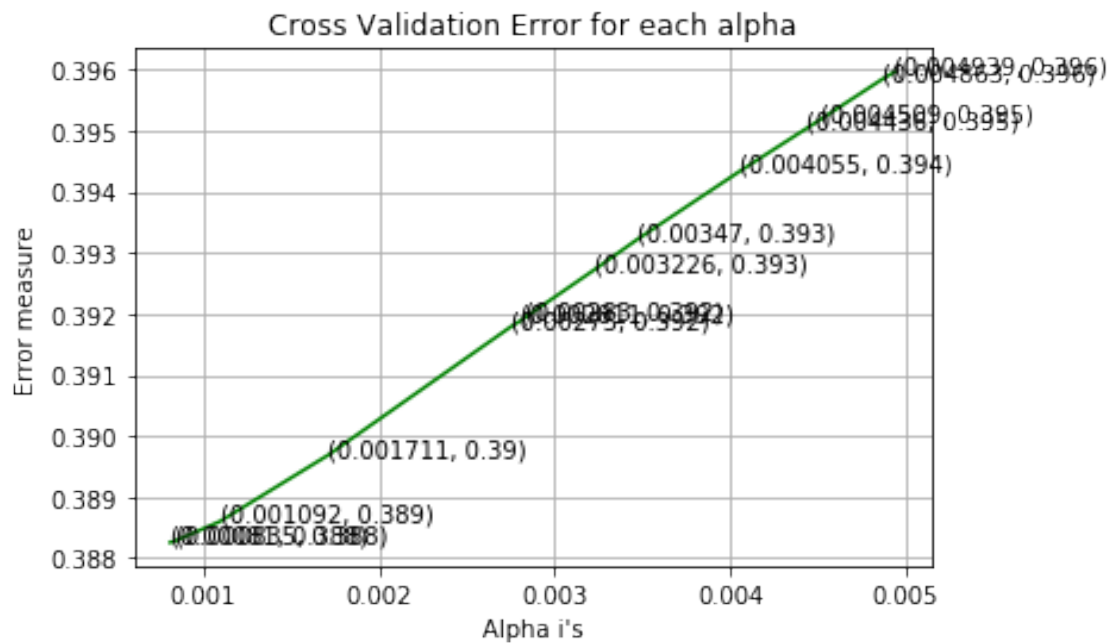
```

```

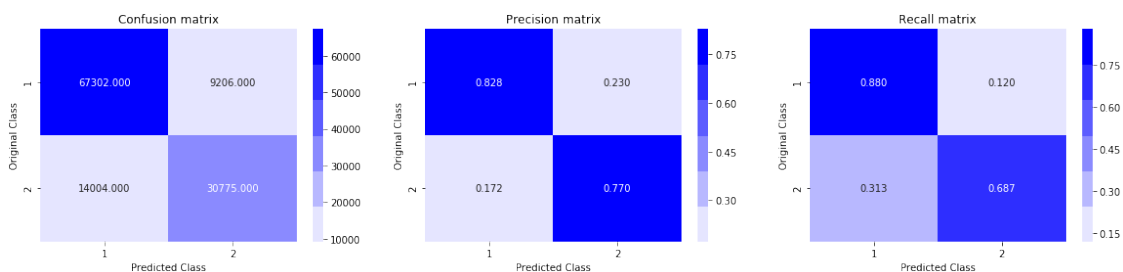
For values of alpha = 0.00081 The log loss is: 0.38825354941894474
For values of alpha = 0.000835 The log loss is: 0.3882756602391445
For values of alpha = 0.001092 The log loss is: 0.3885959531928289
For values of alpha = 0.001711 The log loss is: 0.3896869070388385

```

For values of alpha = 0.00275 The log loss is: 0.3917852442253969
 For values of alpha = 0.002811 The log loss is: 0.39189532200667276
 For values of alpha = 0.00283 The log loss is: 0.39193474315728954
 For values of alpha = 0.003226 The log loss is: 0.39271932002087345
 For values of alpha = 0.00347 The log loss is: 0.3932148339643286
 For values of alpha = 0.004055 The log loss is: 0.39433385702572826
 For values of alpha = 0.004436 The log loss is: 0.3950361601874909
 For values of alpha = 0.004509 The log loss is: 0.3951700685766183
 For values of alpha = 0.004863 The log loss is: 0.39583426043986775
 For values of alpha = 0.004939 The log loss is: 0.39596458548444097



For values of best alpha = 0.00081 The train log loss is: 0.3865499173703697
 For values of best alpha = 0.00081 The test log loss is: 0.38825354941894474
 Total number of data points : 121287



0.0.3 SVM:

```
In [105]: alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
```

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

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

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

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

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

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

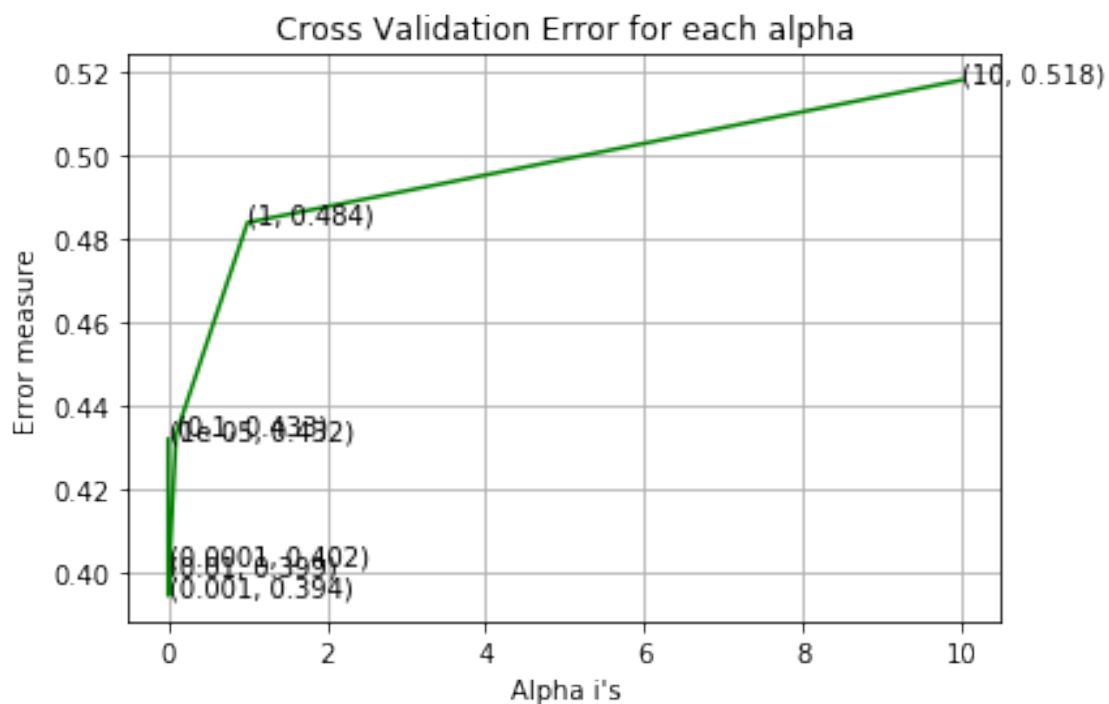
predict_y = sig_clf.predict_proba(X_train_fi)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(X_test_fi)
```

```

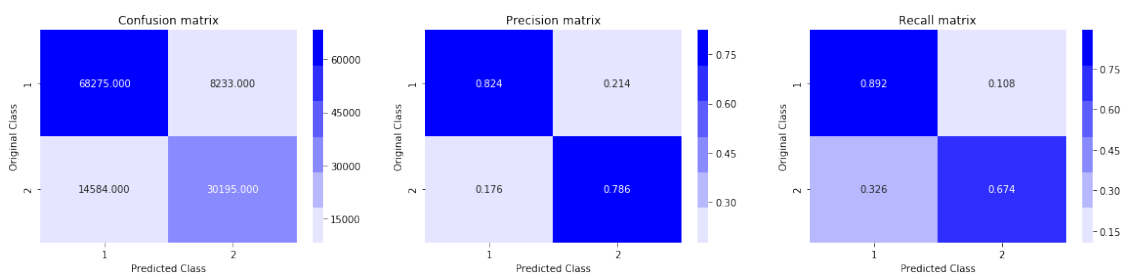
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

For values of alpha = 1e-05 The log loss is: 0.43193065554448745
 For values of alpha = 0.0001 The log loss is: 0.4018598817909984
 For values of alpha = 0.001 The log loss is: 0.3944586702255559
 For values of alpha = 0.01 The log loss is: 0.39931692189061074
 For values of alpha = 0.1 The log loss is: 0.4331257577821657
 For values of alpha = 1 The log loss is: 0.4837845291193158
 For values of alpha = 10 The log loss is: 0.5178851954488961



For values of best alpha = 0.001 The train log loss is: 0.39233257202331845
 For values of best alpha = 0.001 The test log loss is: 0.3944586702255559
 Total number of data points : 121287



0.0.4 XGBoost

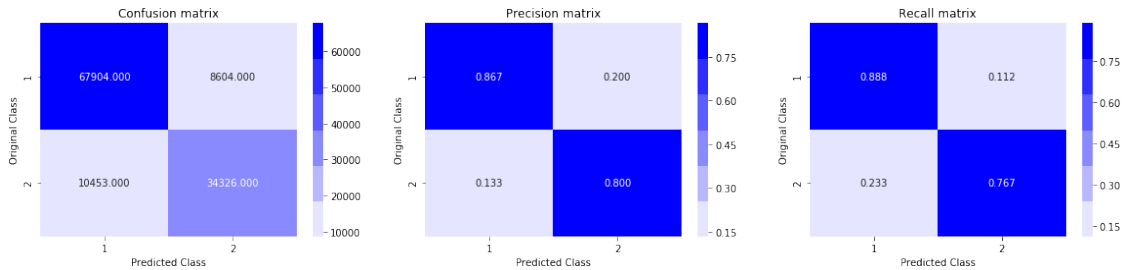
```
In [19]: import xgboost as xgb
```

```
In [21]: estimators = [100,150,200,300,400,600,800]
test_scores = []
train_scores = []
for i in estimators:
    clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.1,n_estimators=i,n_jobs=-1)
    clf.fit(X_train_fi,y_train)
    predict_y = clf.predict_proba(X_train_fi)
    log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
    train_scores.append(log_loss_train)
    predict_y = clf.predict_proba(X_test_fi)
    log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
    test_scores.append(log_loss_test)
    print('estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(estimators,train_scores,label='Train Log Loss')
```

```
estimators = 100 Train Log Loss 0.35271536543312704 Test Log Loss 0.3543473342068226
estimators = 150 Train Log Loss 0.3401001847837466 Test Log Loss 0.34269603440192314
estimators = 200 Train Log Loss 0.3321450871899908 Test Log Loss 0.33594061497805644
estimators = 300 Train Log Loss 0.3211245378472585 Test Log Loss 0.3272018539308977
estimators = 400 Train Log Loss 0.313823352470929 Test Log Loss 0.3222817257843711
estimators = 600 Train Log Loss 0.3034313380014822 Test Log Loss 0.31683156878039515
estimators = 800 Train Log Loss 0.29538370418647064 Test Log Loss 0.31334181243708586
```

```
In [23]: import xgboost as xgb
clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.12,n_estimators=600,
                        min_child_weight=5,
                        reg_alpha=150,reg_lambda=350,n_jobs=-1)
clf.fit(X_train_fi,y_train)
predict_y = clf.predict_proba(X_test_fi)
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
predicted_y = np.argmax(predict_y,axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

The test log loss is: 0.32395333663854076



```
In [27]: np.array(X_train_fi.columns)[np.argsort(clf.feature_importances_)[::-1]][0:50]
```

```
Out[27]: array(['word_share', 'freq_qid1', 'token_sort_ratio', 'cwc_max',
                'cwc_min', 'freq_qid2', 'dist_canberra', 'csc_max',
                'longest_substr_ratio', 'fuzz_ratio', 'dist_cityblock',
                'last_word_eq', 'freq_q1-q2', 'token_set_ratio', 'freq_q1+q2',
                'fuzz_partial_ratio', 'csc_min', 'first_word_eq', 'ctc_max',
                'q1len', 'Word_Mover_Dist', 'word_Common', 'abs_len_diff', '105_x',
                '105_y', 'mean_len', '86_x', 'dist_cosine', '17_y', 'ctc_min',
                '166_y', '10_y', 'dist_euclidean', '7_y', '172_x', 'q2_n_words',
                '9_y', '9_x', '25_y', '241_y', '140_y', '111_y', '258_y', '81_y',
                '150_x', '86_y', '51_x', '290_y', '124_x', '45_y'], dtype=object)
```

Trained XGBoost on data dropping avg word vectors with below columns

```
In [44]: X_train_scale.columns
```

```
Out[44]: Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_max',
                'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_len',
                'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
                'fuzz_partial_ratio', 'longest_substr_ratio', 'Word_Mover_Dist',
                'dist_cosine', 'dist_cityblock', 'dist_canberra', 'dist_euclidean',
                'dist_minkowski', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len',
                'q1_n_words', 'q2_n_words', 'word_Common', 'word_Total', 'word_share',
                'freq_q1+q2', 'freq_q1-q2'],
                dtype='object')
```

```
In [29]: estimators = [100,150,200,300,400,600,800]
         test_scores = []
         train_scores = []
         for i in estimators:
             clf = xgb.XGBClassifier(max_depth=3,learning_rate=0.1,n_estimators=i,n_jobs=-1)
             clf.fit(X_train_scale,y_train)
             predict_y = clf.predict_proba(X_train_scale)
             log_loss_train = log_loss(y_train, predict_y, eps=1e-15)
             train_scores.append(log_loss_train)
             predict_y = clf.predict_proba(X_test_scale)
```

```

log_loss_test = log_loss(y_test, predict_y, eps=1e-15)
test_scores.append(log_loss_test)
print('estimators = ',i,'Train Log Loss ',log_loss_train,'Test Log Loss ',log_loss_test)
plt.plot(estimators,train_scores,label='Train Log Loss')
plt.plot(estimators,test_scores,label='Test Log Loss')
plt.xlabel('estimators')
plt.ylabel('Log Loss')

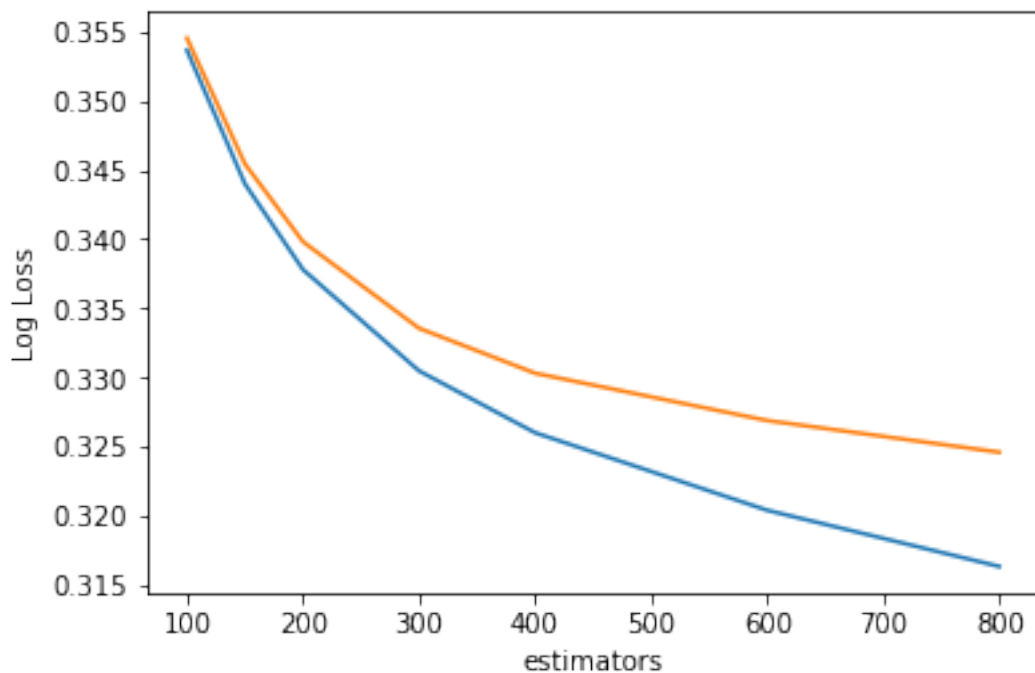
```

```

estimators = 100 Train Log Loss 0.353653999661756 Test Log Loss 0.3545287951428102
estimators = 150 Train Log Loss 0.34398843991044953 Test Log Loss 0.345431883757468
estimators = 200 Train Log Loss 0.3377966667026595 Test Log Loss 0.3398125918620867
estimators = 300 Train Log Loss 0.3304770724113467 Test Log Loss 0.33355668513966974
estimators = 400 Train Log Loss 0.32600683634801697 Test Log Loss 0.3303078436374958
estimators = 600 Train Log Loss 0.3204024156068844 Test Log Loss 0.3268871788204978
estimators = 800 Train Log Loss 0.31634085529657924 Test Log Loss 0.3245975335978598

```

Out[29]: Text(0,0.5,'Log Loss')



```

In [33]: param_dist = {"max_depth": sp_randint(2,5),
                        "learning_rate":uniform(0,0.25),
                        "n_estimators":sp_randint(300,600),
                        "min_child_weight": sp_randint(2, 8),
                        "gamma": uniform(0,4),
                        "subsample":uniform(0.7,0.3),

```

```

        "colsample_bytree": uniform(0.7,0.3),
        "reg_alpha":uniform(100,300),
        "reg_lambda":uniform(100,300)}

model_rs_xgb1 = RandomizedSearchCV(xgb.XGBClassifier(n_jobs=-1,random_state=25), param
                                n_iter=30,scoring='neg_log_loss',cv=5,n_jobs=-1)
model_rs_xgb1.fit(X_train_scale,y_train)
pickle.dump(model_rs_xgb1,open('model_rs_xgb1.p','wb'))

In [36]: dict_scores = []
        idx = 0
        for i in model_rs_xgb1.grid_scores_:
            dict_score = []
            dict_score.append(i[0]['n_estimators'])
            dict_score.append(i[0]['max_depth'])
            dict_score.append(i[0]['subsample'])
            dict_score.append(i[0]['min_child_weight'])
            dict_score.append(i[0]['learning_rate'])
            dict_score.append(i[0]['reg_alpha'])
            dict_score.append(i[0]['reg_lambda'])
            dict_score.append(i[0]['gamma'])
            dict_score.append(i[0]['colsample_bytree'])
            dict_score.append(-i[1])
            dict_score.append((np.abs(i[2]).std()))
            dict_score.append(-model_rs_xgb1.cv_results_['mean_train_score'][idx])
            dict_scores.append(dict_score)
            idx = idx + 1
        scores_df = pd.DataFrame(dict_scores,columns=['n_estimators','depth','subsample','min_
                                                    'learning_rate','reg_alpha','reg_lambda
                                                    'colsample_bytree','Test_score',
                                                    'Test_std','Train_score'])

In [39]: scores_df.sort_values('Test_score').head()

Out[39]:
   n_estimators  depth  subsample  min_child_weight  learning_rate \
10           346     4    0.923334                5      0.208953
28           500     4    0.954113                6      0.131131
8            521     4    0.804484                3      0.153277
6            523     4    0.744275                4      0.109158
22           509     3    0.979516                2      0.078085

   reg_alpha  reg_lambda   gamma  colsample_bytree  Test_score  Test_std \
10  110.805331  316.682194  1.963451      0.826112    0.332645  0.002278
28  119.704012  115.715236  3.768808      0.911753    0.333805  0.002205
8   124.930715  334.800298  2.453220      0.831358    0.334443  0.002472
6   233.725584  266.540244  2.373153      0.828078    0.342799  0.002214
22  190.352679  241.375097  0.154082      0.807970    0.343054  0.002254

```


	Train_score
10	0.327533
28	0.329154
8	0.329945
6	0.340077
22	0.340518

in my view 2nd line (28) is beeter score than forst beacuse of train test scores and test standard deviation

```
In [42]: print('Best score params')
         scores_df.loc[28]
```

Best score params

```
Out[42]: n_estimators      500.000000
         depth             4.000000
         subsample         0.954113
         min_child_weight   6.000000
         learning_rate      0.131131
         reg_alpha          119.704012
         reg_lambda         115.715236
         gamma              3.768808
         colsample_bytree   0.911753
         Test_score         0.333805
         Test_std           0.002205
         Train_score        0.329154
         Name: 28, dtype: float64
```

```
In [40]: import xgboost as xgb
         clf = xgb.XGBClassifier(max_depth=4,learning_rate=0.131131,n_estimators=500,
                                min_child_weight=6,
                                reg_alpha=119.704012,reg_lambda=115.715236,
                                gamma=3.768808,colsample_bytree=0.911753,n_jobs=-1)

         clf.fit(X_train_scale,y_train)
         predict_y = clf.predict_proba(X_test_scale)
         print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
         predicted_y =np.argmax(predict_y,axis=1)
         plot_confusion_matrix(y_test, predicted_y)
```

The test log loss is: 0.33146013829337256

