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 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 quest cause seekers to spend more time finding the best answer to their question, and make writers feel they need to a Quora values canonical questions because they provide a better experience to active seekers and writers, and of 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/AACZdt
- 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-4c1c

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
3. No strict latency concerns.
4. Interpretability is partially important.

```
from google.colab import drive
```

```
from google.colab import drive
drive.mount('/content/drive')
%cd ./drive/My Drive
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

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

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

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

2.1.2 Example Data point

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

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

2.2.2 Performance Metric

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

Metric(s):

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

Reading the data

```
!pip3 install fuzzywuzzy
!pip3 install distance
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")
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 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

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

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

```
Requirement already satisfied: fuzzywuzzy in /usr/local/lib/python3.6/dist-packages (
Requirement already satisfied: distance in /usr/local/lib/python3.6/dist-packages (0.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
data = pd.read_csv("train.csv")
```

```
data=data[0:50000]
```

```
data.head()
```

	id	qid1	qid2	question1	
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step g
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Ind
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be in
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [m
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would

```
data.shape[0],data.shape[1]
```

```
(50000, 6)
```

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 suffic

```
from sklearn.model_selection import train_test_split

df_train,df_test=train_test_split(data,test_size=0.25)
```

```
print(df_train.shape[0])
print(df_test.shape[0])
```

```
37500
12500
```

```
df_train.head()
```

	id	qid1	qid2	question1	
46036	46036	47257	60047	What Quora mean?	W
6903	6903	13510	13511	Why do I do drugs?	
8970	8970	17457	17458	Is there a medication for Aspergers?	Is medi
6285	6285	12321	12322	Is the demand to Shut Down JNU justified?	Is the demand to shut d
47581	47581	84932	84933	Which are the best institutions in India for a...	Which is the best in

```
#Checking whether there are any rows with null values
```

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

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
```

```
#Test
#Checking whether there are any rows with null values
nan_rows = df_test[df_test.isnull().any(1)]
print (nan_rows)
# Filling the null values with ' '
df_test = df_test.fillna('')
nan_rows = df_test[df_test.isnull().any(1)]
print (nan_rows)
```

```
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
Empty DataFrame
Columns: [id, qid1, qid2, question1, question2, is_duplicate]
Index: []
```

3.3 Basic Feature Extraction (before cleaning)

Let us now construct a few features like:

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's
- **q1len** = Length of q1
- **q2len** = Length of q2
- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2
- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)
- **word_share** = (word_common)/(word_Total)
- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2
- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

3.4 Preprocessing of Text

- Preprocessing:
    - Removing html tags
    - Removing Punctuations

- Performing stemming
- Removing Stopwords
- Expanding contractions etc.

```
# 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").repl
                           .replace("n't", " not").replace("what's", "what is").replace("i
                           .replace("'ve", " have").replace("i'm", "i am").replace("'re",
                           .replace("he's", "he is").replace("she's", "she is").replace("'
                           .replace("%", " percent ").replace("₹", " rupee ").replace("$",
                           .replace("€", " euro ").replace("'ll", " will")
    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)


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

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


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


    return x
```

3.5 Advanced Feature Extraction (NLP and Fuzzy Features)

Definition:

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

Features:

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

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

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

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

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

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

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

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

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

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

- **fuzz_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywuzzy-f

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

- **token_sort_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzyw
- **token_set_ratio** : https://github.com/seatgeek/fuzzywuzzy#usage http://chairnerd.seatgeek.com/fuzzywu
- **longest_substr_ratio** : Ratio of length longest common substring to min lenghth of token count of Q1 and longest_substr_ratio = len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

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

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

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

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

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

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

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


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

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

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

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

    #Average Token Length of both Questions
    token_features[9] = (len(q1_tokens) + len(q2_tokens))/2
    return token_features
```

```
# get the Longest Common sub string

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

def extract_features(df):
    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'])



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

    print("token features...")

    # Merging Features with dataset

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

    df["cwc_min"]       = list(map(lambda x: x[0], token_features))
    df["cwc_max"]       = list(map(lambda x: x[1], token_features))
    df["csc_min"]       = list(map(lambda x: x[2], token_features))
    df["csc_max"]       = list(map(lambda x: x[3], token_features))
    df["ctc_min"]       = list(map(lambda x: x[4], token_features))
```

```
        df["ctc_min"]       = list(map(lambda x: x[4], token_features))
        df["ctc_max"]       = list(map(lambda x: x[5], token_features))
        df["last_word_eq"]  = list(map(lambda x: x[6], token_features))
        df["first_word_eq"] = list(map(lambda x: x[7], token_features))
        df["abs_len_diff"]  = list(map(lambda x: x[8], token_features))
        df["mean_len"]      = list(map(lambda x: x[9], token_features))

        #Computing Fuzzy Features and Merging with Dataset

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

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


df_train_afe = extract_features(df_train)
df_test_afe=extract_features(df_test)
```

```
token features...
fuzzy features..
token features...
fuzzy features..
```

```
df_train_afe.head()
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2
46036	46036	47257	60047	what quora mean	why is quora called quora	0	1	2
6903	6903	13510	13511	why do i do drugs	why do drugs exist	0	1	1
8970	8970	17457	17458	is there a medication for aspergers	is medication good for aspergers	0	1	1
6285	6285	12321	12322	is the demand to shut down jnu justified	is the demand to shut down jnu by a section of...	1	1	1
47581	47581	84932	84933	which are the best institutions in india for a...	which is the best institute in india for compu...	1	1	1

3.6 Featurizing text data with tfidf word-vectors

```
df_train['question1'] = df_train['question1'].apply(lambda x: str(x))
df_train['question2'] = df_train['question2'].apply(lambda x: str(x))
df_test['question1'] = df_test['question1'].apply(lambda x: str(x))
df_test['question2'] = df_test['question2'].apply(lambda x: str(x))


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions_train = list(df_train['question1']+df_train['question2'])
questions_test = list(df_test['question1']+df_test['question2'])

tfidf = TfidfVectorizer(lowercase=False,max_df=100,min_df=10 )
tfidf.fit(questions_train)
tfidf_train=tfidf.transform(questions_train)
tfidf_test=tfidf.transform(questions_test)


print(tfidf_train.shape)
print(tfidf_test.shape)
print(type(tfidf_train))
```

```
(37500, 4040)
(12500, 4040)
<class 'scipy.sparse.csr.csr_matrix'>
```

```
#https://stackoverflow.com/questions/44811405/pandas-concatenating-dataframe-with-sparse-m

tfidf_train= pd.SparseDataFrame(tfidf_train,columns=tfidf.get_feature_names(),default_fill
tfidf_test=pd.SparseDataFrame(tfidf_test,columns=tfidf.get_feature_names(),default_fill_va
```

```
print(tfidf_train.shape)
print(df_train_afe.shape)
print(type(tfidf_train))
```

```
(37500, 4040)
(37500, 32)
<class 'pandas.core.sparse.frame.SparseDataFrame'>
```

```
df_train_afe=df_train_afe.reset_index(drop=True)
df_test_afe=df_test_afe.reset_index(drop=True)
```

```
df_train_afe.head()
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1l
0	46036	47257	60047	what quora mean	why is quora called quora	0	1	2	
1	6903	13510	13511	why do i do drugs	why do drugs exist	0	1	1	
2	8970	17457	17458	is there a medication for aspergers	is medication good for aspergers	0	1	1	
3	6285	12321	12322	is the demand to shut down jnu justified	is the demand to shut down jnu by a section of...	1	1	1	
4	47581	84932	84933	which are the best institutions in india for a...	which is the best institute in india for compu...	1	1	1	

```
X_train=pd.concat([df_train_afe,tfidf_train],axis=1)
X_test=pd.concat([df_test_afe,tfidf_test],axis=1)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(37500, 4072)
(12500, 4072)
```

```
y_train= X_train['is_duplicate']
y_test=X_test['is_duplicate']
```

```
print(y_train.shape)
```

```
(37500,)
```

```
X_train.head()
```

	id	qid1	qid2	question1	question2	is_duplicate	freq_qid1	freq_qid2	q1l
0	46036	47257	60047	what quora mean	why is quora called quora	0	1	2	
1	6903	13510	13511	why do i do drugs	why do drugs exist	0	1	1	
2	8970	17457	17458	is there a medication for aspergers	is medication good for aspergers	0	1	1	
3	6285	12321	12322	is the demand to shut down jnu justified	is the demand to shut down jnu by a section of...	1	1	1	
4	47581	84932	84933	which are the best institutions in india for a...	which is the best institute in india for compu...	1	1	1	

5 rows × 4072 columns

```
X_train.drop([ 'id','qid1','qid2','question1','question2','is_duplicate'], axis=1, inplace
X_test.drop([ 'id','qid1','qid2','question1','question2','is_duplicate'], axis=1, inplace=
```

```
X_train.head()
```

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Tota
0	1	2	16	26	3	5	1.0	8.
1	1	1	18	19	5	4	2.0	8.
2	1	1	36	33	6	5	4.0	11.
3	1	1	41	128	8	24	8.0	30.
4	1	1	94	64	18	12	8.0	29.

5 rows × 4065 columns

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

```
(37500, 4065)
(37500,)
(12500, 4065)
(12500,)
```

```
#Standardize the data

from sklearn import preprocessing
# Get column names first
names = X_train.columns
# Create the Scaler object
scaler = preprocessing.StandardScaler()
# Fit your data on the scaler object
scaled_df = scaler.fit_transform(X_train)
X_train = pd.DataFrame(scaled_df, columns=names)
```

```
#Test
# Get column names first
names = X_test.columns
# Create the Scaler object
scaler = preprocessing.StandardScaler()
# Fit your data on the scaler object
scaled_df = scaler.fit_transform(X_test)
X_test = pd.DataFrame(scaled_df, columns=names)
```

```
X_train
```

	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common
0	-0.288846	1.257059	-1.450212	-0.997005	-1.457146	-0.968724	-1.124734
1	-0.288846	-0.285673	-1.383615	-1.202181	-1.090198	-1.125882	-0.803337
2	-0.288846	-0.285673	-0.784241	-0.791828	-0.906724	-0.968724	-0.160544
3	-0.288846	-0.285673	-0.617749	1.992711	-0.539776	2.017283	1.125043
4	-0.288846	-0.285673	1.147073	0.116811	1.294965	0.131384	1.125043
5	-0.288846	-0.285673	-0.917435	-0.176299	-0.723250	-0.182932	-1.124734
6	-0.288846	-0.285673	-0.884137	-0.352164	-1.090198	-0.497249	-1.124734
7	-0.288846	-0.285673	-1.083928	-0.733206	-1.090198	-0.654407	-0.803337
8	-0.288846	-0.285673	-0.351361	-0.381475	-0.356302	-0.497249	0.482249
9	-0.288846	-0.285673	0.281311	0.175433	-0.356302	-0.340090	0.482249
10	-0.288846	-0.285673	-0.984032	-0.791828	-0.723250	-0.654407	0.160853
11	-0.288846	-0.285673	-0.784241	-0.322853	-0.723250	-0.025774	-1.124734
12	1.394143	-0.285673	0.181416	-0.615963	0.010646	-0.654407	0.160853
13	-0.288846	-0.285673	-0.451256	-0.557341	-0.356302	-0.497249	0.803646
14	-0.288846	-0.285673	0.048222	-0.117677	0.194120	-0.025774	1.767836
15	-0.288846	-0.285673	1.646550	0.204744	0.377594	-0.497249	-0.803337
16	1.394143	1.257059	-0.517853	0.028878	-0.172828	0.131384	0.482249
17	-0.288846	-0.285673	1.147073	0.527164	1.294965	0.445701	1.125043
18	-0.288846	-0.285673	-0.251465	-0.117677	-0.723250	-0.654407	-0.481941
19	-0.288846	-0.285673	0.481102	0.175433	0.928017	0.602859	1.125043
20	-0.288846	-0.285673	2.945192	-0.381475	3.863601	-0.497249	-1.446131
21	-0.288846	-0.285673	1.047177	0.908206	0.744543	0.602859	3.053423
22	-0.288846	-0.285673	2.945192	-0.528030	3.313179	-0.497249	-1.124734
23	-0.288846	-0.285673	0.847386	0.321987	0.561069	0.288543	1.125043
24	-0.288846	-0.285673	-0.917435	0.527164	-0.723250	0.602859	0.482249
25	-0.288846	-0.285673	-0.517853	-0.059055	-0.172828	-0.182932	-0.160544
26	-0.288846	-0.285673	-0.817540	-0.615963	-0.906724	-0.497249	-1.124734
27	-0.288846	-0.285673	1.180371	0.439231	0.928017	0.602859	-1.446131
28	-0.288846	-0.285673	0.614296	-0.059055	0.194120	-0.340090	-1.124734
29	-0.288846	-0.285673	0.414505	-0.293542	-0.356302	-0.497249	-1.446131
...	...	...	...	...	...	...	...
37470	9.809089	1.257059	-0.484555	-0.879761	-0.539776	-0.811565	-0.160544

37471	1.394143	1.257059	-1.083928	-0.879761	-1.090198	-0.811565	-0.160544
37472	-0.288846	-0.285673	1.213670	1.054761	1.111491	0.602859	1.125043
37473	-0.288846	12.056183	-0.318062	-0.879761	0.010646	-0.811565	-1.124734
37474	-0.288846	-0.285673	0.514401	2.227198	0.928017	2.645916	0.482249
37475	-0.288846	-0.285673	-0.218167	-0.586652	0.010646	-0.340090	0.803646
37476	-0.288846	-0.285673	3.011789	0.439231	3.313179	0.760017	-1.124734
37477	-0.288846	-0.285673	0.913983	0.878895	1.111491	1.074334	0.803646
37478	-0.288846	-0.285673	-0.651047	-0.410786	-0.539776	-0.340090	-0.481941
37479	-0.288846	-0.285673	-0.584450	1.816845	-0.539776	1.702967	-0.803337
37480	-0.288846	-0.285673	-0.218167	-0.234920	0.010646	-0.025774	1.446439
37481	1.394143	-0.285673	-0.318062	0.234054	-0.356302	0.131384	-0.160544
37482	-0.288846	-0.285673	-0.451256	-0.117677	-0.172828	0.288543	0.482249
37483	-0.288846	-0.285673	1.446759	0.116811	0.928017	-0.025774	0.160853
37484	1.394143	1.257059	-0.750943	-0.498719	-0.906724	-0.497249	-0.160544
37485	-0.288846	-0.285673	-0.251465	-0.176299	-0.172828	-0.497249	0.482249
37486	-0.288846	-0.285673	0.780789	0.175433	0.377594	-0.025774	0.803646
37487	-0.288846	-0.285673	1.679849	-0.000433	1.845387	0.131384	-0.481941
37488	-0.288846	-0.285673	-0.750943	-0.381475	-0.356302	-0.182932	0.160853
37489	-0.288846	-0.285673	-0.118271	-0.234920	0.194120	0.131384	-0.160544
37490	-0.288846	-0.285673	0.447804	-0.264231	0.010646	-0.025774	0.160853
37491	-0.288846	-0.285673	-0.417958	-0.469408	-0.356302	-0.182932	-0.803337
37492	-0.288846	-0.285673	-0.817540	-0.879761	-0.539776	-0.811565	-0.160544
37493	-0.288846	-0.285673	-0.484555	1.611668	-0.539776	1.231492	0.160853
37494	1.394143	-0.285673	0.048222	-0.352164	-0.172828	0.131384	-0.160544
37495	-0.288846	-0.285673	0.580998	-0.029744	1.111491	-0.025774	-1.124734
37496	-0.288846	-0.285673	-0.850838	0.058189	-0.723250	0.602859	-0.803337
37497	-0.288846	-0.285673	-1.250421	-1.026316	-1.273672	-0.968724	-0.160544
37498	-0.288846	-0.285673	-0.850838	-0.469408	-1.090198	-0.654407	-0.160544
37499	-0.288846	-0.285673	-0.318062	-0.352164	-0.172828	-0.182932	-0.160544

37500 rows × 4065 columns

```
# 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
```

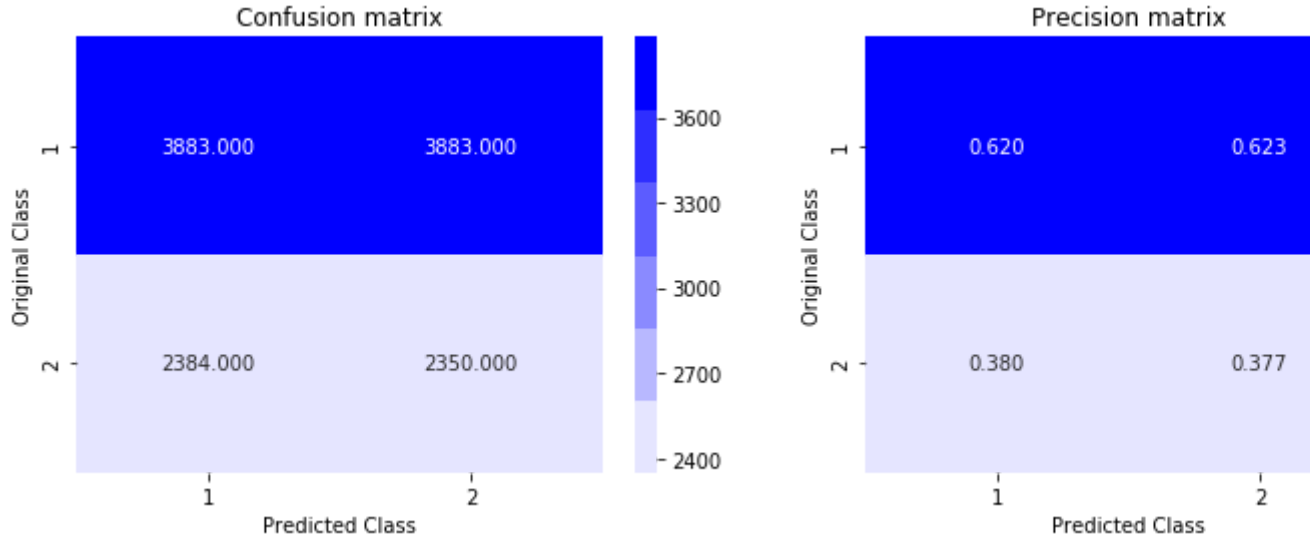
```
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted

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

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

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

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

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

    plt.show()


train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len
```

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

```
Class 0:  0.6289333333333333 Class 1:  0.37106666666666666
---------- Distribution of output variable in train data ----------
Class 0:  0.37872 Class 1:  0.37872
```

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
# we create a output array that has exactly same size as the CV data
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-15))

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

```
Log loss on Test Data using Random Model 0.8866800891022211
```

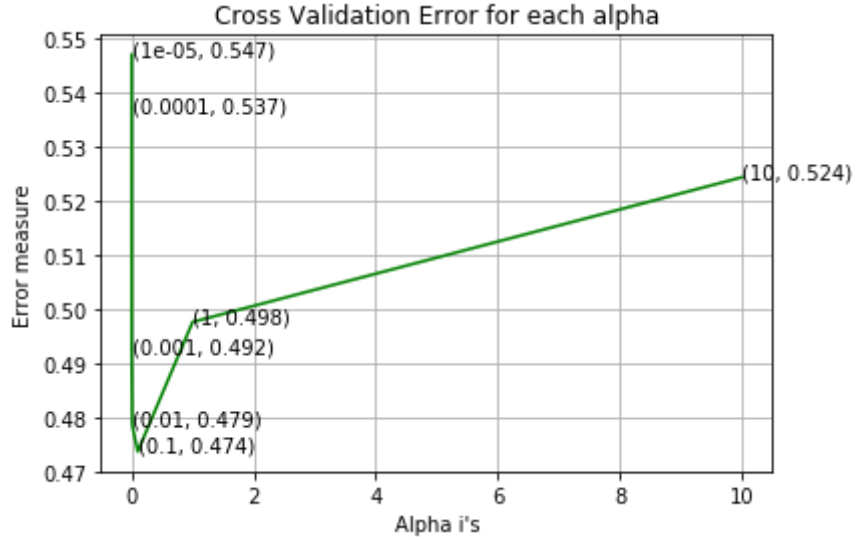


Logistic Regression with hyperparameter tuning

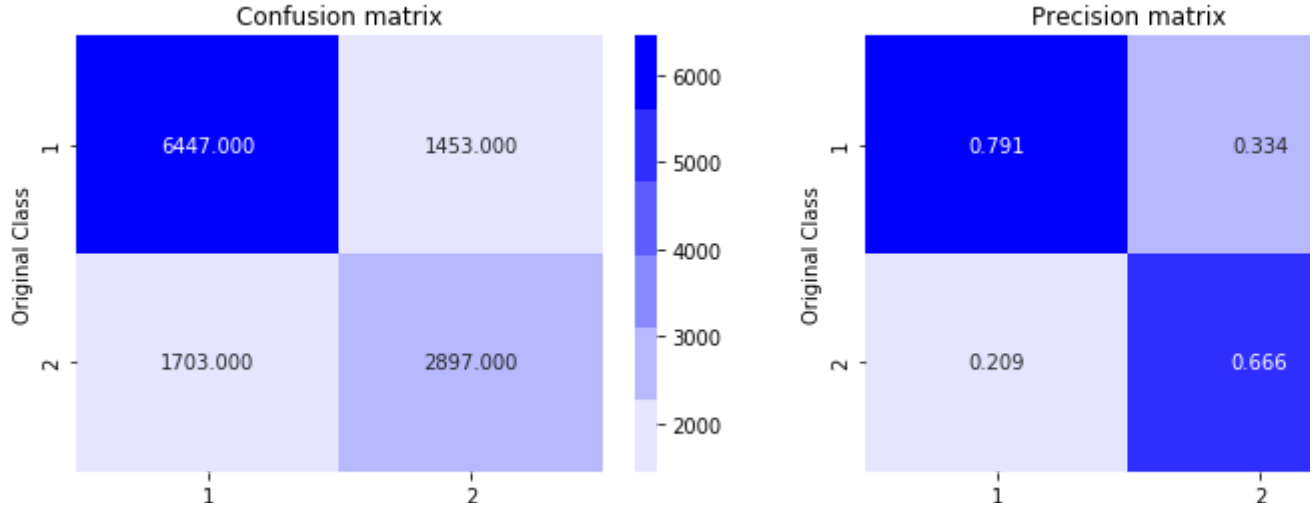
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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',class_weight ='balanced', random
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
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: 0.5470398012003502
For values of alpha =  0.0001 The log loss is: 0.5366359128719703
For values of alpha =  0.001 The log loss is: 0.4920774583921008
For values of alpha =  0.01 The log loss is: 0.47858478044794817
For values of alpha =  0.1 The log loss is: 0.4737503749188069
For values of alpha =  1 The log loss is: 0.4976778454103811
For values of alpha =  10 The log loss is: 0.5243890806706727
```




```
For values of best alpha =  0.1 The train log loss is: 0.4079769021538423
For values of best alpha =  0.1 The test log loss is: 0.47412357579388975
Total number of data points : 12500
```

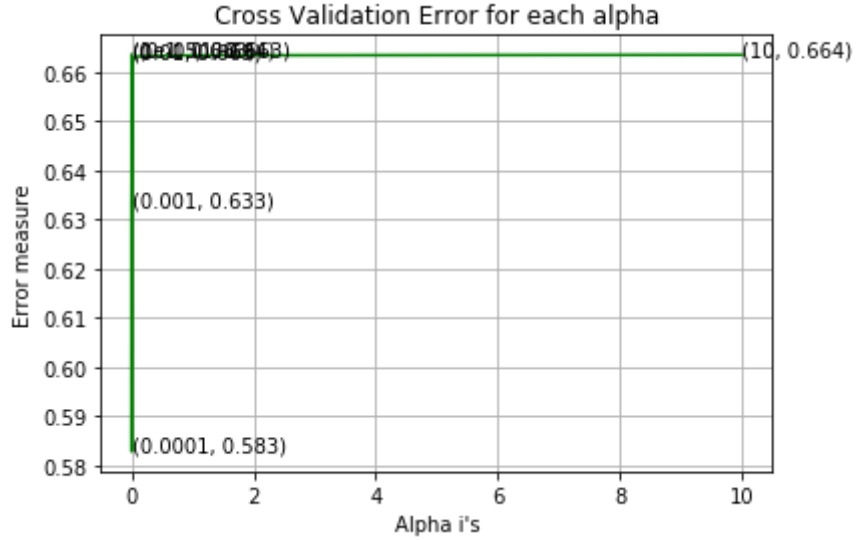


Linear SVM with hyperparameter tuning

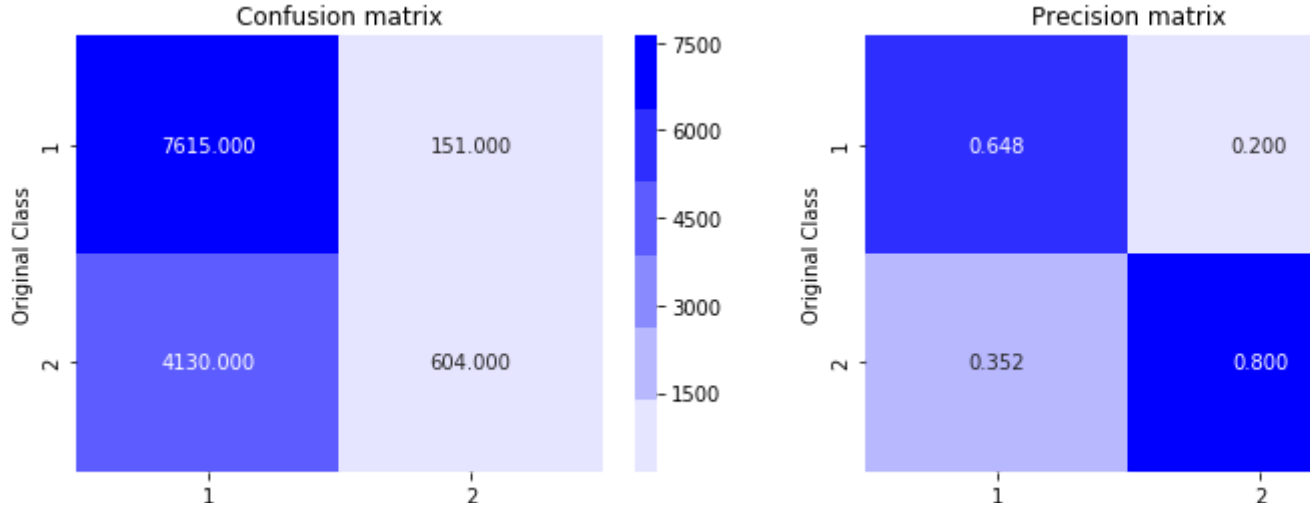
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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',class_weight ='balanced', rand
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
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: 0.6635584953279854
For values of alpha =  0.0001 The log loss is: 0.5827881038270943
For values of alpha =  0.001 The log loss is: 0.632599457384758
For values of alpha =  0.01 The log loss is: 0.6628558935427534
For values of alpha =  0.1 The log loss is: 0.6632241388843334
For values of alpha =  1 The log loss is: 0.6634265702862369
For values of alpha =  10 The log loss is: 0.6635377700823614
```




```
For values of best alpha =  0.0001 The train log loss is: 0.5502826246647671
For values of best alpha =  0.0001 The test log loss is: 0.586131654259812
Total number of data points : 12500
```



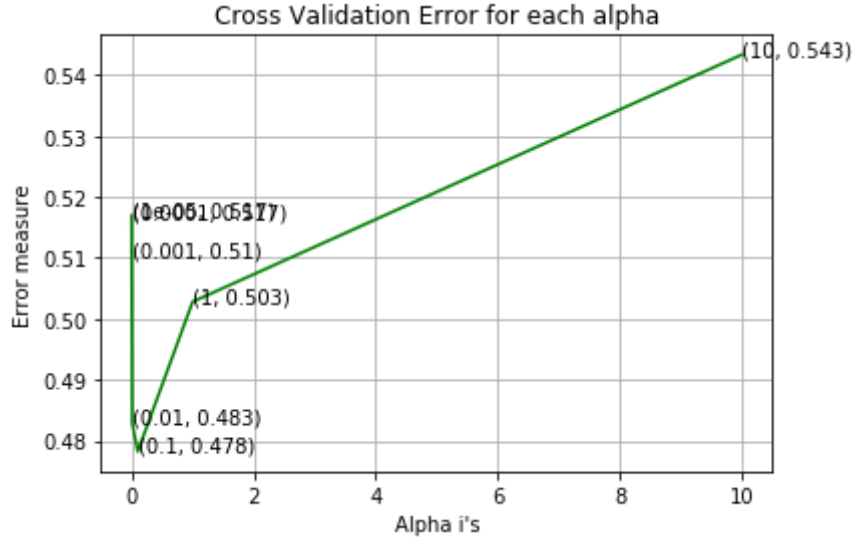
1.Test log loss for logistic regression is 0.47

2.Test log loss for linear svm is 0.58

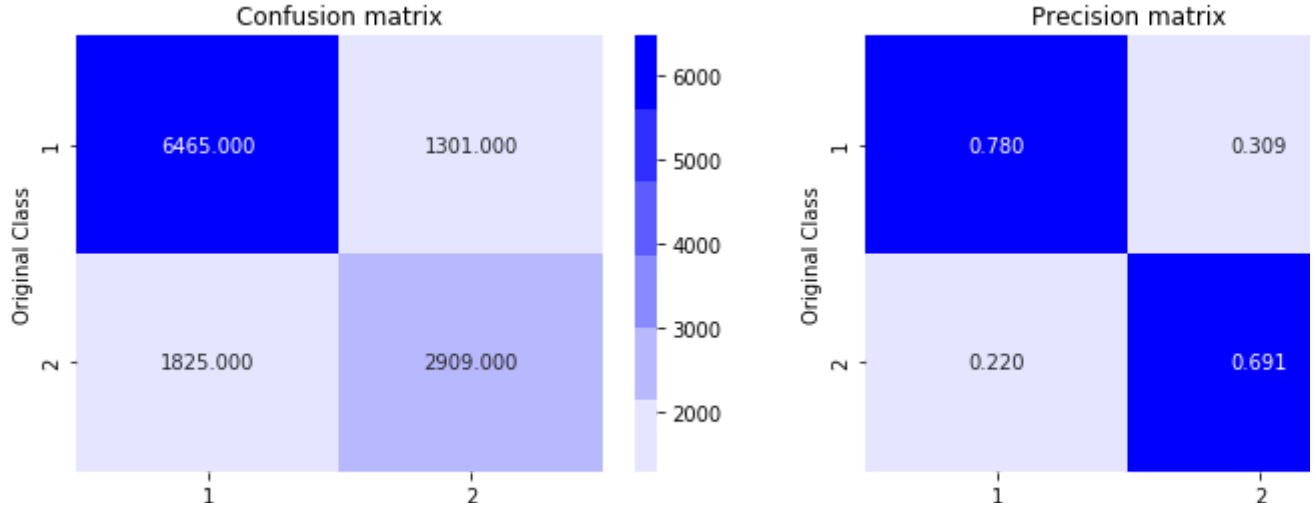
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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',class_weight ='balanced',learnin
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
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: 0.5169469999888603
For values of alpha =  0.0001 The log loss is: 0.5165304885581296
For values of alpha =  0.001 The log loss is: 0.5102183485626823
For values of alpha =  0.01 The log loss is: 0.48282411551124765
For values of alpha =  0.1 The log loss is: 0.4782271029067348
For values of alpha =  1 The log loss is: 0.5027972765374535
For values of alpha =  10 The log loss is: 0.5433434921423149
```



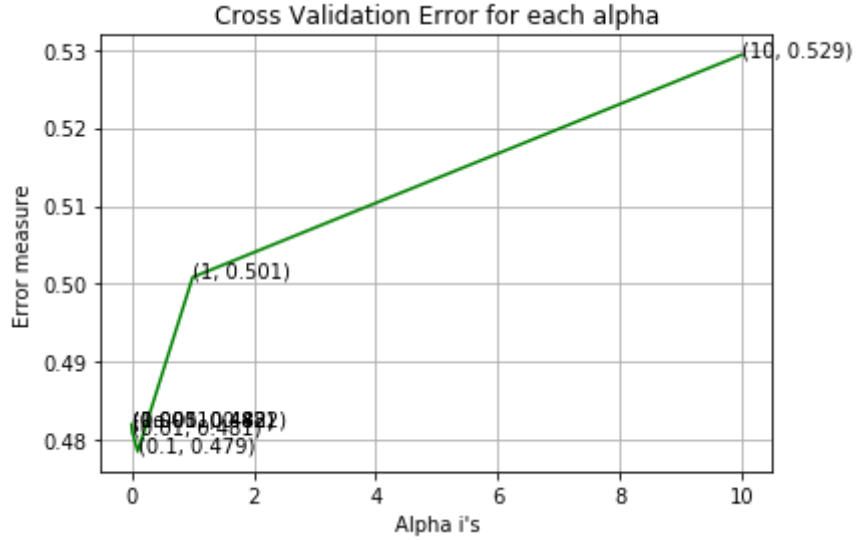

```
For values of best alpha =  0.1 The train log loss is: 0.4049801819606242
For values of best alpha =  0.1 The test log loss is: 0.4771584353256933
Total number of data points : 12500
```



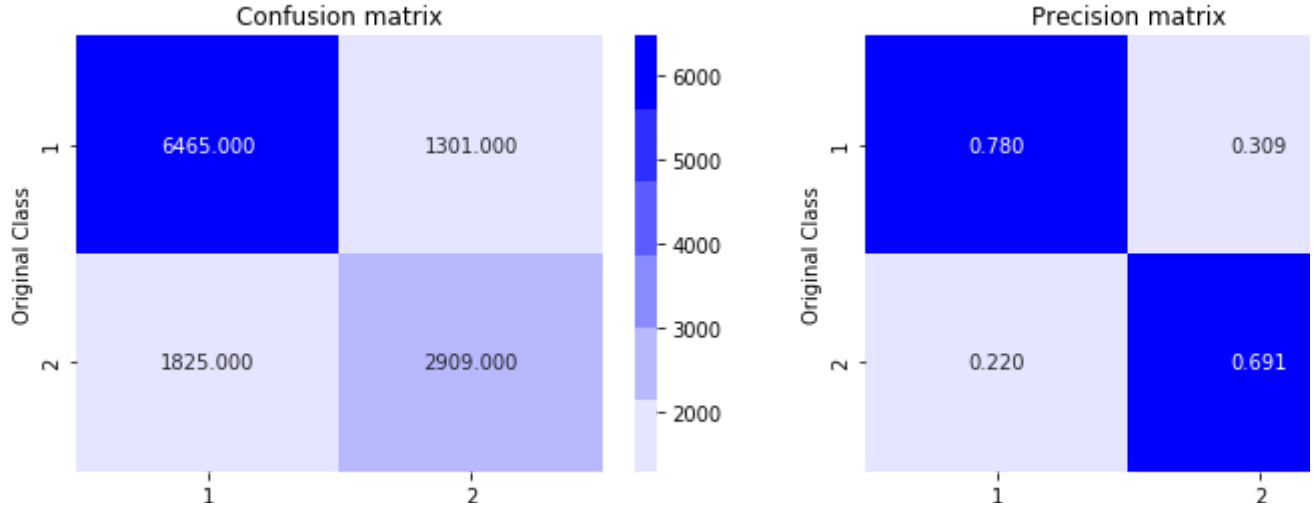

```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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',class_weight ='balanced',learnin
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
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: 0.48201274305562763
For values of alpha =  0.0001 The log loss is: 0.4820030472790877
For values of alpha =  0.001 The log loss is: 0.4819069439049096
For values of alpha =  0.01 The log loss is: 0.48103479597585597
For values of alpha =  0.1 The log loss is: 0.4785356846390298
For values of alpha =  1 The log loss is: 0.5008524896984311
For values of alpha =  10 The log loss is: 0.5293501157987367
```



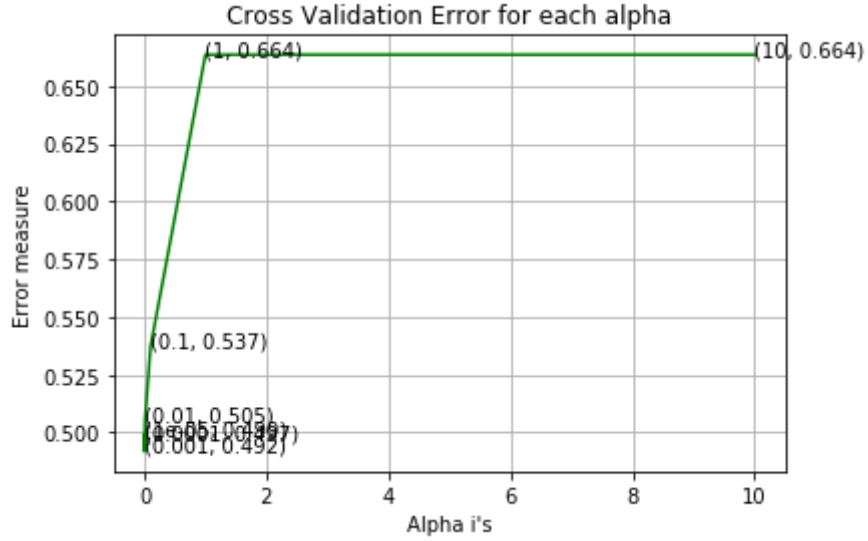

```
For values of best alpha =  0.1 The train log loss is: 0.4049801819606242
For values of best alpha =  0.1 The test log loss is: 0.4771584353256933
Total number of data points : 12500
```



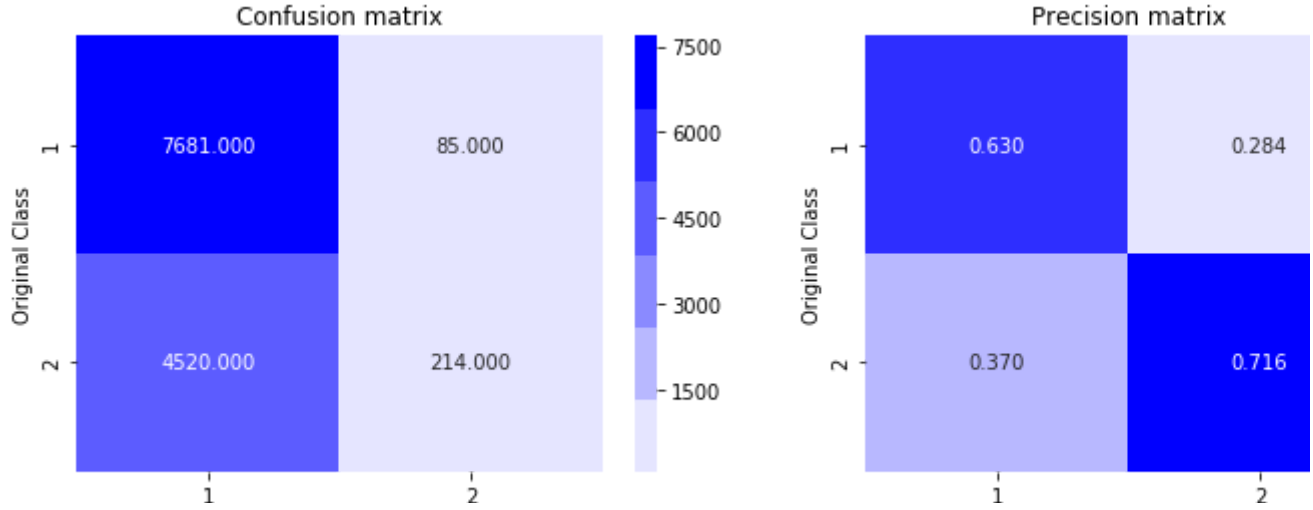

```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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',class_weight ='balanced',learn
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

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


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

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
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: 0.4986730612266795
For values of alpha =  0.0001 The log loss is: 0.49718277349161744
For values of alpha =  0.001 The log loss is: 0.4917639918226238
For values of alpha =  0.01 The log loss is: 0.5051952690890071
For values of alpha =  0.1 The log loss is: 0.537129727032264
For values of alpha =  1 The log loss is: 0.6635591703549996
For values of alpha =  10 The log loss is: 0.6635591703549993
```




```
For values of best alpha =  0.001 The train log loss is: 0.5995021748632015
For values of best alpha =  0.001 The test log loss is: 0.6170247730392437
Total number of data points : 12500
```




```
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/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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',class_weight ='balanced',learnin
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe

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


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

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