pcd

July 21, 2020

Personalized cancer diagnosis

1. Business Problem

1.1. Description
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)
Download training_variants.zip and training_text.zip from Kaggle.
Context:
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462
Problem statement :
Classify the given genetic variations/mutations based on evidence from text-based clinical literature.
1.2. Source/Useful Links
Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.

- Both these data files are have a common column called ID
- Data file's information:

```
<li>
training_variants (ID , Gene, Variations, Class)
</li>
<li>
training_text (ID, Text)
</li>
```

2.1.2. Example Data Point
training_variants
ID,Gene,Variation,Class 0,FAM58A,Truncating Mutations,1 1,CBL,W802*,2 2,CBL,Q249E,2 ...
training_text
ID,Text 0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem
2.2.1. Type of Machine Learning Problem

```
There are nine different classes a genetic mutation can be classified into => Multi cla
```

2.2.2. Performance Metric
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation
Metric(s): * Multi class log-loss * Confusion matrix
2.2.3. Machine Learing Objectives and Constraints
Objective: Predict the probability of each data-point belonging to each of the nine classes.
Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets
Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id
=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redire
ct_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20http
s%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.reado
nly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

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

[4]:
```
import os
os.chdir("/content/drive/My Drive/PCD")
!ls -l
```

```
total 207220
-rw------- 1 root root 212125752 Jun 20  2018 training_text
-rw------- 1 root root     66688 Jun 23  2017 training_variants
```

3. Exploratory Data Analysis

[60]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import nltk
nltk.download("stopwords")
import warnings
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 sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

[6]:
```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

```
Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

[6]:
	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

training/training_variants is a comma separated file containing the description of the genetic
Fields are

    ID : the id of the row used to link the mutation to the clinical evidence
    Gene : the gene where this genetic mutation is located
    Variation : the aminoacid change for this mutations
    Class : 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

[7]:
```
# note the seprator in this file
data_text =pd.
 ↪read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

```
Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']
```

[7]:
	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3. Preprocessing of text

[8]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
```

```
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then retain that word from the data
            if not word in stop_words:
                string += word + " "

        data_text[column][index] = string
```

[9]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time,␣
 ↪"seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 29.143758000000002 seconds
```

[10]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

[10]:

	ID	Gene	...	Class	TEXT
0	0	FAM58A	...	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	...	2	abstract background non small cell lung cancer...
2	2	CBL	...	2	abstract background non small cell lung cancer...
3	3	CBL	...	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	...	4	oncogenic mutations monomeric casitas b lineag...

[5 rows x 5 columns]

[11]:
```
result[result.isnull().any(axis=1)]
```

[11]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

[12]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +'␣
↪'+result['Variation']
```

[13]:
```
result[result['ID']==1109]
```

[13]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

[14]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of output␣
↪varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true,␣
↪stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same␣
↪distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train,␣
↪stratify=y_train, test_size=0.2)
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

[15]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

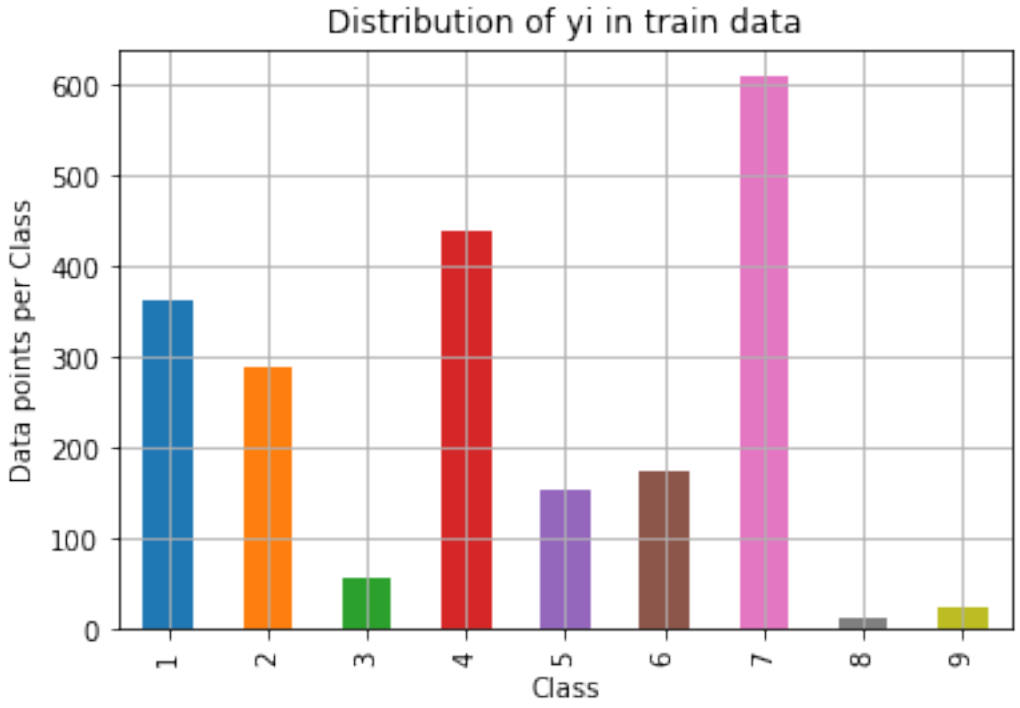
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

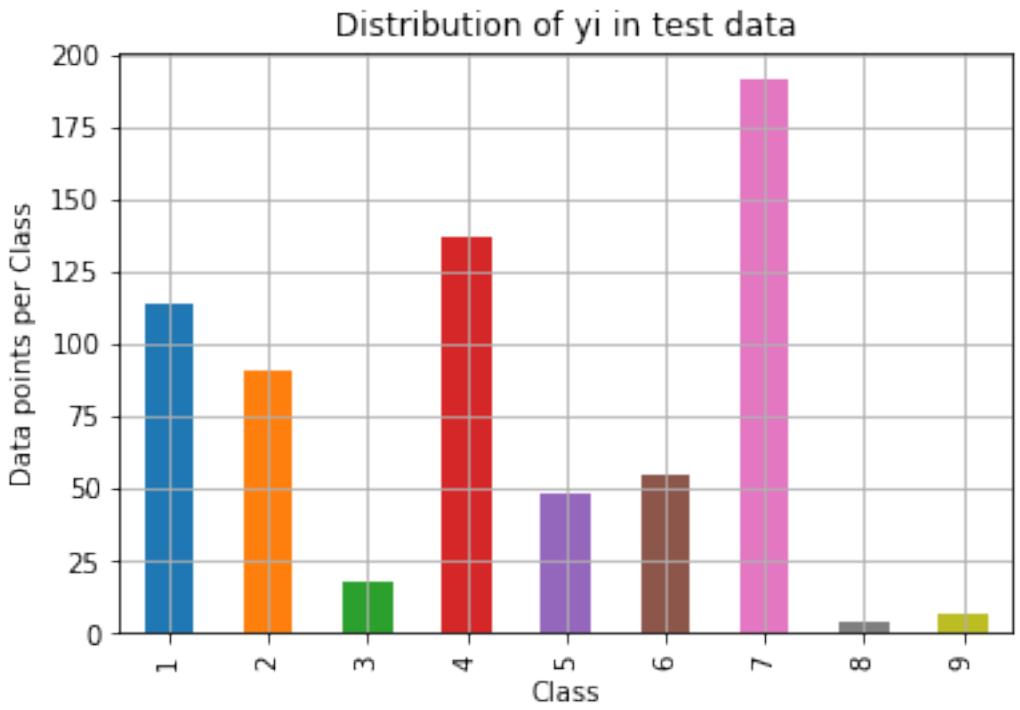
[ ]:
```
# it returns a dict, keys as class labels and values as the number of data␣
↪points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
```

```
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.
 ↪argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing␣
 ↪order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.
 ↪values[i], '(', np.round((train_class_distribution.values[i]/train_df.
 ↪shape[0]*100), 3), '%)')


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.
 ↪argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing␣
 ↪order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.
 ↪values[i], '(', np.round((test_class_distribution.values[i]/test_df.
 ↪shape[0]*100), 3), '%)')

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.
 ↪argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing␣
 ↪order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
```

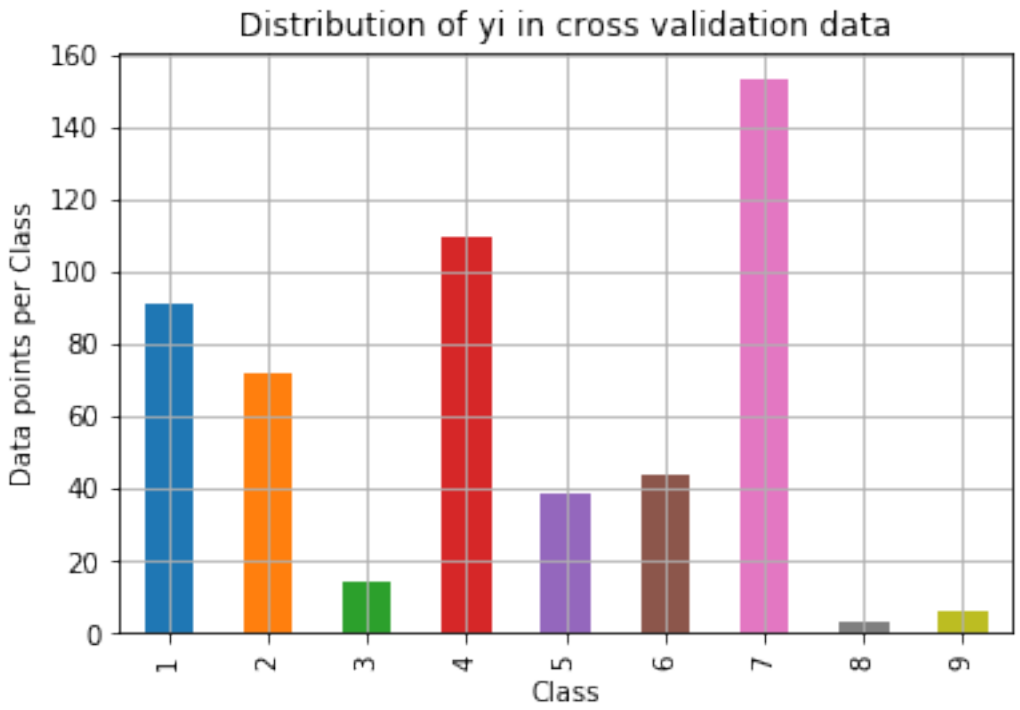
```
    print('Number of data points in class', i+1, ':',cv_class_distribution.
↪values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.
↪shape[0]*100), 3), '%)')
```




```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
------------------------------------------------------------------------
```




```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

[115]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i␣
 ↪are predicted class j

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

    # C = [[1, 2],
```

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
```

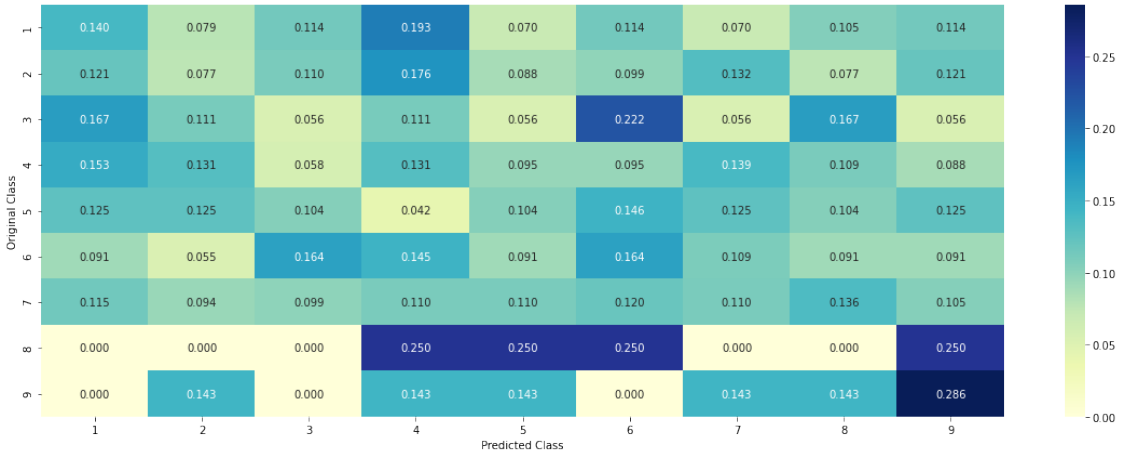
```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

[ ]:
```
# 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
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random␣
 ↪Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random␣
 ↪Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

```
Log loss on Cross Validation Data using Random Model 2.495391718230231
Log loss on Test Data using Random Model 2.5174687957723947
-------------------- Confusion matrix --------------------
```

---------- Precision matrix (Columm Sum=1) ----------

---------- Recall matrix (Row sum=1) ----------



3.3 Univariate Analysis

[61]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in␣
 ↪train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in␣
 ↪class1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9)␣
 ↪representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
```

```
    # Q61L                                          3
    # S222D                                         2
    # P130S                                         2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for␣
↪each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature␣
↪occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs␣
↪to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) &␣
↪(train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) &␣
↪(train_df[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that␣
↪particular feature occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
```

```
    #     {'BRCA1': [0.200757575757575, 0.037878787878788, 0.
↪06818181818181777, 0.13636363636363635, 0.25, 0.1931818181818181, 0.
↪037878787878788, 0.037878787878788, 0.037878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.
↪061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.
↪06632653061224902, 0.05102040816326307, 0.05102040816326307, 0.
↪056122448979591837],
    #      'EGFR': [0.05681818181818181, 0.21590909090909091, 0.0625, 0.
↪06818181818181777, 0.06818181818181777, 0.0625, 0.34659090909090912, 0.
↪0625, 0.05681818181818181],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.
↪060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.
↪34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.
↪060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.
↪069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.
↪062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.
↪062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.
↪072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.
↪066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.
↪066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.
↪073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.
↪080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.
↪066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each␣
↪feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is␣
↪there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace

smoothing

(numerator + 10*alpha) / (denominator + 90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

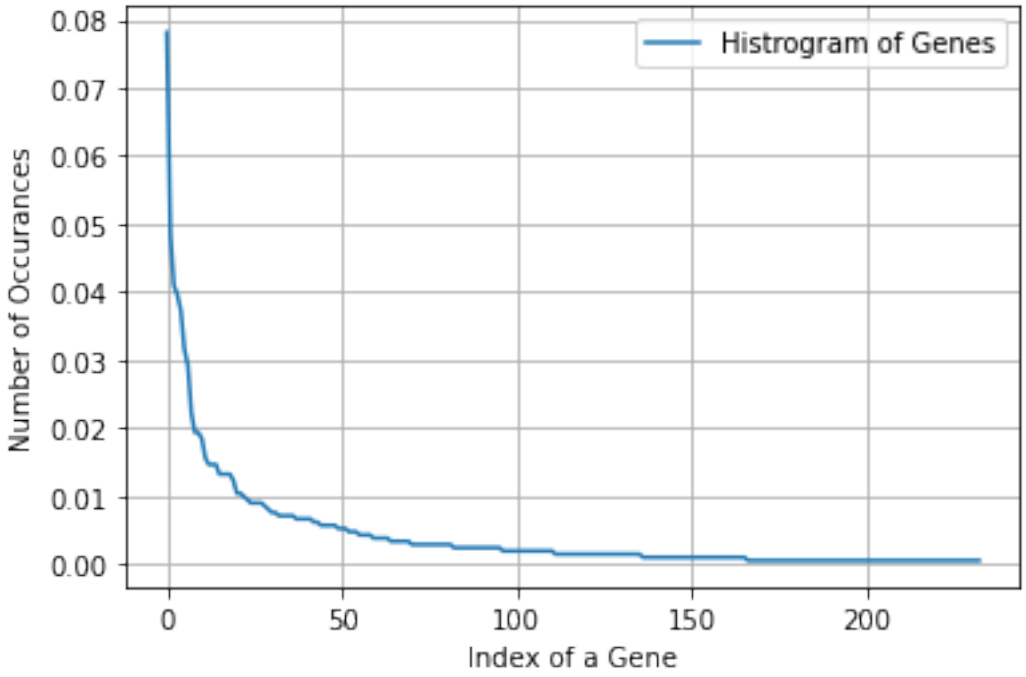
[62]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 233
BRCA1     166
TP53      103
EGFR       87
BRCA2      84
PTEN       79
KIT        67
BRAF       62
ALK        47
ERBB2      41
PDGFRA     41
Name: Gene, dtype: int64
```

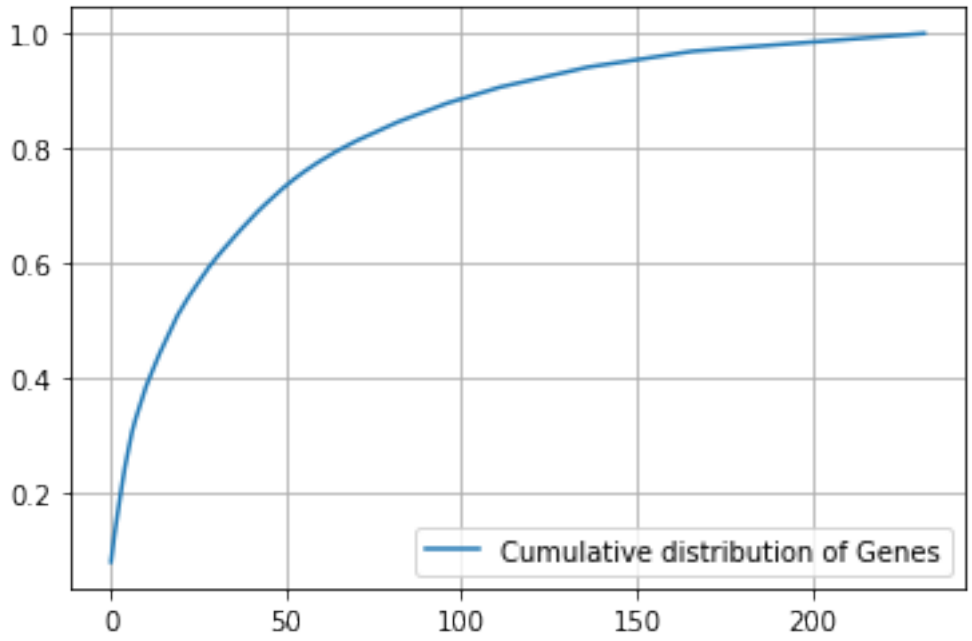
[63]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes␣
 ↪in the train data, and they are distibuted as follows",)
```

```
Ans: There are 233 different categories of genes in the train data, and they are
distibuted as follows
```

[64]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



[65]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One hot Encoding

Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

[66]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",␣
 ↪train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",␣
 ↪test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

[ ]:
```
print("train_gene_feature_responseCoding is converted feature using respone␣
 ↪coding method. The shape of gene feature:",␣
 ↪train_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding
method. The shape of gene feature: (2124, 9)
```

[67]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(max_features=1000)
train_gene_feature_onehotCoding = gene_vectorizer.
 ↪fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

[ ]:
```
train_df['Gene'].head()
```

[ ]:
```
2446     BRCA1
1989    MAP2K1
1347      AKT1
265       EGFR
2780     BRCA2
Name: Gene, dtype: object
```

[ ]:
```
gene_vectorizer.get_feature_names()
```

```
[ ]: ['abl1',
      'acvr1',
      'ago2',
      'akt1',
      'akt2',
      'akt3',
      'alk',
      'apc',
      'ar',
      'araf',
      'arid1a',
      'arid1b',
      'arid2',
      'arid5b',
      'asxl2',
      'atm',
      'atrx',
      'aurka',
      'axl',
      'b2m',
      'bap1',
      'bard1',
      'bcl10',
      'bcl2',
      'bcl2l11',
      'bcor',
      'braf',
      'brca1',
      'brca2',
      'brd4',
      'brip1',
      'btk',
      'card11',
      'carm1',
      'casp8',
      'cbl',
      'ccnd1',
      'ccnd3',
      'ccne1',
      'cdh1',
      'cdk12',
      'cdk4',
      'cdk6',
      'cdkn1a',
      'cdkn1b',
      'cdkn2a',
      'cdkn2b',
```

```
'cdkn2c',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'fubp1',
```

```
'gata3',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'jak1',
'jak2',
'jun',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
```

```
'myd88',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rac1',
'rad21',
'rad50',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
```

```
'rheb',
'rhoa',
'rictor',
'rit1',
'ros1',
'rras2',
'runx1',
'rybp',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stat3',
'stk11',
'tcf3',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vhl',
'whsc1',
'xpo1',
'xrcc2',
'yap1']
```

[ ]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot␣
 ↪encoding method. The shape of gene feature:",␣
 ↪train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 232)
```

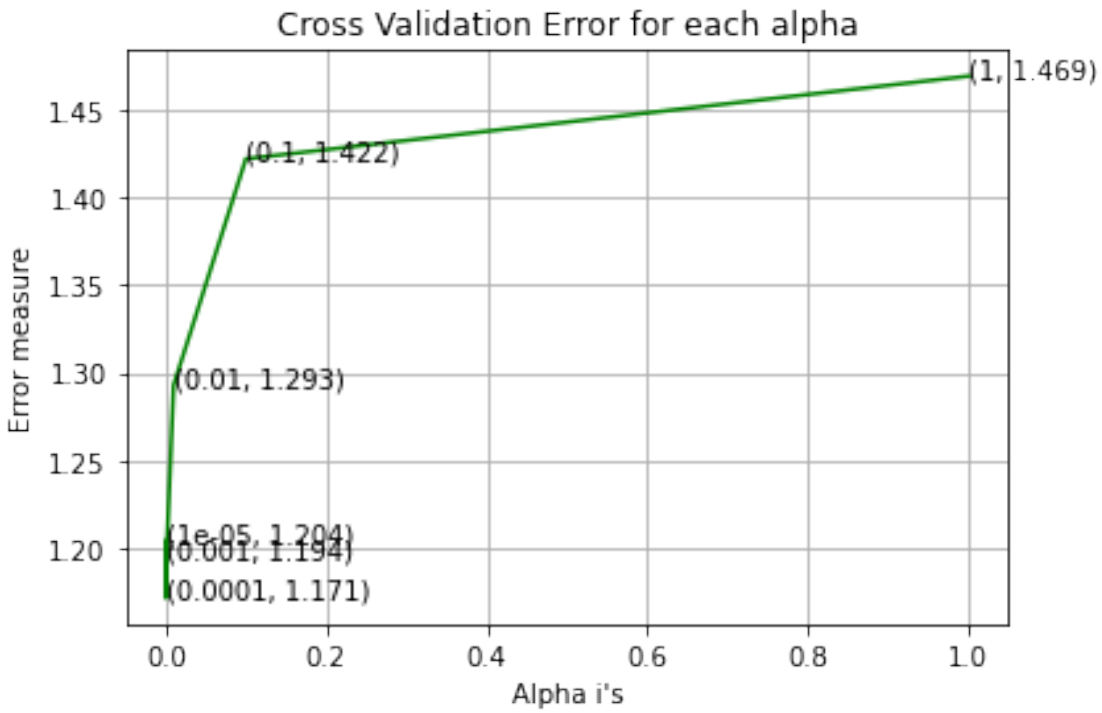
Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

[ ]:
```
alpha = [10 ** x for x in range(-5, 1)] # 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,␣
 ↪predict_y, labels=clf.classes_, eps=1e-15))

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

```
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2037460869728638
For values of alpha =  0.0001 The log loss is: 1.1712707611988553
For values of alpha =  0.001 The log loss is: 1.193528239616325
For values of alpha =  0.01 The log loss is: 1.2928037893453217
For values of alpha =  0.1 The log loss is: 1.4220473499737907
For values of alpha =  1 The log loss is: 1.4692623928282829
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9919226966540244
For values of best alpha =  0.0001 The cross validation log loss is:
1.1712707611988553
For values of best alpha =  0.0001 The test log loss is: 1.2058017184695575
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

[ ]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",␣
 ↪unique_genes.shape[0], " genes in train dataset?")

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].
 ↪shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":
 ↪",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"␣
 ↪,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  232  genes
in train dataset?
Ans
1. In test data 641 out of 665 : 96.39097744360903
2. In cross validation data 520 out of  532 : 97.74436090225564
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

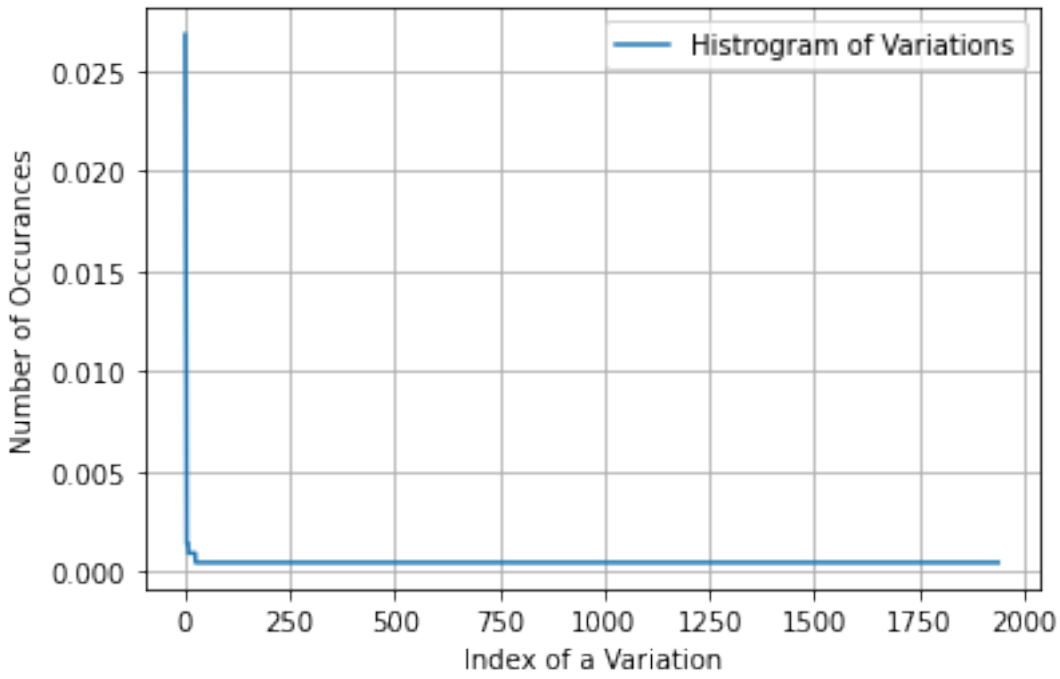
[ ]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1940
Truncating_Mutations    57
Deletion                46
Amplification           40
Fusions                 20
G12V                     3
E17K                     3
Q61L                     3
Q61H                     3
Q61R                     2
R170W                    2
Name: Variation, dtype: int64
```

[ ]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of␣
 ↪variations in the train data, and they are distibuted as follows",)
```

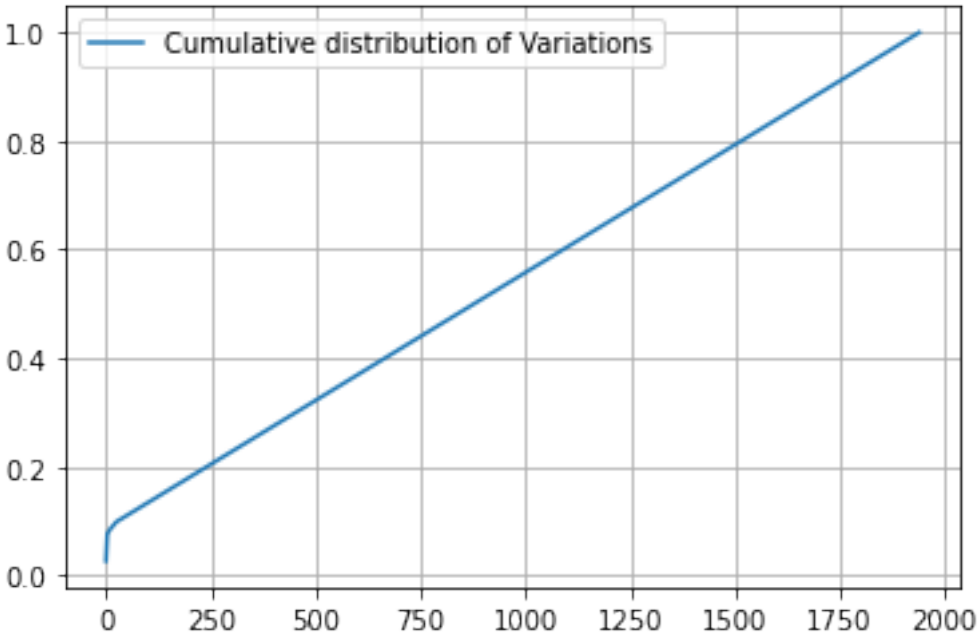
```
Ans: There are 1940 different categories of variations in the train data, and
they are distibuted as follows
```

[ ]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



[ ]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02683616 0.04849341 0.0673258  ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans.There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One hot Encoding

Response coding

We will be using both these methods to featurize the Variation Feature

[68]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", cv_df))
```

[ ]:
```
print("train_variation_feature_responseCoding is a converted feature using the␣
 ↪response coding method. The shape of Variation feature:",␣
 ↪train_variation_feature_responseCoding.shape)
```

```
train_variation_feature_responseCoding is a converted feature using the response
coding method. The shape of Variation feature: (2124, 9)
```

[69]:
```
# one-hot encoding of variation feature.
variation_vectorizer =  TfidfVectorizer(max_features=1000)
train_variation_feature_onehotCoding = variation_vectorizer.
 ↪fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.
 ↪transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.
 ↪transform(cv_df['Variation'])
```

[ ]:
```
print("train_variation_feature_onehotEncoded is converted feature using the␣
 ↪onne-hot encoding method. The shape of Variation feature:",␣
 ↪train_variation_feature_onehotCoding.shape)
```

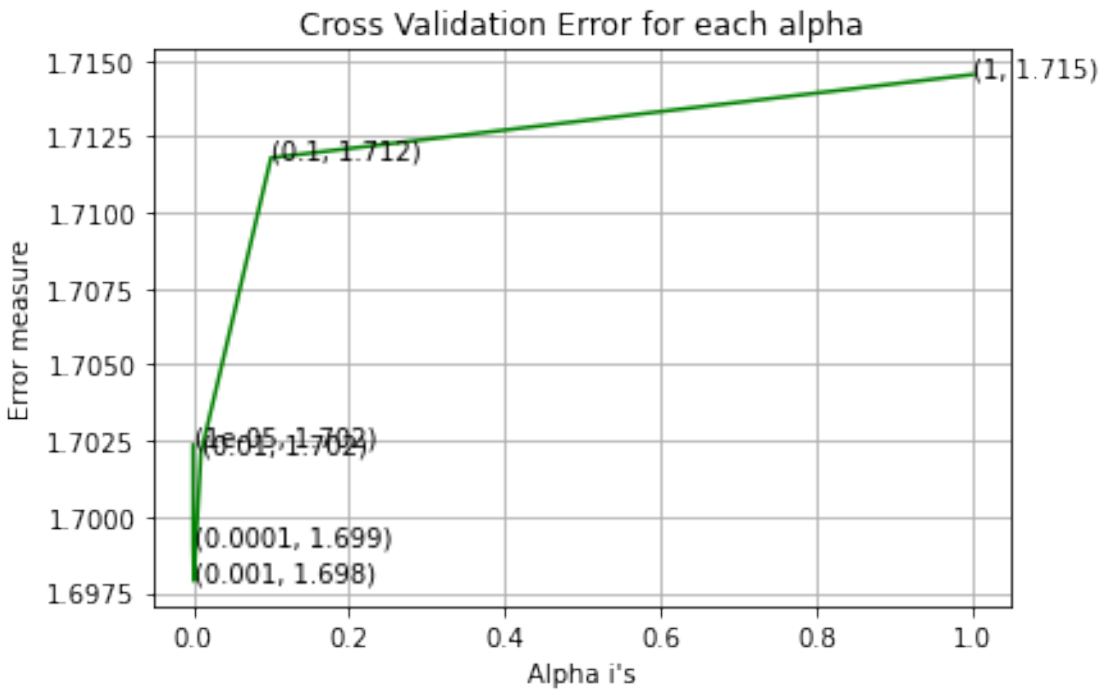
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1000)
```

Q10. How good is this Variation feature in predicting y_i?
Let's build a model just like the earlier!

[ ]:
```
alpha = [10 ** x for x in range(-5, 1)]

# 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

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

```
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,␣
 ↪predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7023537102871553
For values of alpha =  0.0001 The log loss is: 1.699099058085448
For values of alpha =  0.001 The log loss is: 1.6978888845025468
For values of alpha =  0.01 The log loss is: 1.7020830835751843
For values of alpha =  0.1 The log loss is: 1.7117981634430948
For values of alpha =  1 The log loss is: 1.7145342142941327
```




```
For values of best alpha =  0.001 The train log loss is: 1.3739490811646562
For values of best alpha =  0.001 The cross validation log loss is:
1.6978888845025468
For values of best alpha =  0.001 The test log loss is: 1.7009346113248875
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Not sure! But lets be very sure using the below analysis.

[ ]:
```
print("Q12. How many data points are covered by total ", unique_variations.
 ↪shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].
 ↪isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].
 ↪shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":
 ↪",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"␣
 ↪,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1940  genes in test and cross
validation data sets?
Ans
1. In test data 72 out of 665 : 10.827067669172932
2. In cross validation data 58 out of  532 : 10.902255639097744
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

[ ]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
```

[ ]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/
 ↪(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/
 ↪len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

[72]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer =  TfidfVectorizer(min_df = 3, max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.
 ↪fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns␣
 ↪(1*number of features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1
```

```
# zip(list(text_features),text_fea_counts) will zip a word with its number of␣
 ↪times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


print("Total number of unique words in train data :", len(train_text_features))
```

```
Total number of unique words in train data : 1000
```

[ ]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

[ ]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

[ ]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/
 ↪train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
 ↪test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/
 ↪cv_text_feature_responseCoding.sum(axis=1)).T
```

[74]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding,␣
 ↪axis=0)
```

```
# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding,␣
 ↪axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

[ ]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] ,␣
 ↪reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

[ ]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({255.04972040872224: 1, 181.56862357255199: 1, 140.44201958806687: 1,
130.63680297794357: 1, 129.92841283763627: 1, 120.63450967639037: 1,
120.16772790509471: 1, 116.13917660538034: 1, 111.93501521263111: 1,
108.59746828838426: 1, 107.01718350958444: 1, 89.94960922385417: 1,
89.19136389552048: 1, 84.26130091497838: 1, 81.05534466550755: 1,
79.47692375148212: 1, 79.0828748891833: 1, 78.94453011051094: 1,
78.31058926610032: 1, 77.69230299479393: 1, 76.75396000930728: 1,
76.20233057593175: 1, 71.13739120323973: 1, 69.5173784279098: 1,
68.96637114574993: 1, 68.67250867174059: 1, 66.38004390782167: 1,
66.15557573147358: 1, 65.54495335777578: 1, 64.91141907792266: 1,
64.80707481432057: 1, 64.02443307230844: 1, 63.480864836589475: 1,
59.584368352310705: 1, 59.153730467103024: 1, 57.55279123183286: 1,
56.98517993156729: 1, 56.83283752332981: 1, 52.940749867967725: 1,
51.28058414628144: 1, 50.69486309254998: 1, 50.27148397940525: 1,
50.23630736893718: 1, 50.088880888242045: 1, 48.67583186234295: 1,
48.40997845287781: 1, 47.93549869223447: 1, 47.90558019889383: 1,
47.46748818355218: 1, 46.09735544131558: 1, 45.38835888174282: 1,
44.74509494803155: 1, 43.76721827860422: 1, 43.73052173441656: 1,
43.682368546067174: 1, 43.478097424680925: 1, 43.31872064525182: 1,
42.72708226417405: 1, 42.095975226512955: 1, 41.97404833737468: 1,
41.774214986358416: 1, 41.72912418491017: 1, 41.44606112608632: 1,
41.26243359728778: 1, 41.13477781173304: 1, 41.02764852404993: 1,
40.239774166549: 1, 39.88803421885259: 1, 39.54628064349951: 1,
39.28776341405644: 1, 38.99552433073783: 1, 38.388616576805994: 1,
38.349409598216006: 1, 37.72482885888074: 1, 37.41605581992676: 1,
37.10935739748791: 1, 37.08264184956396: 1, 37.01109944467859: 1,
36.94656990767372: 1, 36.86811691478386: 1, 36.62074886672974: 1,
36.32252873984548: 1, 36.20693614395933: 1, 35.93643670744269: 1,
```

35.67472828196405: 1, 35.63376391156963: 1, 35.596199914459206: 1,
34.66755130480336: 1, 34.65644374225489: 1, 33.77922033487365: 1,
33.68648714578218: 1, 33.454512553991776: 1, 33.179511637087856: 1,
33.11272606364582: 1, 32.88154095896932: 1, 32.843456572581616: 1,
32.697598160476346: 1, 32.67994579896331: 1, 32.6734867146166: 1,
32.45934628452946: 1, 32.13881364451859: 1, 32.135365813498794: 1,
31.86439333929354: 1, 31.824234639658282: 1, 31.65138166318625: 1,
31.64859735160251: 1, 31.64610553700356: 1, 31.554841715451612: 1,
31.488026828716286: 1, 31.452417994842023: 1, 31.195656876506067: 1,
31.187035822977688: 1, 31.080308981228058: 1, 30.85741043369895: 1,
30.744857592884387: 1, 30.69062430028399: 1, 30.620632439432033: 1,
30.404497426501703: 1, 30.320445978339542: 1, 30.238169658895433: 1,
30.21781974773519: 1, 29.587737043581587: 1, 29.532297055821605: 1,
29.495255542607065: 1, 29.186141263159133: 1, 29.022848485546497: 1,
28.922178304141777: 1, 28.917056344389174: 1, 28.877981010668645: 1,
28.5193552489952: 1, 28.16032903053864: 1, 28.15816930868154: 1,
28.006481928328853: 1, 27.997096176971365: 1, 27.89218295887293: 1,
27.80619235746622: 1, 27.657592374458282: 1, 27.613460067361846: 1,
27.34603701005524: 1, 27.17771214561733: 1, 27.052752492993914: 1,
26.79552743633719: 1, 26.753563230346572: 1, 26.720941015691352: 1,
26.669001926581437: 1, 26.59373941143874: 1, 26.21107819036529: 1,
26.052240538200092: 1, 26.02462426501239: 1, 25.930246093166932: 1,
25.88491059975813: 1, 25.87255373722604: 1, 25.75538031230356: 1,
25.65567628851535: 1, 25.564093965155436: 1, 25.49422999860785: 1,
25.40466305205688: 1, 25.330469810593772: 1, 25.16334131643505: 1,
25.122030143654115: 1, 24.999792947461163: 1, 24.858746484715684: 1,
24.844309105884694: 1, 24.712017496098877: 1, 24.619981802586185: 1,
24.615551096390984: 1, 24.480212872411162: 1, 24.47904591113123: 1,
24.450168329397766: 1, 24.449516772040404: 1, 24.317816734149563: 1,
24.230966274739846: 1, 24.045464062184244: 1, 23.88584978136314: 1,
23.86959809367758: 1, 23.802770464596573: 1, 23.78356000368878: 1,
23.680769100259162: 1, 23.64305778555242: 1, 23.58553570985195: 1,
23.523473114929505: 1, 23.458687808843703: 1, 23.361359059747933: 1,
23.19223551303497: 1, 23.18223810316792: 1, 23.176756356301237: 1,
23.15163814254794: 1, 23.140745048572665: 1, 23.051474748791318: 1,
22.979289993200695: 1, 22.96150544322193: 1, 22.914924893577442: 1,
22.864515868493076: 1, 22.86202952427329: 1, 22.780199590230517: 1,
22.750939135784215: 1, 22.712958679364434: 1, 22.68706823100372: 1,
22.68194612589554: 1, 22.516313387519173: 1, 22.45096497542528: 1,
22.3389836357219: 1, 22.255509574380046: 1, 22.165456481493447: 1,
22.1584515521233: 1, 22.13550933802428: 1, 22.116445869155: 1,
22.02717246413062: 1, 22.01477140042435: 1, 21.925104573469877: 1,
21.766641219606026: 1, 21.726769595032902: 1, 21.70112510523254: 1,
21.677855982567536: 1, 21.658962917458695: 1, 21.63894381174161: 1,
21.552450625891005: 1, 21.463730627061352: 1, 21.334962136697445: 1,
21.334574428156852: 1, 21.32815163583993: 1, 21.29769200899373: 1,
21.26677386878127: 1, 21.265000102349017: 1, 21.252670476291062: 1,
21.220497442434702: 1, 21.183606459626155: 1, 21.17231412626937: 1,

21.112854320318288: 1, 21.06877728192226: 1, 20.941427555495128: 1,
20.938851920324154: 1, 20.93082425683578: 1, 20.843567816251678: 1,
20.828845147736676: 1, 20.760481378922726: 1, 20.698997920863196: 1,
20.695846888852074: 1, 20.693131410849: 1, 20.54791893502529: 1,
20.529394077086074: 1, 20.510673297785594: 1, 20.47067737267965: 1,
20.396233639948264: 1, 20.377182235790418: 1, 20.29417920215168: 1,
20.198095842947403: 1, 20.133768810246643: 1, 20.02719120459024: 1,
19.981624967026097: 1, 19.95255476886027: 1, 19.884093309312515: 1,
19.774699967420872: 1, 19.771998835760577: 1, 19.74491337037835: 1,
19.70887236002135: 1, 19.69456094760745: 1, 19.693151680931763: 1,
19.68795197766164: 1, 19.679626178507366: 1, 19.644727947898513: 1,
19.593687454668075: 1, 19.563522779382268: 1, 19.544627130057002: 1,
19.50602659267242: 1, 19.460423003352606: 1, 19.44440746445248: 1,
19.33228358942679: 1, 19.283646375348685: 1, 19.250797077541858: 1,
19.230201596603916: 1, 19.172757300173878: 1, 19.104465822673465: 1,
19.10442168501124: 1, 19.0935503469063: 1, 19.085803747679766: 1,
19.080285573271446: 1, 19.01409149171531: 1, 18.97947462963999: 1,
18.970325624073688: 1, 18.88537889299989: 1, 18.858093548854388: 1,
18.839588098255593: 1, 18.82807512283918: 1, 18.827515049231888: 1,
18.783838035842248: 1, 18.69085472764108: 1, 18.674187335384623: 1,
18.660228238679476: 1, 18.65835371753106: 1, 18.638894820002427: 1,
18.597704996062646: 1, 18.531340310494517: 1, 18.455048285252246: 1,
18.454282429995303: 1, 18.452375891539692: 1, 18.42005088257811: 1,
18.41984586759515: 1, 18.419458811010376: 1, 18.346359229792135: 1,
18.32493509942301: 1, 18.276182671252606: 1, 18.25521150770399: 1,
18.25127045756656: 1, 18.22300012909806: 1, 18.206199730932415: 1,
18.1774115884031: 1, 18.06257829907999: 1, 18.03287552596514: 1,
17.965950940254622: 1, 17.956600852491892: 1, 17.950217088144626: 1,
17.94812910049302: 1, 17.929595198789368: 1, 17.887807170668598: 1,
17.862777722741185: 1, 17.84043635994736: 1, 17.835715305564115: 1,
17.732725565659944: 1, 17.729247148970103: 1, 17.68542477941812: 1,
17.68263454385546: 1, 17.672321419473395: 1, 17.663954392122523: 1,
17.64963119108511: 1, 17.628539942337575: 1, 17.6163742263191: 1,
17.610048226477875: 1, 17.541804495744604: 1, 17.448194032071736: 1,
17.43535360232438: 1, 17.39282747899639: 1, 17.376487092600314: 1,
17.3580736102413: 1, 17.298248688577488: 1, 17.26480687845259: 1,
17.223512567693668: 1, 17.185053376635963: 1, 17.12447379536949: 1,
17.121837719048994: 1, 17.083712505549528: 1, 17.080075661345322: 1,
17.068864661170032: 1, 17.049483771445917: 1, 17.03939626601441: 1,
17.037090886837856: 1, 16.976343258617298: 1, 16.96206037604358: 1,
16.96198118650064: 1, 16.96086960899695: 1, 16.946975649053506: 1,
16.924818033862522: 1, 16.92269234383309: 1, 16.921215433068298: 1,
16.91768859441963: 1, 16.886515798442986: 1, 16.78360296676335: 1,
16.739280143156765: 1, 16.735301636161275: 1, 16.715693366914298: 1,
16.715307065741886: 1, 16.686962394362567: 1, 16.652646857410762: 1,
16.626289105574678: 1, 16.605433023524792: 1, 16.548470994388293: 1,
16.524938019769206: 1, 16.456944718436826: 1, 16.409764299527932: 1,
16.384056939088552: 1, 16.363355720443494: 1, 16.35260689524525: 1,

16.32251948475322: 1, 16.29648440284627: 1, 16.280661423423254: 1,
16.193262081108802: 1, 16.191177250412576: 1, 16.145045151423254: 1,
16.134728738337316: 1, 16.105948149748368: 1, 16.082342117041026: 1,
16.079237686565154: 1, 15.99514390648299: 1, 15.991475677857657: 1,
15.984548514919567: 1, 15.966225139014869: 1, 15.934753982605432: 1,
15.934667948837731: 1, 15.896937820231873: 1, 15.861700565828126: 1,
15.856532744043408: 1, 15.855323670103102: 1, 15.84980580313245: 1,
15.758153851214031: 1, 15.710415766152128: 1, 15.686241518704632: 1,
15.63869971766946: 1, 15.623272372514926: 1, 15.550522121073849: 1,
15.533983030811156: 1, 15.527084614136147: 1, 15.512653869210768: 1,
15.504756238864397: 1, 15.442924761185916: 1, 15.368612402447992: 1,
15.36269596312725: 1, 15.3157840638757: 1, 15.309986611735146: 1,
15.195698243419514: 1, 15.184433591673134: 1, 15.17937750741217: 1,
15.176687305593445: 1, 15.172363893874339: 1, 15.166523411761759: 1,
15.146125708899907: 1, 15.129152491263563: 1, 15.118381235083385: 1,
15.107764829372343: 1, 15.107462053306971: 1, 15.076554755877345: 1,
15.072399157822936: 1, 15.013222102113154: 1, 15.010178680130418: 1,
15.005340045902162: 1, 14.987946872219208: 1, 14.975056135677454: 1,
14.940071972905374: 1, 14.923746610924495: 1, 14.876965973489522: 1,
14.874711270995991: 1, 14.86803721849218: 1, 14.821185180646577: 1,
14.79988439325074: 1, 14.799031899571217: 1, 14.771693208620242: 1,
14.759177051627004: 1, 14.753382898855698: 1, 14.749496075100252: 1,
14.748280378313279: 1, 14.732533522323838: 1, 14.717334747419189: 1,
14.679924060475775: 1, 14.672724811443294: 1, 14.654065545134749: 1,
14.634197202862088: 1, 14.606697635180469: 1, 14.60405365623659: 1,
14.592016568891315: 1, 14.567742257597212: 1, 14.558241512643878: 1,
14.525858407129595: 1, 14.519022451497081: 1, 14.484684747663525: 1,
14.483785209656814: 1, 14.482942775456559: 1, 14.481327044262903: 1,
14.446115638435582: 1, 14.442881843650511: 1, 14.432379532028762: 1,
14.414641503728463: 1, 14.366720359005905: 1, 14.331499816224168: 1,
14.323691250272192: 1, 14.31378157998439: 1, 14.28033257864168: 1,
14.275381148445629: 1, 14.27295896209933: 1, 14.228141367451952: 1,
14.203353656033594: 1, 14.173472066888593: 1, 14.125071005284898: 1,
14.114959441140106: 1, 14.112121362063784: 1, 14.069770766474697: 1,
14.048380140541044: 1, 13.996919449091772: 1, 13.992016582256854: 1,
13.959570629413284: 1, 13.906891070270248: 1, 13.872932579610968: 1,
13.860908352093475: 1, 13.845269035068114: 1, 13.773225659942923: 1,
13.72826176932314: 1, 13.67835542000415: 1, 13.60590227755828: 1,
13.605076941909374: 1, 13.579036173113966: 1, 13.57367164352365: 1,
13.559929542073458: 1, 13.555184325306108: 1, 13.519780679482455: 1,
13.512915449957937: 1, 13.450261732472475: 1, 13.44977036568961: 1,
13.44966394709856: 1, 13.449150793690144: 1, 13.445644301410344: 1,
13.395687987391627: 1, 13.36178809820618: 1, 13.35799585633814: 1,
13.343146032741643: 1, 13.314294654861953: 1, 13.247269582912926: 1,
13.24633045869193: 1, 13.233766099456211: 1, 13.206574538178687: 1,
13.173300624395193: 1, 13.153892971293844: 1, 13.141219101688556: 1,
13.140288557145922: 1, 13.132966320245846: 1, 13.117152295683402: 1,
13.07059761698197: 1, 13.037674107173734: 1, 13.027307649019225: 1,

13.012428072565102: 1, 12.9899979617699: 1, 12.988636616154073: 1,
12.974371659049693: 1, 12.945963381037863: 1, 12.938179579075376: 1,
12.92355994771836: 1, 12.902064945095349: 1, 12.872415111590115: 1,
12.834971993616005: 1, 12.833672340557255: 1, 12.827144572548956: 1,
12.806607557853548: 1, 12.80594566614643: 1, 12.731683459549183: 1,
12.717895440971661: 1, 12.705070759545766: 1, 12.702030823201545: 1,
12.666660097750468: 1, 12.656192338854812: 1, 12.64811580643248: 1,
12.619606006719932: 1, 12.613587287120096: 1, 12.597011793923773: 1,
12.582625051850538: 1, 12.581360643793653: 1, 12.576495461670198: 1,
12.5696919431825: 1, 12.543041201219241: 1, 12.509668019859923: 1,
12.492404309501222: 1, 12.476474399808332: 1, 12.454976422250718: 1,
12.428161417420595: 1, 12.42662053010953: 1, 12.401409003846343: 1,
12.399643550976686: 1, 12.396663966583317: 1, 12.392742347521512: 1,
12.386478519619406: 1, 12.36682589501154: 1, 12.361093849851379: 1,
12.359409110533663: 1, 12.352065346202835: 1, 12.309251588741645: 1,
12.30720597232028: 1, 12.296509060586658: 1, 12.267870992917377: 1,
12.258303328693263: 1, 12.182744636274284: 1, 12.142289012070274: 1,
12.102997779326325: 1, 12.095022009589982: 1, 12.061178048770678: 1,
12.045494437914027: 1, 12.027185785403244: 1, 12.01760421889543: 1,
12.00271200946868: 1, 12.00233898862083: 1, 11.947336487110844: 1,
11.94020460202953: 1, 11.924692460260761: 1, 11.924316305765293: 1,
11.914801399805011: 1, 11.905231083423562: 1, 11.841373216796779: 1,
11.83910626096046: 1, 11.827976063114065: 1, 11.819282374082613: 1,
11.81304573344322: 1, 11.810158197432669: 1, 11.80717616388726: 1,
11.801389982152191: 1, 11.7944361018612: 1, 11.788389168985512: 1,
11.730041530884675: 1, 11.724614367879306: 1, 11.722677589109306: 1,
11.721880315105066: 1, 11.702142013389428: 1, 11.680475013846705: 1,
11.675284994280856: 1, 11.669931319324544: 1, 11.609845333966701: 1,
11.584262954685812: 1, 11.582963953793781: 1, 11.574723570292889: 1,
11.56154061567548: 1, 11.536670352566732: 1, 11.528689654311847: 1,
11.492147671510693: 1, 11.472172486199685: 1, 11.469349407595345: 1,
11.459758865322518: 1, 11.451740268401272: 1, 11.441959622646982: 1,
11.432599822320956: 1, 11.38191022791747: 1, 11.380426579580538: 1,
11.370721340717687: 1, 11.319513289532996: 1, 11.312393212977469: 1,
11.312112651736074: 1, 11.300955497174652: 1, 11.299725670663875: 1,
11.2762963395603: 1, 11.271426711240727: 1, 11.261734509438428: 1,
11.252335835328367: 1, 11.23624495030003: 1, 11.233297638515472: 1,
11.209895063771889: 1, 11.195732974641382: 1, 11.169954970426291: 1,
11.15962870452563: 1, 11.154676234625576: 1, 11.153667629678727: 1,
11.133853000335122: 1, 11.124045954483226: 1, 11.121530738266161: 1,
11.11641683151529: 1, 11.114793099784684: 1, 11.056042087470972: 1,
11.03093552678492: 1, 10.973412377238654: 1, 10.942128490202613: 1,
10.93880283104951: 1, 10.93782120382247: 1, 10.914535842404293: 1,
10.89389107728953: 1, 10.891398028713313: 1, 10.891120290641433: 1,
10.884995423853354: 1, 10.86835130688834: 1, 10.860746659343505: 1,
10.859912851602592: 1, 10.857737882629602: 1, 10.851530134734363: 1,
10.823558546281374: 1, 10.819749212553647: 1, 10.808162048676232: 1,
10.807037689429567: 1, 10.806909537168313: 1, 10.793599115636686: 1,

10.784268015740054: 1, 10.781453854178906: 1, 10.765586903008561: 1,
10.765286682393084: 1, 10.736381293807838: 1, 10.726261608572594: 1,
10.723327943634699: 1, 10.702624942304972: 1, 10.701966343164223: 1,
10.693490456233254: 1, 10.692065038759702: 1, 10.689598999930865: 1,
10.688891191554728: 1, 10.685691854989782: 1, 10.68356404752287: 1,
10.682851622036763: 1, 10.662432986349108: 1, 10.656763126592294: 1,
10.64638923611467: 1, 10.641018555415675: 1, 10.629913236579908: 1,
10.595944917662573: 1, 10.591790455308637: 1, 10.581085642169443: 1,
10.577013628325128: 1, 10.576054836290195: 1, 10.562835908432701: 1,
10.516751879398415: 1, 10.501962635159433: 1, 10.481945503513288: 1,
10.472081738040458: 1, 10.47113017656581: 1, 10.468102609952515: 1,
10.462461917166802: 1, 10.456557059495905: 1, 10.449978863200657: 1,
10.442220711959607: 1, 10.40791447417601: 1, 10.394708209758313: 1,
10.37986945309899: 1, 10.365911015655012: 1, 10.33807510836911: 1,
10.321463060114425: 1, 10.293953440990697: 1, 10.28539557789001: 1,
10.281881857885923: 1, 10.279800098872025: 1, 10.26604193604157: 1,
10.259438316660765: 1, 10.251723940236623: 1, 10.248574013930904: 1,
10.240491191075499: 1, 10.214675403378942: 1, 10.18868321610676: 1,
10.162004160813208: 1, 10.159032347028942: 1, 10.145893623348222: 1,
10.144128427445642: 1, 10.129742932101486: 1, 10.125172357151024: 1,
10.074198722359306: 1, 10.062769512500738: 1, 10.04245738792345: 1,
10.03894442580614: 1, 10.03776323229906: 1, 10.027906679405461: 1,
10.024809652685411: 1, 10.022802257711897: 1, 10.014410320569862: 1,
10.014224994709851: 1, 10.012054520081296: 1, 10.00639947188646: 1,
9.981436354058253: 1, 9.978726226337365: 1, 9.97218806281339: 1,
9.919773965040985: 1, 9.905249739022315: 1, 9.900352488526442: 1,
9.860255007843504: 1, 9.84961948159886: 1, 9.834127446889072: 1,
9.8313227783833: 1, 9.824261258359245: 1, 9.817689012095762: 1,
9.794741418212942: 1, 9.776176816709972: 1, 9.772120437000881: 1,
9.751303051547243: 1, 9.744605746621191: 1, 9.7434894196925: 1,
9.741265669429264: 1, 9.72420231682961: 1, 9.722598874654832: 1,
9.711938859395072: 1, 9.702127060015696: 1, 9.701711984487469: 1,
9.690672908558646: 1, 9.686732568656762: 1, 9.682405465594387: 1,
9.679006064051984: 1, 9.673371080857846: 1, 9.65156715287762: 1,
9.636156991048267: 1, 9.62910654333378: 1, 9.618621457538113: 1,
9.61703635850115: 1, 9.612387203249693: 1, 9.589749312039407: 1,
9.57976166781801: 1, 9.548364568532532: 1, 9.53883316903584: 1,
9.51928915078374: 1, 9.513832020277196: 1, 9.513428861341634: 1,
9.503599463656345: 1, 9.500581206799136: 1, 9.48728069869503: 1, 9.475137089186:
1, 9.46094247247219: 1, 9.456168766807759: 1, 9.450475131831329: 1,
9.449248975894006: 1, 9.427446605610953: 1, 9.417622256780795: 1,
9.413089607734348: 1, 9.354037873603746: 1, 9.33466085904733: 1,
9.330639888339103: 1, 9.32885049253202: 1, 9.328625667451915: 1, 9.312711394764:
1, 9.31236572360172: 1, 9.306652312510021: 1, 9.304678945174688: 1,
9.304447079897365: 1, 9.281913137331564: 1, 9.254446501980508: 1,
9.244629944580867: 1, 9.240686853312413: 1, 9.223359651312506: 1,
9.220929891470282: 1, 9.216212552038682: 1, 9.203658607604941: 1,
9.188806111063046: 1, 9.182052082745383: 1, 9.1808486236593: 1,

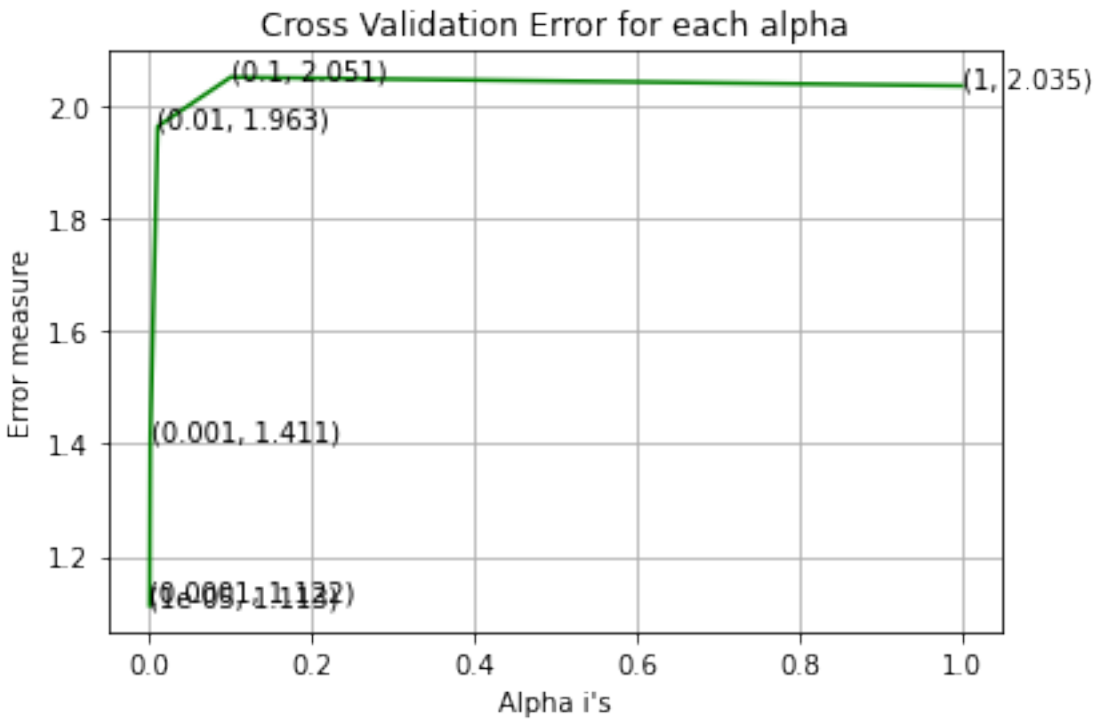
9.17650093279884: 1, 9.17473334505798: 1, 9.171479075230474: 1,
9.168254492825243: 1, 9.149708914607283: 1, 9.145601348260286: 1,
9.145179336388043: 1, 9.14305183251336: 1, 9.14203659548163: 1,
9.134856505518126: 1, 9.123174511420801: 1, 9.119563692942332: 1,
9.105890640756343: 1, 9.102567064223425: 1, 9.066704119783536: 1,
9.061039470381768: 1, 9.06073658380295: 1, 9.05329726601478: 1,
9.051723707755261: 1, 9.049201314353978: 1, 9.047237214175201: 1,
9.044235009451247: 1, 9.024980757278067: 1, 9.003338960444943: 1,
9.003084007882348: 1, 8.990322522328928: 1, 8.98433455060594: 1,
8.97657865770062: 1, 8.974843433398753: 1, 8.96945320455055: 1,
8.964225909424975: 1, 8.9619015625553: 1, 8.949946982671706: 1,
8.94374941310439: 1, 8.920392416261894: 1, 8.914046324649027: 1,
8.91268961430809: 1, 8.908640717412444: 1, 8.905491949971124: 1,
8.896006019299318: 1, 8.890164055839266: 1, 8.885210925861351: 1,
8.873584903473434: 1, 8.848753056739396: 1, 8.848236240863919: 1,
8.826271917103451: 1, 8.809514918986983: 1, 8.788004274217464: 1,
8.756950455166558: 1, 8.733608088505095: 1, 8.708950709623625: 1,
8.6966105142762: 1, 8.665721132341297: 1, 8.654808668185586: 1,
8.654570601602428: 1, 8.648309534943667: 1, 8.633280185421684: 1,
8.613590936777083: 1, 8.577745254913918: 1, 8.577398956724247: 1,
8.565362810347589: 1, 8.562223185910797: 1, 8.55601023927229: 1,
8.53074599400124: 1, 8.527631241522448: 1, 8.521250257275838: 1,
8.508811709224915: 1, 8.50176632820878: 1, 8.498001885761036: 1,
8.494311392943558: 1, 8.4727696110605: 1, 8.461984217660344: 1,
8.453744128831286: 1, 8.452155836278994: 1, 8.429874125474262: 1,
8.423873473178418: 1, 8.420024449201566: 1, 8.400599246394592: 1,
8.382301284104017: 1, 8.38122192172578: 1, 8.38065960487398: 1,
8.362935063599645: 1, 8.360240471946963: 1, 8.351223592092415: 1,
8.348432355512829: 1, 8.34290419606934: 1, 8.328598124275025: 1,
8.3210729872558: 1, 8.312552138537542: 1, 8.29429926770891: 1,
8.280636202864795: 1, 8.269490880073086: 1, 8.243381081171398: 1,
8.240560788477817: 1, 8.232718553468809: 1, 8.231166941589468: 1,
8.229754867177096: 1, 8.219545512139941: 1, 8.203551918595531: 1,
8.199726194935884: 1, 8.196667543039379: 1, 8.194653528946692: 1,
8.179624339098735: 1, 8.157950712140828: 1, 8.102624138849295: 1,
8.096000790942353: 1, 8.066554204061239: 1, 8.061792766243517: 1,
8.060463733782944: 1, 8.056509390380862: 1, 8.053484120396082: 1,
8.0305087806477: 1, 8.020930081576397: 1, 8.019082073205096: 1,
8.018411618695593: 1, 8.005253558288677: 1, 7.994476617575743: 1,
7.993774836870519: 1, 7.98964914144801: 1, 7.989581156070056: 1,
7.989540844424929: 1, 7.95810855579533: 1, 7.95105216147675: 1,
7.947880684095217: 1, 7.947842370312096: 1, 7.947055138267986: 1,
7.937174064844546: 1, 7.922280254041825: 1, 7.908350663956532: 1,
7.907940166215842: 1, 7.903457015261909: 1, 7.903036536299714: 1,
7.897066315072114: 1, 7.8831265526658045: 1, 7.876612265836396: 1,
7.846295595875528: 1, 7.839440304960756: 1, 7.829137402200846: 1,
7.822704809683382: 1, 7.814938631680795: 1, 7.798839228351989: 1,
7.777295776249997: 1, 7.773893086606074: 1, 7.758241411138575: 1,

```
7.739582836646035: 1, 7.736097950974734: 1, 7.711313721839665: 1,
7.690599541531203: 1, 7.685688598265415: 1, 7.65015270911309: 1,
7.623925585963859: 1, 7.58641362907211: 1, 7.582498540155795: 1,
7.575717072801508: 1, 7.571699711975207: 1, 7.5665264026999886: 1,
7.499564335811845: 1, 7.468727676892715: 1, 7.462227343877661: 1,
7.4549133348605965: 1, 7.429222414328697: 1, 7.41826924806706: 1,
7.410626545695093: 1, 7.407403665014663: 1, 7.405011628292186: 1,
7.3940510913872775: 1, 7.385279973309275: 1, 7.369935230153495: 1,
7.344164638321065: 1, 7.319479512088164: 1, 7.3125797547324956: 1,
7.3026532933793895: 1, 7.299534108300518: 1, 7.286369866238734: 1,
7.252886336646913: 1, 7.235380097221915: 1, 7.2046113401562355: 1,
7.177306770507079: 1, 7.1632278841022545: 1, 7.147599366245975: 1,
7.141161468434612: 1, 7.109868196126418: 1, 7.052943226064099: 1,
7.0430517140923: 1, 7.033667625307038: 1, 7.014742698725334: 1,
6.927765400427318: 1, 6.9157920914001405: 1, 6.793883716977359: 1,
6.790804430781415: 1, 6.7796784985611485: 1, 6.778537647152562: 1,
6.489408545945357: 1, 6.374097622230388: 1})
```

[ ]:
```
# Train a Logistic regression+Calibration model using text features whicha re␣
 ↪on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, y_train)
```

```
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,␣
 ↪predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.1133846697674203
For values of alpha =  0.0001 The log loss is: 1.1217360910784127
For values of alpha =  0.001 The log loss is: 1.410770052384869
For values of alpha =  0.01 The log loss is: 1.9632540260330835
For values of alpha =  0.1 The log loss is: 2.050637642644482
For values of alpha =  1 The log loss is: 2.0354127440177767
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7022168906066618
For values of best alpha =  1e-05 The cross validation log loss is:
1.1133846697674203
For values of best alpha =  1e-05 The test log loss is: 1.0798670975629097
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it seems like!

[ ]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

[ ]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train␣
 ↪data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in␣
 ↪train data")
```

```
94.0 % of word of test data appeared in train data
94.6 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

[ ]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities␣
 ↪belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y-␣
 ↪test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

[ ]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

[ ]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec =  TfidfVectorizer(max_features=1000)
    var_count_vec =  TfidfVectorizer(max_features=1000)
    text_count_vec =  TfidfVectorizer(min_df=3, max_features=1000)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
```

```
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".
 ↪format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point␣
 ↪[{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".
 ↪format(word,yes_no))

    print("Out of the top ",no_features," features ", word_present, "are␣
 ↪present in query point")
```

Stacking the three types of Tfidf features

[ ]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding =␣
 ↪hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =␣
 ↪hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding =␣
 ↪hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,␣
 ↪train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))
```

```
test_x_onehotCoding = hstack((test_gene_var_onehotCoding,␣
 ↪test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding,␣
 ↪cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.
 ↪hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.
 ↪hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.
 ↪hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,␣
 ↪train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding,␣
 ↪test_text_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding,␣
 ↪cv_text_feature_responseCoding))
```

[ ]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",␣
 ↪train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ",␣
 ↪test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data␣
 ↪=", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2232)
(number of data points * number of features) in test data =  (665, 2232)
(number of data points * number of features) in cross validation data = (532,
2232)
```

[ ]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",␣
 ↪train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ",␣
 ↪test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data␣
 ↪=", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
```

```
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532,
27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

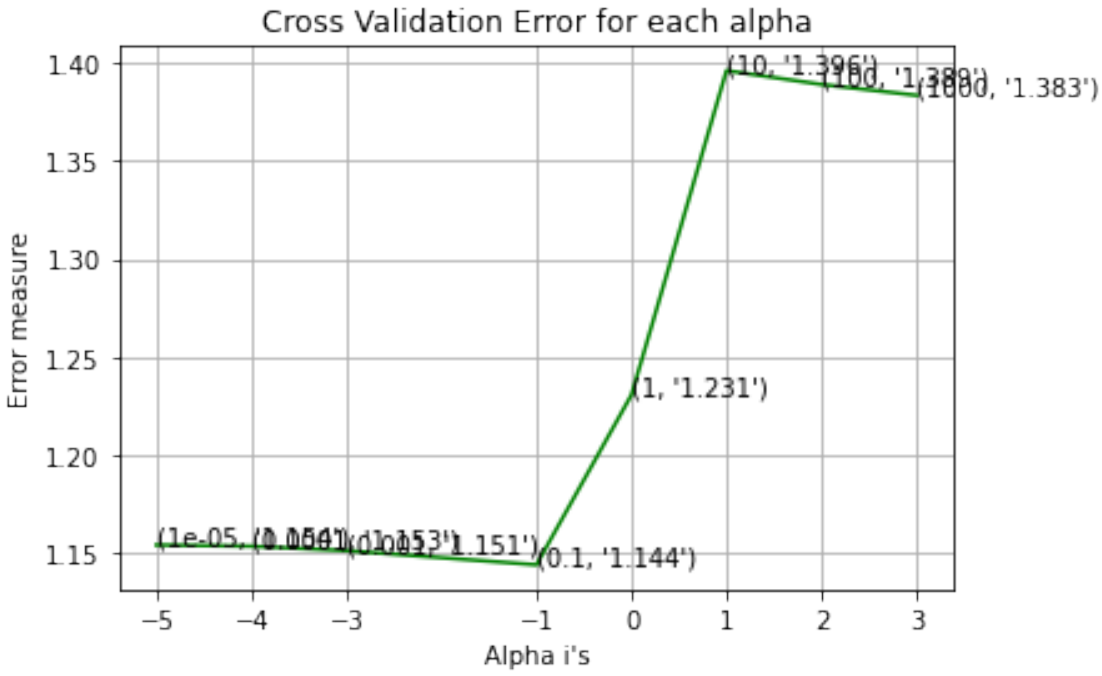
4.1.1.1. Hyper parameter tuning

[ ]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/
 ↪stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True,␣
 ↪class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X,␣
 ↪y
# predict(X)        Perform classification on an array of test vectors X.
# predict_log_proba(X)        Return log-probability estimates for the test␣
 ↪vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/naive-bayes-algorithm-1/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/naive-bayes-algorithm-1/
# -----------------------


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
```

```
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use␣
 ↪log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.1538771455041295
for alpha = 0.0001
Log Loss : 1.1532773797623161
for alpha = 0.001
Log Loss : 1.1510135556496344
```

```
for alpha = 0.1
Log Loss : 1.1437769053681766
for alpha = 1
Log Loss : 1.230529564291743
for alpha = 10
Log Loss : 1.3960571238013058
for alpha = 100
Log Loss : 1.3889854374216628
for alpha = 1000
Log Loss : 1.3834409302055215
```



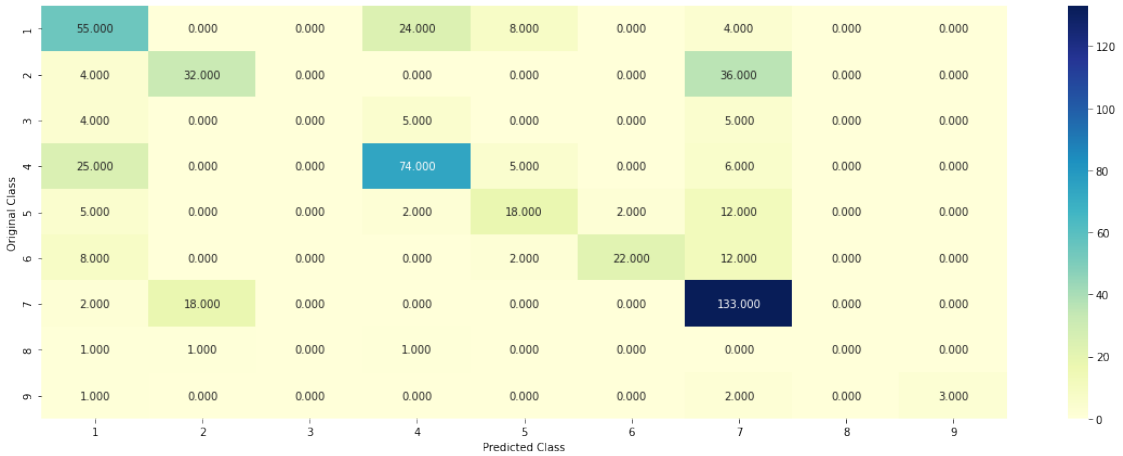

```
For values of best alpha =  0.1 The train log loss is: 0.780409551111857
For values of best alpha =  0.1 The cross validation log loss is:
1.1437769053681766
For values of best alpha =  0.1 The test log loss is: 1.1790687763208791
```

4.1.1.2. Testing the model with best hyper paramters

[ ]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/
 ↪stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True,␣
 ↪class_prior=None)

# some of methods of MultinomialNB()
```

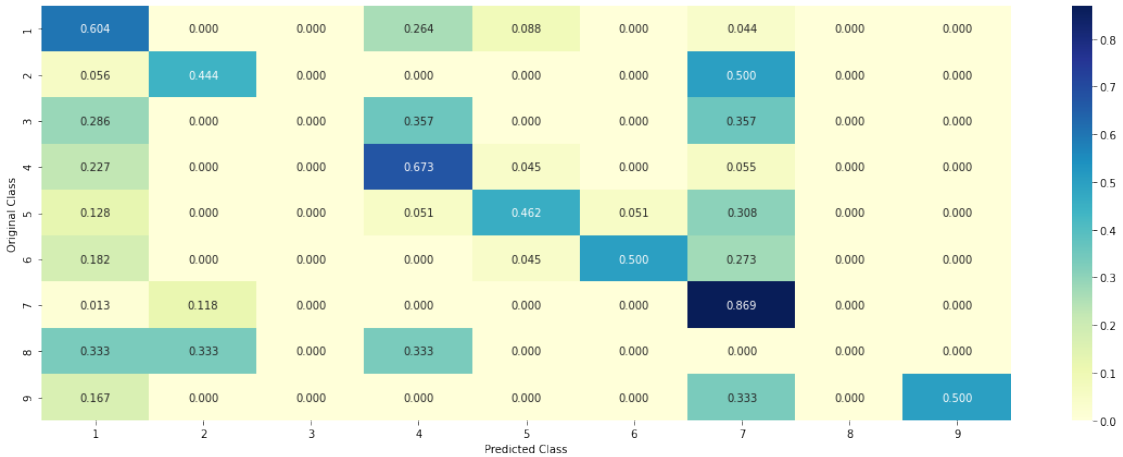
```
# fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X,␣
 ↪y
# predict(X)        Perform classification on an array of test vectors X.
# predict_log_proba(X)        Return log-probability estimates for the test␣
 ↪vector X.
# ------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/naive-bayes-algorithm-1/
# ------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
# ----------------------------

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability␣
 ↪estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.
 ↪predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.1437769053681766
Number of missclassified point : 0.36654135338345867
--------------------- Confusion matrix --------------------
```



---------- Precision matrix (Columm Sum=1) ----------

---------- Recall matrix (Row sum=1) ----------



4.1.1.3. Feature Importance, Correctly classified point

[ ]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0508 0.0871 0.0184 0.0484 0.0409 0.0401
0.7047 0.0061 0.0035]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

[ ]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4376 0.0638 0.0262 0.085  0.1999 0.1074
0.0661 0.0086 0.0055]]
Actual Class : 6
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

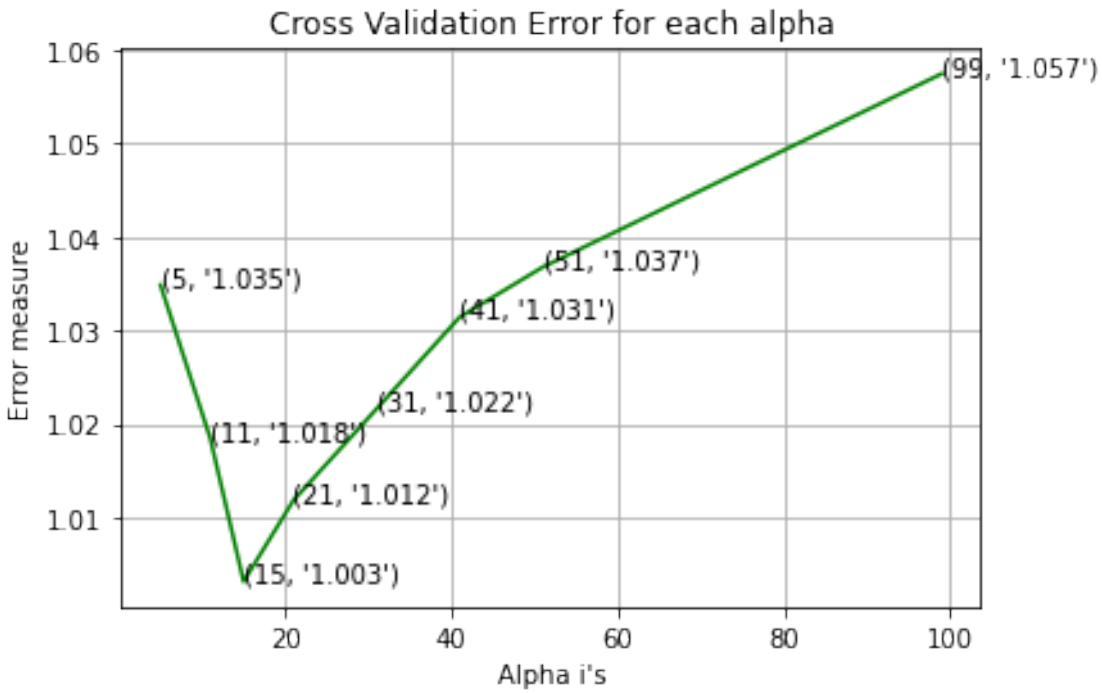
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

[ ]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# ----------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto,␣
 ↪leaf_size=30, p=2,
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#-----------------------------------
# video link:
#-----------------------------------


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
```

```
    # to avoid rounding error while multiplying probabilites we use␣
 ↪log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_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(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0348723448054626
for alpha = 11
Log Loss : 1.01845129410488
for alpha = 15
Log Loss : 1.0031880941484637
for alpha = 21
Log Loss : 1.0118678718053646
for alpha = 31
Log Loss : 1.0216278117195825
for alpha = 41
Log Loss : 1.0314358949669142
for alpha = 51
Log Loss : 1.036783743533819
for alpha = 99
Log Loss : 1.0574388881585233
```



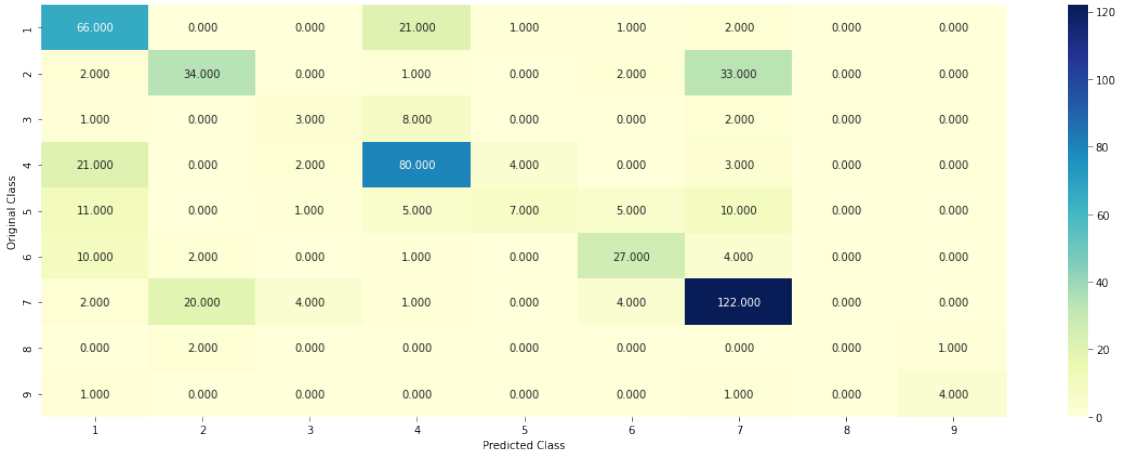

```
For values of best alpha =  15 The train log loss is: 0.661647960043316
For values of best alpha =  15 The cross validation log loss is:
1.0031880941484637
For values of best alpha =  15 The test log loss is: 1.0897661399243246
```

4.2.2. Testing the model with best hyper paramters

[ ]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto,␣
 ↪leaf_size=30, p=2,
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
```

```
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,␣
 ↪cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.0031880941484637
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```


```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.2.3.Sample Query point -1

[ ]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1,␣
 ↪-1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs␣
 ↪to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 6
The  15  nearest neighbours of the test points belongs to classes [6 5 1 6 6 6 1
1 1 1 1 5 4 4 5]
Fequency of nearest points : Counter({1: 6, 6: 4, 5: 3, 4: 2})
```

4.2.4. Sample Query Point-2

[ ]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100
```

```
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].
 ↪reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1,␣
 ↪-1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of␣
 ↪the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs
to classes [7 7 7 2 2 7 7 7 7 7 7 5 2 7 7]
Fequency of nearest points : Counter({7: 11, 2: 3, 5: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

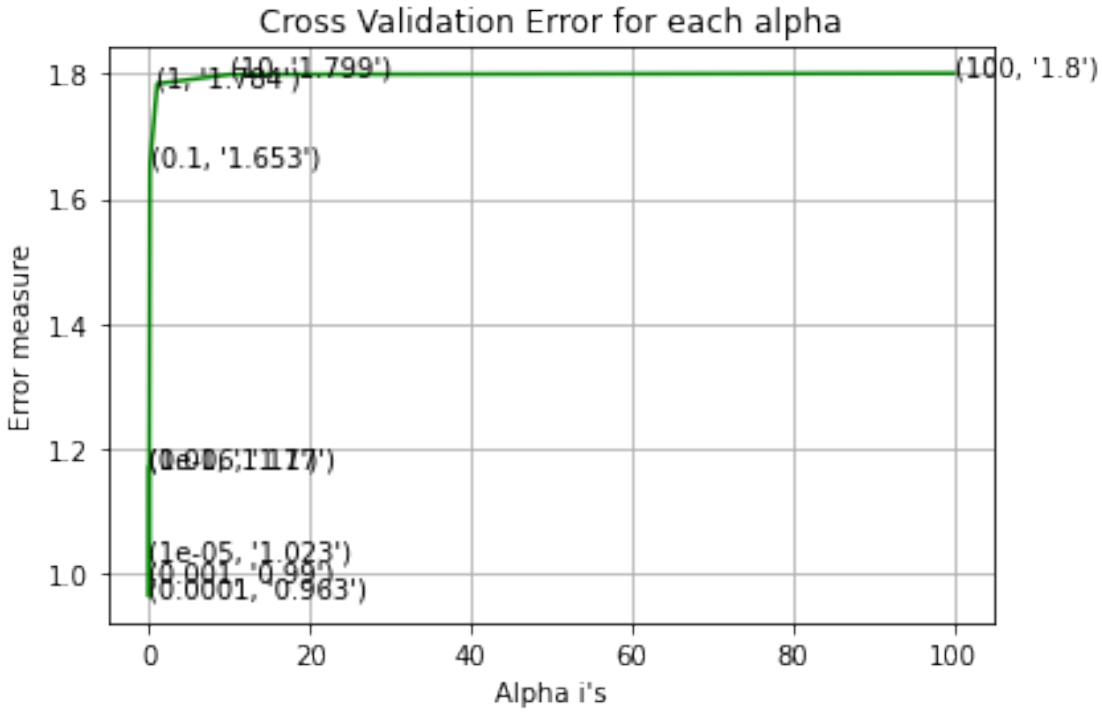
4.3.1.1. Hyper paramter tuning

[ ]:
```
# 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#-------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
```

```
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',␣
 ↪loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use␣
 ↪log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1697503573900314
for alpha = 1e-05
Log Loss : 1.0233671347320137
for alpha = 0.0001
Log Loss : 0.9628591088740591
for alpha = 0.001
Log Loss : 0.9900609560021435
for alpha = 0.01
Log Loss : 1.1699915163520418
for alpha = 0.1
Log Loss : 1.6530353402136744
for alpha = 1
Log Loss : 1.7838158882419415
for alpha = 10
Log Loss : 1.7986941439965871
for alpha = 100
Log Loss : 1.8004256233029055
```

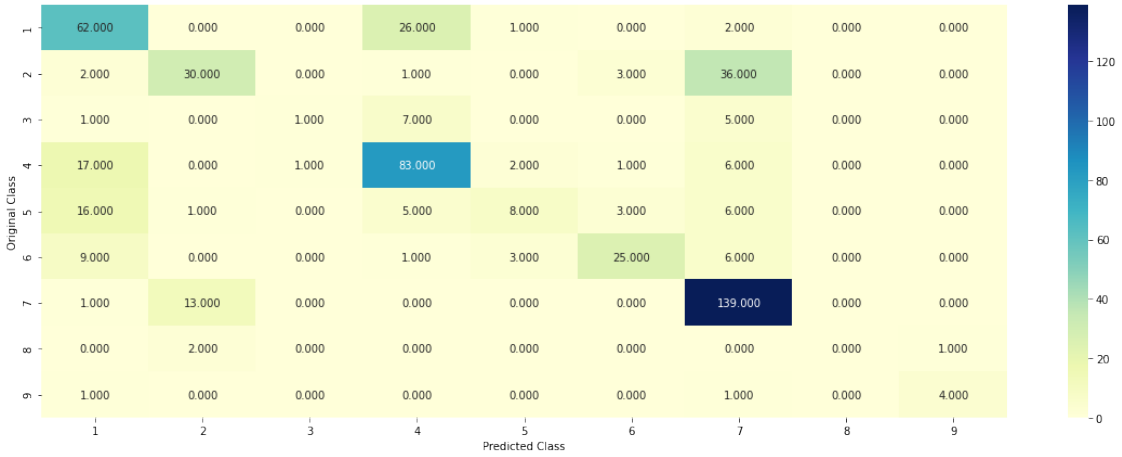


```
For values of best alpha =  0.0001 The train log loss is: 0.5479925495205686
For values of best alpha =  0.0001 The cross validation log loss is:
0.9628591088740591
For values of best alpha =  0.0001 The test log loss is: 0.9800900266428243
```

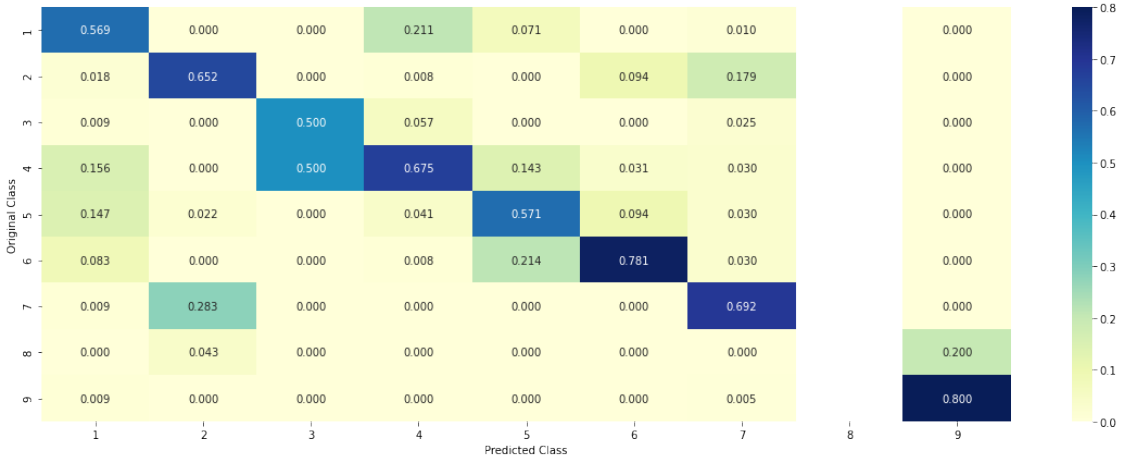
4.3.1.2. Testing the model with best hyper paramters

[ ]:
```
# 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,␣
 ↪cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 0.9628591088740591
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.3.1.3. Feature Importance

[ ]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
```

```
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i],␣
↪yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query␣
↪point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0],"␣
↪class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or␣
↪Not']))
```

4.3.1.3.1. Correctly Classified point

[ ]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0042 0.2259 0.0011 0.0075 0.0368 0.0645
0.6547 0.0041 0.0011]]
Actual Class : 7
--------------------------------------------------
14 Text feature [partial] present in test data point [True]
15 Text feature [oncogene] present in test data point [True]
20 Text feature [activation] present in test data point [True]
21 Text feature [constitutive] present in test data point [True]
30 Text feature [activated] present in test data point [True]
```

31 Text feature [phospho] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
49 Text feature [overexpression] present in test data point [True]
52 Text feature [activate] present in test data point [True]
56 Text feature [none] present in test data point [True]
57 Text feature [inhibited] present in test data point [True]
62 Text feature [important] present in test data point [True]
66 Text feature [codon] present in test data point [True]
68 Text feature [oncogenic] present in test data point [True]
71 Text feature [enhanced] present in test data point [True]
74 Text feature [ras] present in test data point [True]
76 Text feature [substrate] present in test data point [True]
81 Text feature [increased] present in test data point [True]
88 Text feature [reduced] present in test data point [True]
92 Text feature [group] present in test data point [True]
93 Text feature [advanced] present in test data point [True]
95 Text feature [mapk] present in test data point [True]
103 Text feature [signaling] present in test data point [True]
105 Text feature [phosphorylated] present in test data point [True]
107 Text feature [analyses] present in test data point [True]
112 Text feature [able] present in test data point [True]
114 Text feature [showing] present in test data point [True]
115 Text feature [reduction] present in test data point [True]
120 Text feature [second] present in test data point [True]
122 Text feature [adenocarcinoma] present in test data point [True]
124 Text feature [mechanisms] present in test data point [True]
126 Text feature [extracellular] present in test data point [True]
129 Text feature [3b] present in test data point [True]
131 Text feature [mek] present in test data point [True]
134 Text feature [occur] present in test data point [True]
138 Text feature [conserved] present in test data point [True]
139 Text feature [cancers] present in test data point [True]
153 Text feature [75] present in test data point [True]
158 Text feature [eight] present in test data point [True]
162 Text feature [different] present in test data point [True]
164 Text feature [factor] present in test data point [True]
165 Text feature [activating] present in test data point [True]
171 Text feature [pathways] present in test data point [True]
173 Text feature [s3] present in test data point [True]
175 Text feature [38] present in test data point [True]
186 Text feature [purified] present in test data point [True]
193 Text feature [concentrations] present in test data point [True]
194 Text feature [investigated] present in test data point [True]
200 Text feature [catenin] present in test data point [True]
202 Text feature [driven] present in test data point [True]
205 Text feature [predicted] present in test data point [True]
209 Text feature [lung] present in test data point [True]
214 Text feature [values] present in test data point [True]

```
218 Text feature [missense] present in test data point [True]
221 Text feature [secondary] present in test data point [True]
224 Text feature [akt] present in test data point [True]
225 Text feature [structural] present in test data point [True]
229 Text feature [inhibitor] present in test data point [True]
236 Text feature [loss] present in test data point [True]
241 Text feature [presence] present in test data point [True]
242 Text feature [bp] present in test data point [True]
243 Text feature [pik3ca] present in test data point [True]
249 Text feature [increase] present in test data point [True]
252 Text feature [positive] present in test data point [True]
254 Text feature [express] present in test data point [True]
255 Text feature [signalling] present in test data point [True]
259 Text feature [2011] present in test data point [True]
261 Text feature [leading] present in test data point [True]
264 Text feature [function] present in test data point [True]
267 Text feature [60] present in test data point [True]
274 Text feature [approximately] present in test data point [True]
275 Text feature [expressing] present in test data point [True]
278 Text feature [due] present in test data point [True]
279 Text feature [leukemia] present in test data point [True]
291 Text feature [hours] present in test data point [True]
300 Text feature [strongly] present in test data point [True]
301 Text feature [gtp] present in test data point [True]
302 Text feature [versus] present in test data point [True]
306 Text feature [31] present in test data point [True]
307 Text feature [significance] present in test data point [True]
310 Text feature [terminal] present in test data point [True]
313 Text feature [unknown] present in test data point [True]
314 Text feature [leads] present in test data point [True]
317 Text feature [correlation] present in test data point [True]
318 Text feature [hydrogen] present in test data point [True]
324 Text feature [2008] present in test data point [True]
325 Text feature [atp] present in test data point [True]
328 Text feature [phosphorylation] present in test data point [True]
331 Text feature [provided] present in test data point [True]
332 Text feature [01] present in test data point [True]
336 Text feature [based] present in test data point [True]
345 Text feature [highly] present in test data point [True]
350 Text feature [per] present in test data point [True]
354 Text feature [even] present in test data point [True]
355 Text feature [acid] present in test data point [True]
356 Text feature [serum] present in test data point [True]
360 Text feature [discovery] present in test data point [True]
362 Text feature [derived] present in test data point [True]
365 Text feature [induce] present in test data point [True]
370 Text feature [her2] present in test data point [True]
371 Text feature [erbb2] present in test data point [True]
```

```
373 Text feature [death] present in test data point [True]
376 Text feature [included] present in test data point [True]
377 Text feature [download] present in test data point [True]
382 Text feature [membrane] present in test data point [True]
383 Text feature [required] present in test data point [True]
385 Text feature [factors] present in test data point [True]
394 Text feature [acids] present in test data point [True]
396 Text feature [helix] present in test data point [True]
398 Text feature [05] present in test data point [True]
402 Text feature [recently] present in test data point [True]
405 Text feature [status] present in test data point [True]
406 Text feature [blot] present in test data point [True]
414 Text feature [pocket] present in test data point [True]
415 Text feature [affected] present in test data point [True]
420 Text feature [interestingly] present in test data point [True]
425 Text feature [followed] present in test data point [True]
427 Text feature [conformation] present in test data point [True]
433 Text feature [51] present in test data point [True]
435 Text feature [tyrosine] present in test data point [True]
443 Text feature [suppressor] present in test data point [True]
445 Text feature [pathway] present in test data point [True]
450 Text feature [sensitive] present in test data point [True]
456 Text feature [akt1] present in test data point [True]
459 Text feature [size] present in test data point [True]
461 Text feature [wt] present in test data point [True]
462 Text feature [efficacy] present in test data point [True]
466 Text feature [recombinant] present in test data point [True]
469 Text feature [variants] present in test data point [True]
470 Text feature [72] present in test data point [True]
476 Text feature [high] present in test data point [True]
477 Text feature [cells] present in test data point [True]
479 Text feature [controls] present in test data point [True]
482 Text feature [unable] present in test data point [True]
483 Text feature [ligand] present in test data point [True]
484 Text feature [control] present in test data point [True]
492 Text feature [lack] present in test data point [True]
494 Text feature [24] present in test data point [True]
495 Text feature [pi3k] present in test data point [True]
Out of the top  500  features  139 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

[ ]:
```
test_point_index = 56
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
```

```
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.2043 0.0714 0.0062 0.0055 0.0592 0.0173
0.6301 0.0053 0.0007]]
Actual Class : 2
--------------------------------------------------
20 Text feature [activation] present in test data point [True]
21 Text feature [constitutive] present in test data point [True]
30 Text feature [activated] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
52 Text feature [activate] present in test data point [True]
62 Text feature [important] present in test data point [True]
66 Text feature [codon] present in test data point [True]
81 Text feature [increased] present in test data point [True]
93 Text feature [advanced] present in test data point [True]
95 Text feature [mapk] present in test data point [True]
103 Text feature [signaling] present in test data point [True]
107 Text feature [analyses] present in test data point [True]
114 Text feature [showing] present in test data point [True]
120 Text feature [second] present in test data point [True]
122 Text feature [adenocarcinoma] present in test data point [True]
124 Text feature [mechanisms] present in test data point [True]
126 Text feature [extracellular] present in test data point [True]
134 Text feature [occur] present in test data point [True]
139 Text feature [cancers] present in test data point [True]
155 Text feature [defective] present in test data point [True]
158 Text feature [eight] present in test data point [True]
162 Text feature [different] present in test data point [True]
165 Text feature [activating] present in test data point [True]
171 Text feature [pathways] present in test data point [True]
181 Text feature [individuals] present in test data point [True]
192 Text feature [presented] present in test data point [True]
194 Text feature [investigated] present in test data point [True]
205 Text feature [predicted] present in test data point [True]
218 Text feature [missense] present in test data point [True]
225 Text feature [structural] present in test data point [True]
229 Text feature [inhibitor] present in test data point [True]
232 Text feature [dimerization] present in test data point [True]
236 Text feature [loss] present in test data point [True]
241 Text feature [presence] present in test data point [True]
```

```
242 Text feature [bp] present in test data point [True]
244 Text feature [splice] present in test data point [True]
249 Text feature [increase] present in test data point [True]
252 Text feature [positive] present in test data point [True]
254 Text feature [express] present in test data point [True]
261 Text feature [leading] present in test data point [True]
264 Text feature [function] present in test data point [True]
274 Text feature [approximately] present in test data point [True]
278 Text feature [due] present in test data point [True]
313 Text feature [unknown] present in test data point [True]
331 Text feature [provided] present in test data point [True]
336 Text feature [based] present in test data point [True]
345 Text feature [highly] present in test data point [True]
360 Text feature [discovery] present in test data point [True]
362 Text feature [derived] present in test data point [True]
388 Text feature [epithelial] present in test data point [True]
394 Text feature [acids] present in test data point [True]
405 Text feature [status] present in test data point [True]
407 Text feature [2006] present in test data point [True]
435 Text feature [tyrosine] present in test data point [True]
438 Text feature [performed] present in test data point [True]
445 Text feature [pathway] present in test data point [True]
469 Text feature [variants] present in test data point [True]
477 Text feature [cells] present in test data point [True]
483 Text feature [ligand] present in test data point [True]
495 Text feature [pi3k] present in test data point [True]
Out of the top  500  features  60 are present in query point
```

4.3.2. Without Class balancing

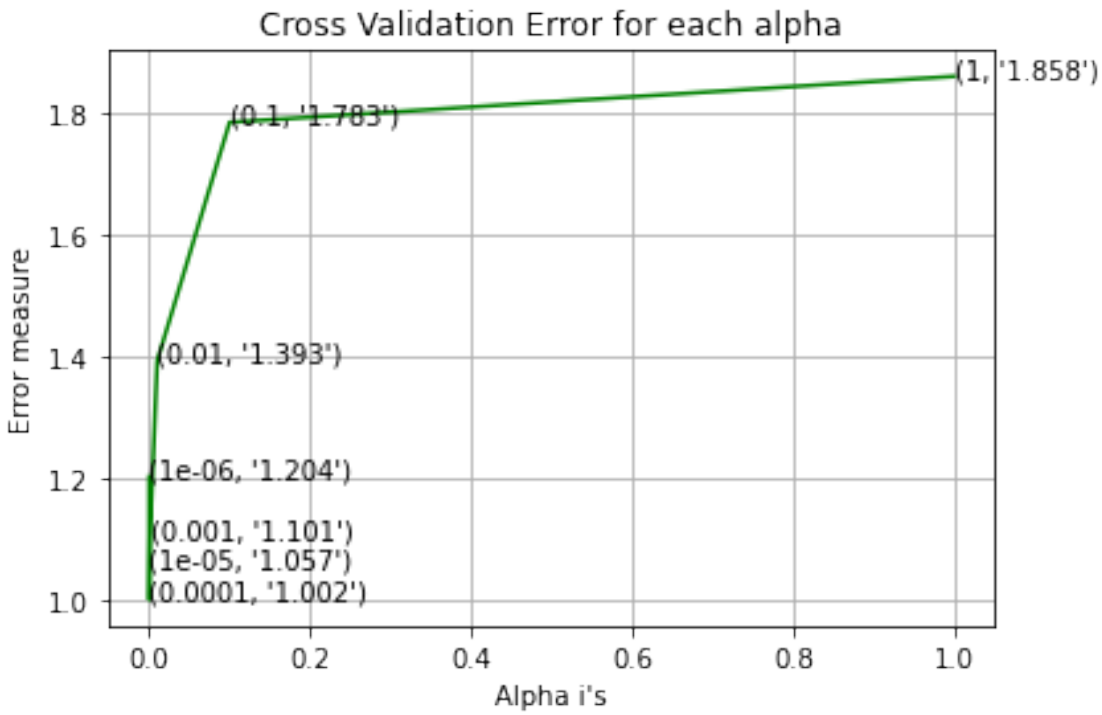
4.3.2.1. Hyper paramter tuning

[ ]:
```
# 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
```

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_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(cv_log_error_array)
```

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.2037264791785944
for alpha = 1e-05
Log Loss : 1.0567157192235035
for alpha = 0.0001
Log Loss : 1.0024522942728542
for alpha = 0.001
Log Loss : 1.1013189235322176
for alpha = 0.01
Log Loss : 1.3929508310625587
for alpha = 0.1
Log Loss : 1.783029441373136
for alpha = 1
Log Loss : 1.8581859612461096
```



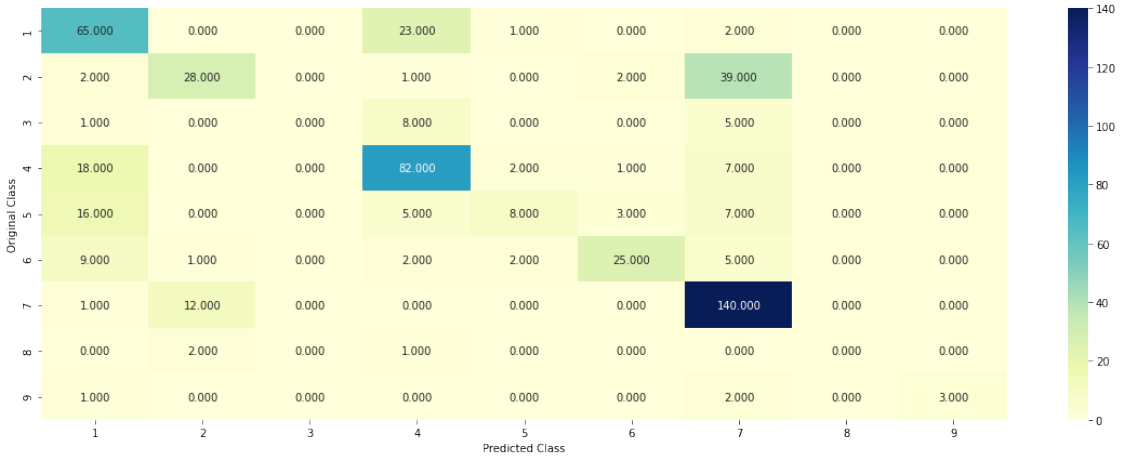

```
For values of best alpha =  0.0001 The train log loss is: 0.5371554557467861
For values of best alpha =  0.0001 The cross validation log loss is:
1.0024522942728542
For values of best alpha =  0.0001 The test log loss is: 1.005977957480678
```

4.3.2.2. Testing model with best hyper parameters

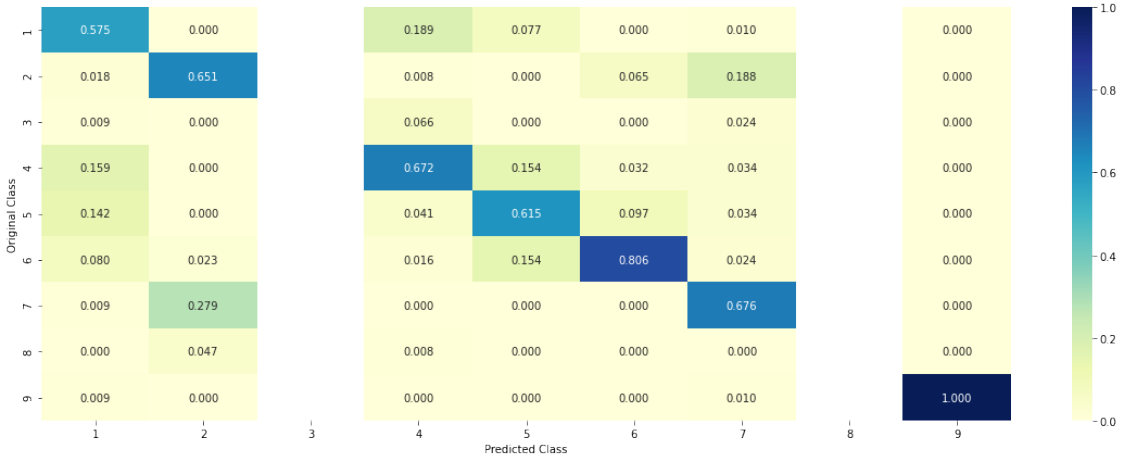
[ ]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
#------------------------------

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,␣
 ↪cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.0024522942728542
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

4.3.2.3. Feature Importance, Correctly Classified point

[ ]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 7
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0096 0.16   0.0112 0.0241 0.098  0.0205
0.6711 0.0046 0.0009]]
Actual Class : 7
--------------------------------------------------
12 Text feature [oncogene] present in test data point [True]
19 Text feature [activation] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
35 Text feature [activated] present in test data point [True]
36 Text feature [phospho] present in test data point [True]
37 Text feature [codon] present in test data point [True]
38 Text feature [none] present in test data point [True]
44 Text feature [inhibited] present in test data point [True]
```

46 Text feature [increased] present in test data point [True]
48 Text feature [important] present in test data point [True]
51 Text feature [activate] present in test data point [True]
56 Text feature [group] present in test data point [True]
61 Text feature [phosphorylated] present in test data point [True]
68 Text feature [ras] present in test data point [True]
71 Text feature [downstream] present in test data point [True]
73 Text feature [mek] present in test data point [True]
74 Text feature [substrate] present in test data point [True]
76 Text feature [advanced] present in test data point [True]
82 Text feature [enhanced] present in test data point [True]
95 Text feature [oncogenic] present in test data point [True]
96 Text feature [able] present in test data point [True]
100 Text feature [reduced] present in test data point [True]
106 Text feature [second] present in test data point [True]
109 Text feature [s3] present in test data point [True]
111 Text feature [occur] present in test data point [True]
116 Text feature [showing] present in test data point [True]
118 Text feature [3b] present in test data point [True]
130 Text feature [cancers] present in test data point [True]
136 Text feature [extracellular] present in test data point [True]
143 Text feature [reduction] present in test data point [True]
144 Text feature [mapk] present in test data point [True]
153 Text feature [3t3] present in test data point [True]
156 Text feature [38] present in test data point [True]
165 Text feature [approximately] present in test data point [True]
170 Text feature [mechanisms] present in test data point [True]
171 Text feature [dimerization] present in test data point [True]
173 Text feature [different] present in test data point [True]
174 Text feature [signaling] present in test data point [True]
176 Text feature [factor] present in test data point [True]
177 Text feature [surface] present in test data point [True]
178 Text feature [transformed] present in test data point [True]
181 Text feature [conserved] present in test data point [True]
185 Text feature [eight] present in test data point [True]
197 Text feature [defective] present in test data point [True]
198 Text feature [increase] present in test data point [True]
202 Text feature [activating] present in test data point [True]
203 Text feature [structural] present in test data point [True]
205 Text feature [signalling] present in test data point [True]
212 Text feature [bp] present in test data point [True]
215 Text feature [akt] present in test data point [True]
220 Text feature [versus] present in test data point [True]
224 Text feature [concentrations] present in test data point [True]
225 Text feature [lung] present in test data point [True]
231 Text feature [individuals] present in test data point [True]
233 Text feature [predicted] present in test data point [True]
236 Text feature [presence] present in test data point [True]

244 Text feature [purified] present in test data point [True]
246 Text feature [positive] present in test data point [True]
247 Text feature [inhibitor] present in test data point [True]
249 Text feature [sirna] present in test data point [True]
250 Text feature [due] present in test data point [True]
252 Text feature [discovery] present in test data point [True]
253 Text feature [strongly] present in test data point [True]
256 Text feature [60] present in test data point [True]
261 Text feature [hours] present in test data point [True]
262 Text feature [splice] present in test data point [True]
269 Text feature [express] present in test data point [True]
273 Text feature [significance] present in test data point [True]
276 Text feature [recently] present in test data point [True]
277 Text feature [provided] present in test data point [True]
280 Text feature [acid] present in test data point [True]
285 Text feature [induce] present in test data point [True]
290 Text feature [dose] present in test data point [True]
293 Text feature [31] present in test data point [True]
296 Text feature [pathways] present in test data point [True]
302 Text feature [unknown] present in test data point [True]
305 Text feature [leading] present in test data point [True]
307 Text feature [pattern] present in test data point [True]
308 Text feature [per] present in test data point [True]
310 Text feature [phosphorylation] present in test data point [True]
325 Text feature [amplified] present in test data point [True]
326 Text feature [gtp] present in test data point [True]
329 Text feature [membrane] present in test data point [True]
334 Text feature [atp] present in test data point [True]
337 Text feature [followed] present in test data point [True]
340 Text feature [function] present in test data point [True]
341 Text feature [leads] present in test data point [True]
343 Text feature [missense] present in test data point [True]
346 Text feature [terminal] present in test data point [True]
352 Text feature [stably] present in test data point [True]
356 Text feature [derived] present in test data point [True]
372 Text feature [affect] present in test data point [True]
373 Text feature [high] present in test data point [True]
377 Text feature [expressing] present in test data point [True]
379 Text feature [highly] present in test data point [True]
380 Text feature [loss] present in test data point [True]
382 Text feature [blot] present in test data point [True]
384 Text feature [example] present in test data point [True]
385 Text feature [based] present in test data point [True]
387 Text feature [hydrogen] present in test data point [True]
391 Text feature [unable] present in test data point [True]
395 Text feature [size] present in test data point [True]
409 Text feature [2002] present in test data point [True]
412 Text feature [even] present in test data point [True]

```
426 Text feature [wt] present in test data point [True]
429 Text feature [controls] present in test data point [True]
434 Text feature [helix] present in test data point [True]
435 Text feature [promoter] present in test data point [True]
437 Text feature [required] present in test data point [True]
443 Text feature [acids] present in test data point [True]
447 Text feature [24] present in test data point [True]
448 Text feature [distinct] present in test data point [True]
450 Text feature [factors] present in test data point [True]
452 Text feature [epithelial] present in test data point [True]
453 Text feature [transcription] present in test data point [True]
455 Text feature [affected] present in test data point [True]
456 Text feature [lack] present in test data point [True]
457 Text feature [download] present in test data point [True]
459 Text feature [expected] present in test data point [True]
461 Text feature [serum] present in test data point [True]
464 Text feature [51] present in test data point [True]
468 Text feature [interestingly] present in test data point [True]
469 Text feature [05] present in test data point [True]
476 Text feature [30] present in test data point [True]
477 Text feature [status] present in test data point [True]
482 Text feature [performed] present in test data point [True]
494 Text feature [sd] present in test data point [True]
Out of the top  500  features  127 are present in query point
```

4.3.2.4. Feature Importance, InCorrectly Classified point

Indented block

[ ]:
```
test_point_index = 56
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.870e-01 6.210e-02 3.000e-03 4.400e-03
3.870e-02 1.360e-02 6.867e-01
  4.200e-03 2.000e-04]]
Actual Class : 2
```

```
-------------------------------------------------
19 Text feature [activation] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
35 Text feature [activated] present in test data point [True]
37 Text feature [codon] present in test data point [True]
46 Text feature [increased] present in test data point [True]
48 Text feature [important] present in test data point [True]
51 Text feature [activate] present in test data point [True]
71 Text feature [downstream] present in test data point [True]
76 Text feature [advanced] present in test data point [True]
90 Text feature [analyses] present in test data point [True]
106 Text feature [second] present in test data point [True]
111 Text feature [occur] present in test data point [True]
114 Text feature [adenocarcinoma] present in test data point [True]
116 Text feature [showing] present in test data point [True]
130 Text feature [cancers] present in test data point [True]
136 Text feature [extracellular] present in test data point [True]
144 Text feature [mapk] present in test data point [True]
155 Text feature [presented] present in test data point [True]
163 Text feature [investigated] present in test data point [True]
165 Text feature [approximately] present in test data point [True]
170 Text feature [mechanisms] present in test data point [True]
171 Text feature [dimerization] present in test data point [True]
173 Text feature [different] present in test data point [True]
174 Text feature [signaling] present in test data point [True]
185 Text feature [eight] present in test data point [True]
197 Text feature [defective] present in test data point [True]
198 Text feature [increase] present in test data point [True]
202 Text feature [activating] present in test data point [True]
203 Text feature [structural] present in test data point [True]
212 Text feature [bp] present in test data point [True]
231 Text feature [individuals] present in test data point [True]
233 Text feature [predicted] present in test data point [True]
236 Text feature [presence] present in test data point [True]
246 Text feature [positive] present in test data point [True]
247 Text feature [inhibitor] present in test data point [True]
250 Text feature [due] present in test data point [True]
252 Text feature [discovery] present in test data point [True]
262 Text feature [splice] present in test data point [True]
269 Text feature [express] present in test data point [True]
277 Text feature [provided] present in test data point [True]
296 Text feature [pathways] present in test data point [True]
302 Text feature [unknown] present in test data point [True]
305 Text feature [leading] present in test data point [True]
340 Text feature [function] present in test data point [True]
343 Text feature [missense] present in test data point [True]
356 Text feature [derived] present in test data point [True]
379 Text feature [highly] present in test data point [True]
```

```
380 Text feature [loss] present in test data point [True]
384 Text feature [example] present in test data point [True]
385 Text feature [based] present in test data point [True]
409 Text feature [2002] present in test data point [True]
438 Text feature [ligand] present in test data point [True]
443 Text feature [acids] present in test data point [True]
444 Text feature [syndrome] present in test data point [True]
448 Text feature [distinct] present in test data point [True]
452 Text feature [epithelial] present in test data point [True]
476 Text feature [30] present in test data point [True]
477 Text feature [status] present in test data point [True]
478 Text feature [therapy] present in test data point [True]
482 Text feature [performed] present in test data point [True]
Out of the top  500  features  60 are present in query point
```

4.4. Linear Support Vector Machines

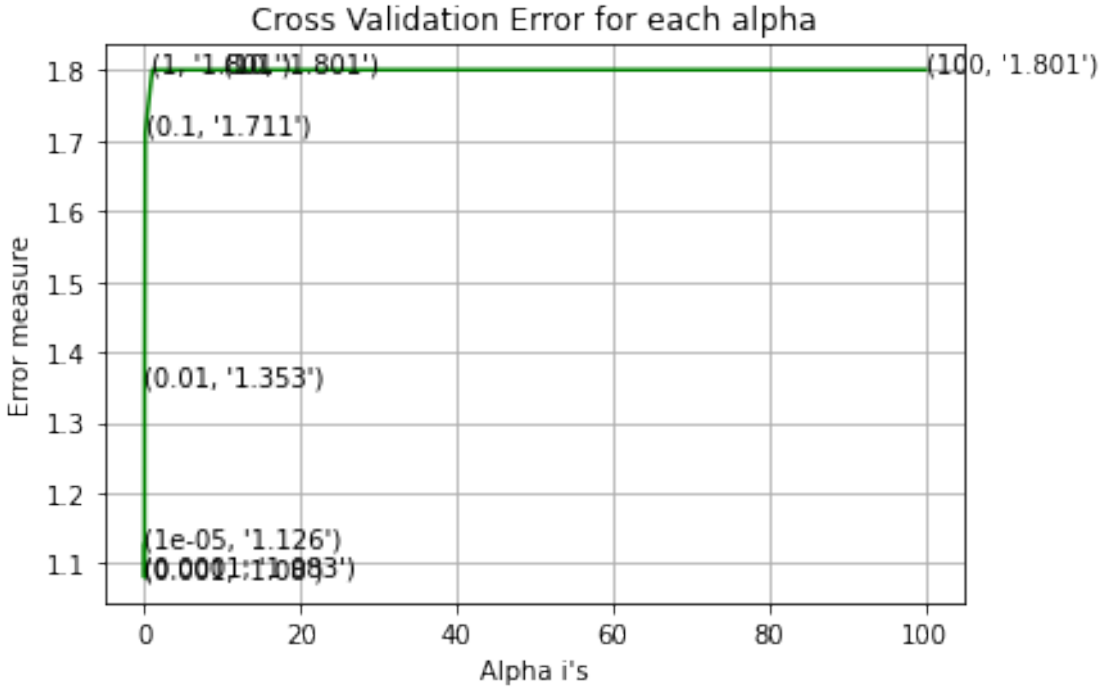
4.4.1. Hyper paramter tuning

[ ]:
```
# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1,␣
 ↪decision_function_shape=ovr, random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# ---------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/mathematical-derivation-copy-8/
# ---------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
```

```
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2',␣
 ↪loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_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(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
```

```
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.1259461424937938
for C = 0.0001
Log Loss : 1.0826393482827097
for C = 0.001
Log Loss : 1.0795388461923687
for C = 0.01
Log Loss : 1.3533213958159915
for C = 0.1
Log Loss : 1.710713908854527
for C = 1
Log Loss : 1.8008870123119656
for C = 10
Log Loss : 1.8008865342639586
for C = 100
Log Loss : 1.8009016783892733
```



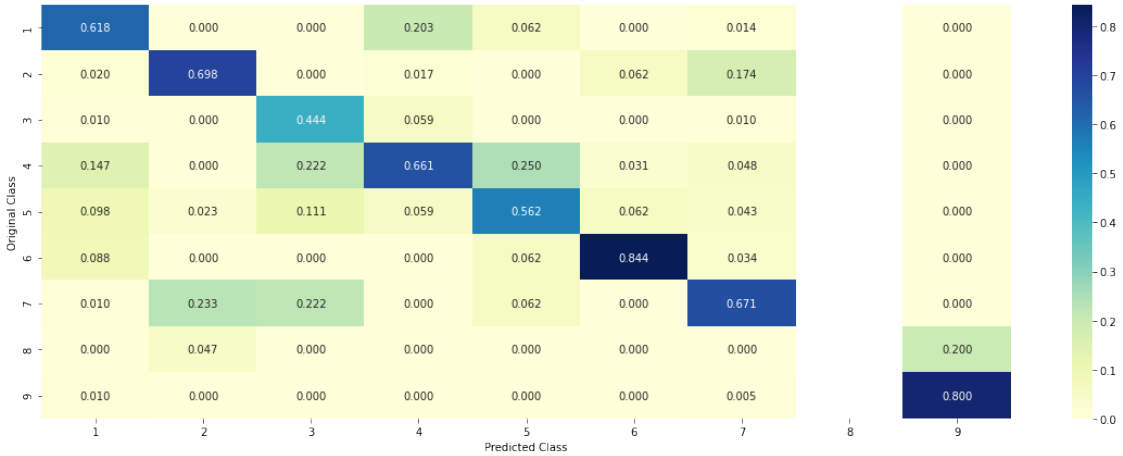

```
For values of best alpha =  0.001 The train log loss is: 0.7966409048252387
For values of best alpha =  0.001 The cross validation log loss is:
1.0795388461923687
For values of best alpha =  0.001 The test log loss is: 1.1097288358929833
```

4.4.2. Testing model with best hyper parameters

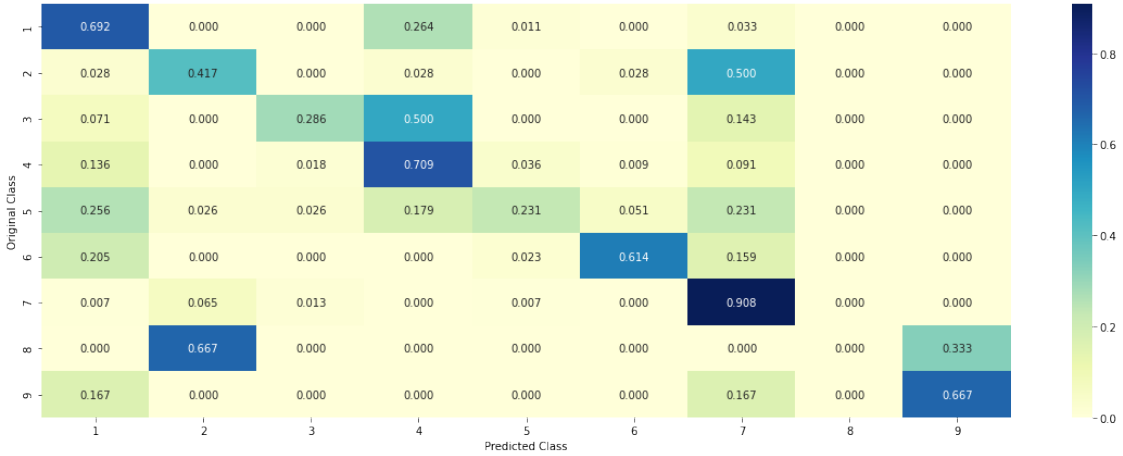
[ ]:
```
# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ----------------------------------
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1,␣
 ↪decision_function_shape=ovr, random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# ----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/mathematical-derivation-copy-8/
# ----------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True,␣
 ↪class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',␣
 ↪random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding,␣
 ↪train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.0795388461923687
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

[ ]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',␣
↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 7
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0815 0.1278 0.0304 0.1417 0.0818 0.0413
0.4813 0.0079 0.0063]]
Actual Class : 7
--------------------------------------------------
22 Text feature [constitutive] present in test data point [True]
24 Text feature [activation] present in test data point [True]
30 Text feature [activate] present in test data point [True]
33 Text feature [substrate] present in test data point [True]
34 Text feature [oncogene] present in test data point [True]
37 Text feature [activated] present in test data point [True]
41 Text feature [akt] present in test data point [True]
47 Text feature [codon] present in test data point [True]
48 Text feature [mapk] present in test data point [True]
49 Text feature [group] present in test data point [True]
53 Text feature [38] present in test data point [True]
56 Text feature [advanced] present in test data point [True]
57 Text feature [3t3] present in test data point [True]
61 Text feature [downstream] present in test data point [True]
62 Text feature [inhibited] present in test data point [True]
69 Text feature [showing] present in test data point [True]
70 Text feature [phosphorylated] present in test data point [True]
71 Text feature [signaling] present in test data point [True]
72 Text feature [cancers] present in test data point [True]
73 Text feature [oncogenic] present in test data point [True]
79 Text feature [05] present in test data point [True]
80 Text feature [ras] present in test data point [True]
83 Text feature [60] present in test data point [True]
84 Text feature [individuals] present in test data point [True]
86 Text feature [activating] present in test data point [True]
93 Text feature [reduced] present in test data point [True]
94 Text feature [none] present in test data point [True]
96 Text feature [increased] present in test data point [True]
97 Text feature [versus] present in test data point [True]
98 Text feature [important] present in test data point [True]
99 Text feature [nsclc] present in test data point [True]
101 Text feature [occur] present in test data point [True]
```

```
114 Text feature [3b] present in test data point [True]
115 Text feature [pattern] present in test data point [True]
116 Text feature [enhanced] present in test data point [True]
117 Text feature [phospho] present in test data point [True]
118 Text feature [reduction] present in test data point [True]
119 Text feature [purified] present in test data point [True]
120 Text feature [second] present in test data point [True]
121 Text feature [factors] present in test data point [True]
122 Text feature [phosphorylation] present in test data point [True]
132 Text feature [inhibitor] present in test data point [True]
135 Text feature [loss] present in test data point [True]
138 Text feature [membrane] present in test data point [True]
258 Text feature [lung] present in test data point [True]
260 Text feature [wt] present in test data point [True]
261 Text feature [s3] present in test data point [True]
262 Text feature [positive] present in test data point [True]
264 Text feature [conformation] present in test data point [True]
267 Text feature [eight] present in test data point [True]
330 Text feature [constitutively] present in test data point [True]
335 Text feature [due] present in test data point [True]
336 Text feature [transformed] present in test data point [True]
337 Text feature [predicted] present in test data point [True]
338 Text feature [conserved] present in test data point [True]
339 Text feature [acid] present in test data point [True]
340 Text feature [200] present in test data point [True]
341 Text feature [altered] present in test data point [True]
342 Text feature [terminal] present in test data point [True]
343 Text feature [weeks] present in test data point [True]
344 Text feature [stably] present in test data point [True]
345 Text feature [extracellular] present in test data point [True]
346 Text feature [structural] present in test data point [True]
347 Text feature [discovery] present in test data point [True]
349 Text feature [factor] present in test data point [True]
350 Text feature [mechanisms] present in test data point [True]
351 Text feature [42] present in test data point [True]
352 Text feature [express] present in test data point [True]
353 Text feature [promoter] present in test data point [True]
354 Text feature [sensitive] present in test data point [True]
355 Text feature [concentrations] present in test data point [True]
384 Text feature [affected] present in test data point [True]
385 Text feature [expressing] present in test data point [True]
387 Text feature [active] present in test data point [True]
388 Text feature [included] present in test data point [True]
389 Text feature [hours] present in test data point [True]
390 Text feature [approximately] present in test data point [True]
391 Text feature [highly] present in test data point [True]
392 Text feature [presence] present in test data point [True]
393 Text feature [mek] present in test data point [True]
```

```
394 Text feature [defective] present in test data point [True]
395 Text feature [51] present in test data point [True]
398 Text feature [derived] present in test data point [True]
399 Text feature [sirna] present in test data point [True]
417 Text feature [24] present in test data point [True]
419 Text feature [pocket] present in test data point [True]
420 Text feature [flag] present in test data point [True]
421 Text feature [blot] present in test data point [True]
422 Text feature [lead] present in test data point [True]
423 Text feature [atp] present in test data point [True]
424 Text feature [provided] present in test data point [True]
426 Text feature [pathways] present in test data point [True]
427 Text feature [function] present in test data point [True]
428 Text feature [increase] present in test data point [True]
433 Text feature [different] present in test data point [True]
435 Text feature [strongly] present in test data point [True]
436 Text feature [31] present in test data point [True]
460 Text feature [recently] present in test data point [True]
461 Text feature [report] present in test data point [True]
463 Text feature [cells] present in test data point [True]
464 Text feature [helix] present in test data point [True]
468 Text feature [malignant] present in test data point [True]
469 Text feature [95] present in test data point [True]
471 Text feature [surface] present in test data point [True]
472 Text feature [erk] present in test data point [True]
474 Text feature [even] present in test data point [True]
476 Text feature [ng] present in test data point [True]
478 Text feature [fusion] present in test data point [True]
479 Text feature [evidence] present in test data point [True]
484 Text feature [leading] present in test data point [True]
485 Text feature [invitrogen] present in test data point [True]
490 Text feature [splice] present in test data point [True]
492 Text feature [epithelial] present in test data point [True]
493 Text feature [pathway] present in test data point [True]
494 Text feature [per] present in test data point [True]
Out of the top  500  features  115 are present in query point
```

4.3.3.2. For Incorrectly classified point

[ ]:
```
test_point_index = 56
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.2346 0.1249 0.0283 0.0627 0.0389 0.0396
0.4603 0.0076 0.0029]]
Actual Class : 2
--------------------------------------------------
22 Text feature [constitutive] present in test data point [True]
24 Text feature [activation] present in test data point [True]
30 Text feature [activate] present in test data point [True]
37 Text feature [activated] present in test data point [True]
47 Text feature [codon] present in test data point [True]
48 Text feature [mapk] present in test data point [True]
56 Text feature [advanced] present in test data point [True]
61 Text feature [downstream] present in test data point [True]
64 Text feature [adenocarcinoma] present in test data point [True]
69 Text feature [showing] present in test data point [True]
71 Text feature [signaling] present in test data point [True]
72 Text feature [cancers] present in test data point [True]
84 Text feature [individuals] present in test data point [True]
86 Text feature [activating] present in test data point [True]
95 Text feature [ligand] present in test data point [True]
96 Text feature [increased] present in test data point [True]
98 Text feature [important] present in test data point [True]
101 Text feature [occur] present in test data point [True]
120 Text feature [second] present in test data point [True]
132 Text feature [inhibitor] present in test data point [True]
135 Text feature [loss] present in test data point [True]
136 Text feature [pi3k] present in test data point [True]
137 Text feature [syndrome] present in test data point [True]
262 Text feature [positive] present in test data point [True]
265 Text feature [analyses] present in test data point [True]
267 Text feature [eight] present in test data point [True]
333 Text feature [presented] present in test data point [True]
335 Text feature [due] present in test data point [True]
337 Text feature [predicted] present in test data point [True]
340 Text feature [200] present in test data point [True]
345 Text feature [extracellular] present in test data point [True]
346 Text feature [structural] present in test data point [True]
347 Text feature [discovery] present in test data point [True]
350 Text feature [mechanisms] present in test data point [True]
352 Text feature [express] present in test data point [True]
390 Text feature [approximately] present in test data point [True]
391 Text feature [highly] present in test data point [True]
```

```
392 Text feature [presence] present in test data point [True]
394 Text feature [defective] present in test data point [True]
398 Text feature [derived] present in test data point [True]
424 Text feature [provided] present in test data point [True]
426 Text feature [pathways] present in test data point [True]
427 Text feature [function] present in test data point [True]
428 Text feature [increase] present in test data point [True]
433 Text feature [different] present in test data point [True]
463 Text feature [cells] present in test data point [True]
469 Text feature [95] present in test data point [True]
477 Text feature [selection] present in test data point [True]
484 Text feature [leading] present in test data point [True]
490 Text feature [splice] present in test data point [True]
492 Text feature [epithelial] present in test data point [True]
493 Text feature [pathway] present in test data point [True]
495 Text feature [cause] present in test data point [True]
Out of the top  500  features  53 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

[ ]:
```
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# predict_proba (X)        Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# ---------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/random-forest-and-their-construction-2/
# ---------------------------------
```

```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',␣
 ↪max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),␣
 ↪(features[i],cv_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(cv_log_error_array)
```

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
 ↪n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train␣
 ↪log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross␣
 ↪validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test␣
 ↪log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.197403520685911
for n_estimators = 100 and max depth =  10
Log Loss : 1.2112328504277914
for n_estimators = 200 and max depth =  5
Log Loss : 1.1893794223846492
for n_estimators = 200 and max depth =  10
Log Loss : 1.2051813557377458
for n_estimators = 500 and max depth =  5
Log Loss : 1.1839846718187392
for n_estimators = 500 and max depth =  10
Log Loss : 1.2042931299203914
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1860074732604928
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2013040522632272
for n_estimators = 2000 and max depth =  5
Log Loss : 1.183494854408345
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2004764303261013
For values of best estimator =  2000 The train log loss is: 0.859544978992291
For values of best estimator =  2000 The cross validation log loss is:
1.183494854408345
For values of best estimator =  2000 The test log loss is: 1.1822825404818955
```

4.5.2. Testing model with best hyper parameters (One Hot Encoding)

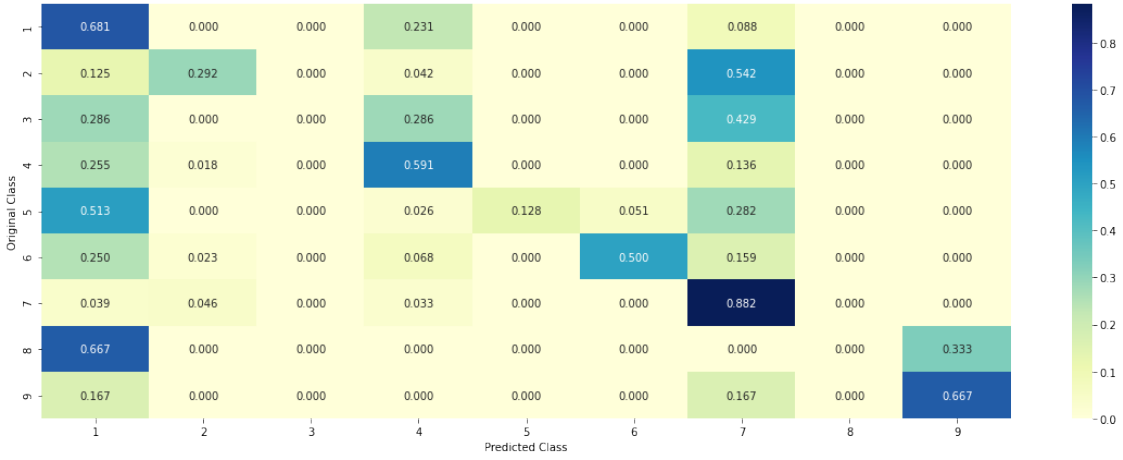
[ ]:
```
# ------------------------------
# default parameters
```

```
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# predict_proba (X)        Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/random-forest-and-their-construction-2/
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
 ↪n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding,␣
 ↪train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.183494854408345
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```


```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

[ ]:
```
clf = RandomForestClassifier(n_estimators=2000, criterion='gini', max_depth=5,␣
↪random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
```

[ ]:
```
CalibratedClassifierCV(base_estimator=RandomForestClassifier(bootstrap=True,
                                                             ccp_alpha=0.0,
                                                             class_weight=None,
                                                             criterion='gini',
                                                             max_depth=5,
    max_features='auto',
    max_leaf_nodes=None,
                                                             max_samples=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
                                                             min_samples_leaf=1,
    min_samples_split=2,
    min_weight_fraction_leaf=0.0,
                                                             n_estimators=2000,
                                                             n_jobs=-1,
                                                             oob_score=False,
                                                             random_state=42,
                                                             verbose=0,
                                                             warm_start=False),
                       cv=None, method='sigmoid')
```

[ ]:
```
test_point_index = 7
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0304 0.1023 0.0237 0.0323 0.0476 0.0355
0.7178 0.0086 0.0018]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
4 Text feature [tyrosine] present in test data point [True]
5 Text feature [function] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
```

```
9 Text feature [activated] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [loss] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [missense] present in test data point [True]
14 Text feature [inhibitor] present in test data point [True]
15 Text feature [erk] present in test data point [True]
16 Text feature [akt] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
21 Text feature [growth] present in test data point [True]
22 Text feature [variants] present in test data point [True]
24 Text feature [receptor] present in test data point [True]
25 Text feature [yeast] present in test data point [True]
26 Text feature [inhibition] present in test data point [True]
28 Text feature [cell] present in test data point [True]
32 Text feature [constitutively] present in test data point [True]
33 Text feature [activate] present in test data point [True]
34 Text feature [protein] present in test data point [True]
35 Text feature [classified] present in test data point [True]
37 Text feature [inhibited] present in test data point [True]
40 Text feature [cells] present in test data point [True]
42 Text feature [kinases] present in test data point [True]
43 Text feature [transforming] present in test data point [True]
44 Text feature [resistance] present in test data point [True]
45 Text feature [expression] present in test data point [True]
51 Text feature [3t3] present in test data point [True]
52 Text feature [functional] present in test data point [True]
53 Text feature [drug] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
56 Text feature [treated] present in test data point [True]
57 Text feature [advanced] present in test data point [True]
58 Text feature [downstream] present in test data point [True]
60 Text feature [tagged] present in test data point [True]
61 Text feature [response] present in test data point [True]
62 Text feature [defective] present in test data point [True]
63 Text feature [mapk] present in test data point [True]
66 Text feature [oncogene] present in test data point [True]
69 Text feature [mek] present in test data point [True]
70 Text feature [proliferation] present in test data point [True]
71 Text feature [nuclear] present in test data point [True]
72 Text feature [ring] present in test data point [True]
73 Text feature [ras] present in test data point [True]
74 Text feature [extracellular] present in test data point [True]
76 Text feature [ovarian] present in test data point [True]
80 Text feature [expressing] present in test data point [True]
81 Text feature [functions] present in test data point [True]
82 Text feature [nsclc] present in test data point [True]
85 Text feature [phosphorylated] present in test data point [True]
```

```
88 Text feature [dna] present in test data point [True]
89 Text feature [conserved] present in test data point [True]
91 Text feature [phospho] present in test data point [True]
92 Text feature [activity] present in test data point [True]
93 Text feature [expected] present in test data point [True]
95 Text feature [lines] present in test data point [True]
97 Text feature [information] present in test data point [True]
98 Text feature [type] present in test data point [True]
99 Text feature [potential] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

4.5.3.2. Inorrectly Classified point

[ ]:
```
test_point_index = 56
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1324 0.2392 0.0304 0.117  0.077  0.0753 0.315
0.0082 0.0056]]
Actuall Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
4 Text feature [tyrosine] present in test data point [True]
5 Text feature [function] present in test data point [True]
9 Text feature [activated] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [loss] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [missense] present in test data point [True]
14 Text feature [inhibitor] present in test data point [True]
18 Text feature [therapy] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
22 Text feature [variants] present in test data point [True]
24 Text feature [receptor] present in test data point [True]
26 Text feature [inhibition] present in test data point [True]
```

```
27 Text feature [trials] present in test data point [True]
28 Text feature [cell] present in test data point [True]
29 Text feature [pathogenic] present in test data point [True]
33 Text feature [activate] present in test data point [True]
34 Text feature [protein] present in test data point [True]
40 Text feature [cells] present in test data point [True]
41 Text feature [repair] present in test data point [True]
42 Text feature [kinases] present in test data point [True]
45 Text feature [expression] present in test data point [True]
52 Text feature [functional] present in test data point [True]
57 Text feature [advanced] present in test data point [True]
58 Text feature [downstream] present in test data point [True]
62 Text feature [defective] present in test data point [True]
63 Text feature [mapk] present in test data point [True]
74 Text feature [extracellular] present in test data point [True]
77 Text feature [patients] present in test data point [True]
78 Text feature [survival] present in test data point [True]
87 Text feature [variant] present in test data point [True]
88 Text feature [dna] present in test data point [True]
92 Text feature [activity] present in test data point [True]
94 Text feature [clinical] present in test data point [True]
95 Text feature [lines] present in test data point [True]
98 Text feature [type] present in test data point [True]
Out of the top  100  features  39 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

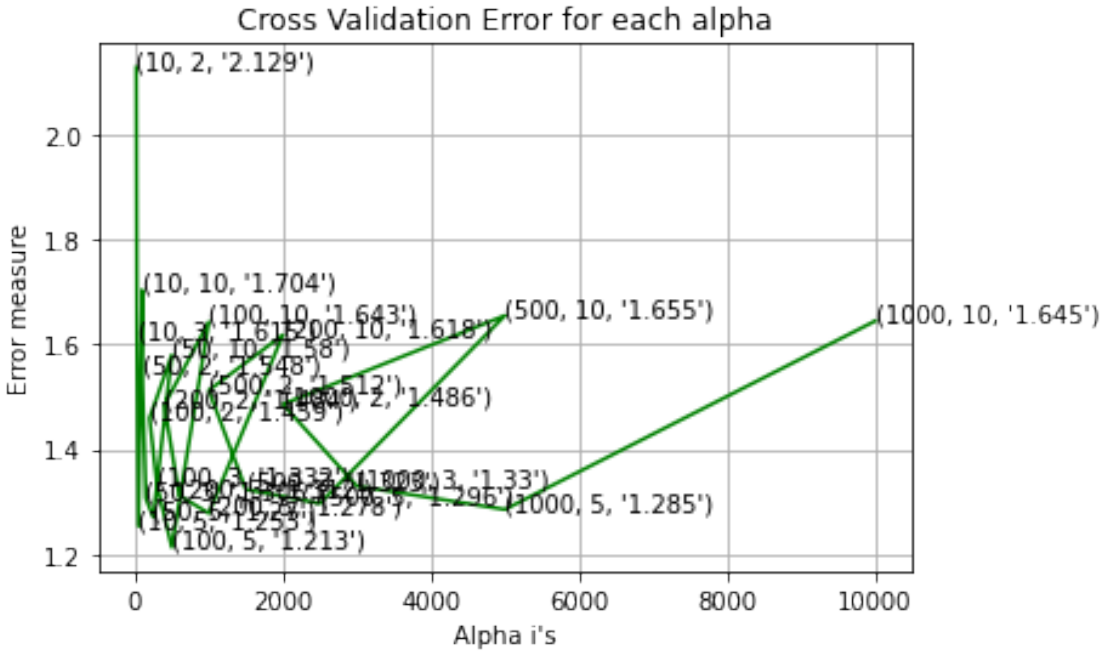
[ ]:
```
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# predict_proba (X)        Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# ---------------------------------
```

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/random-forest-and-their-construction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',␣
 ↪max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

# fig, ax = plt.subplots()
# features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
# ax.plot(features, cv_log_error_array,c='g')
# for i, txt in enumerate(np.round(cv_log_error_array,3)):
#     ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),␣
 ↪(features[i],cv_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(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,␣
 ↪n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log␣
 ↪loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross␣
 ↪validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log␣
 ↪loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.128922475672198
for n_estimators = 10 and max depth =  3
Log Loss : 1.6145465524062061
for n_estimators = 10 and max depth =  5
Log Loss : 1.252910926734783
for n_estimators = 10 and max depth =  10
Log Loss : 1.7040160476162232
for n_estimators = 50 and max depth =  2
Log Loss : 1.548187334762907
for n_estimators = 50 and max depth =  3
Log Loss : 1.3060649453309807
for n_estimators = 50 and max depth =  5
Log Loss : 1.2704593007008433
for n_estimators = 50 and max depth =  10
Log Loss : 1.5797084062299178
for n_estimators = 100 and max depth =  2
Log Loss : 1.4591905939894119
for n_estimators = 100 and max depth =  3
Log Loss : 1.332184261382671
for n_estimators = 100 and max depth =  5
Log Loss : 1.2127621969794387
for n_estimators = 100 and max depth =  10
Log Loss : 1.6429028164120019
```

```
for n_estimators = 200 and max depth =  2
Log Loss : 1.4842932666255249
for n_estimators = 200 and max depth =  3
Log Loss : 1.3116502456762156
for n_estimators = 200 and max depth =  5
Log Loss : 1.277575150533822
for n_estimators = 200 and max depth =  10
Log Loss : 1.6180321978550531
for n_estimators = 500 and max depth =  2
Log Loss : 1.5123621646250367
for n_estimators = 500 and max depth =  3
Log Loss : 1.3232784510439812
for n_estimators = 500 and max depth =  5
Log Loss : 1.2963010154022812
for n_estimators = 500 and max depth =  10
Log Loss : 1.6550268955602976
for n_estimators = 1000 and max depth =  2
Log Loss : 1.485712647397059
for n_estimators = 1000 and max depth =  3
Log Loss : 1.3295649352575725
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2849156403577384
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6446054521913613
```




```
For values of best alpha =  100 The train log loss is: 0.06607752857219287
```

```
For values of best alpha =  100 The cross validation log loss is:
1.2127621969794387
For values of best alpha =  100 The test log loss is: 1.3088361714022614
```

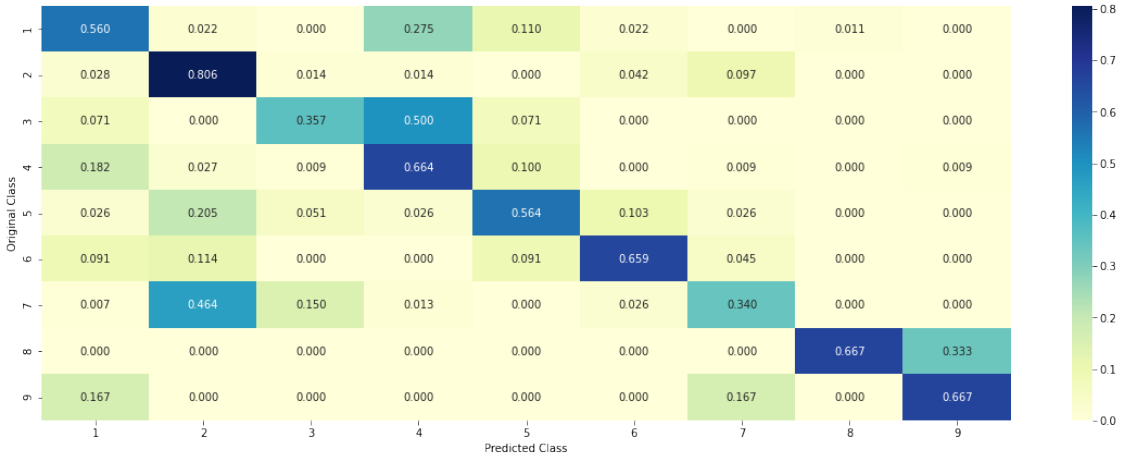
4.5.4. Testing model with best hyper parameters (Response Coding)

[ ]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# predict_proba (X)        Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/random-forest-and-their-construction-2/
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)],␣
 ↪n_estimators=alpha[int(best_alpha/4)], criterion='gini',␣
 ↪max_features='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding,␣
 ↪train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.2127621969794387
Number of mis-classified points : 0.44360902255639095
-------------------- Confusion matrix --------------------
```

---------- Precision matrix (Columm Sum=1) ----------

---------- Recall matrix (Row sum=1) ----------



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

[ ]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)],␣
↪criterion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,␣
↪n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)
```

[ ]:
```
CalibratedClassifierCV(base_estimator=RandomForestClassifier(bootstrap=True,
                                                             ccp_alpha=0.0,
                                                             class_weight=None,
                                                             criterion='gini',
                                                             max_depth=5,
max_features='auto',
max_leaf_nodes=None,
                                                             max_samples=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
                                                             min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
                                                             n_estimators=100,
                                                             n_jobs=-1,
                                                             oob_score=False,
                                                             random_state=42,
                                                             verbose=0,
                                                             warm_start=False),
                       cv=None, method='sigmoid')
```

[ ]:
```
test_point_index = 7
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].
↪reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
↪predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
```

```
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0246 0.192  0.2322 0.0279 0.09   0.0519 0.292
0.0667 0.0227]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

[ ]:
```
test_point_index = 46
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].
 ↪reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
```

```
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0039 0.0465 0.0055 0.0076 0.0022 0.0209
0.9087 0.0021 0.0026]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

[ ]:
```
# 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, max_iter=None, tol=None,
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#-------------------------------


# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1,␣
 ↪decision_function_shape=ovr, random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/mathematical-derivation-copy-8/
# --------------------------------


# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)
```

```
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given␣
 ↪training data.
# predict(X)        Perform classification on samples in X.
# predict_proba (X)        Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/random-forest-and-their-construction-2/
# --------------------------------


clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log',␣
 ↪class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.001, penalty='l2', loss='hinge',␣
 ↪class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.1)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.
 ↪predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.
 ↪predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.
 ↪predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
```

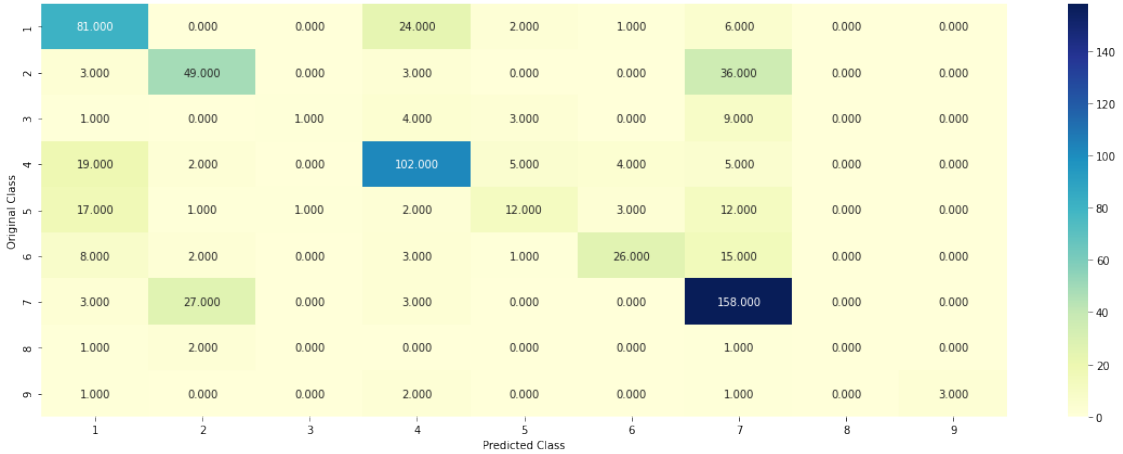
```
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3],␣
 ↪meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" %␣
 ↪(i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 0.96
Support vector machines : Log Loss: 1.08
Naive Bayes : Log Loss: 1.14
---------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.810
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.663
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.214
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.063
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.206
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.468
```

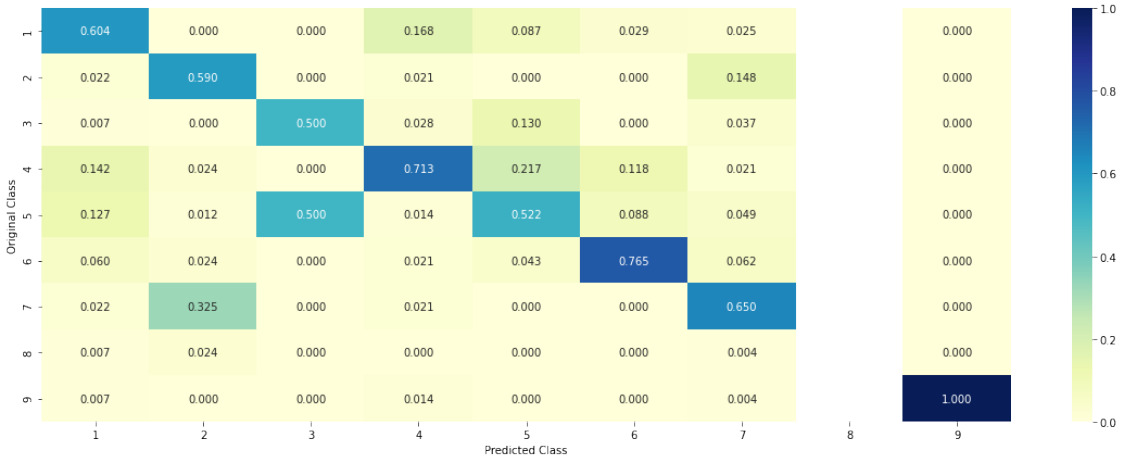
4.7.2 testing the model with the best hyper parameters

[ ]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3],␣
 ↪meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of missclassified point :", np.count_nonzero((sclf.
 ↪predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.
 ↪predict(test_x_onehotCoding))
```

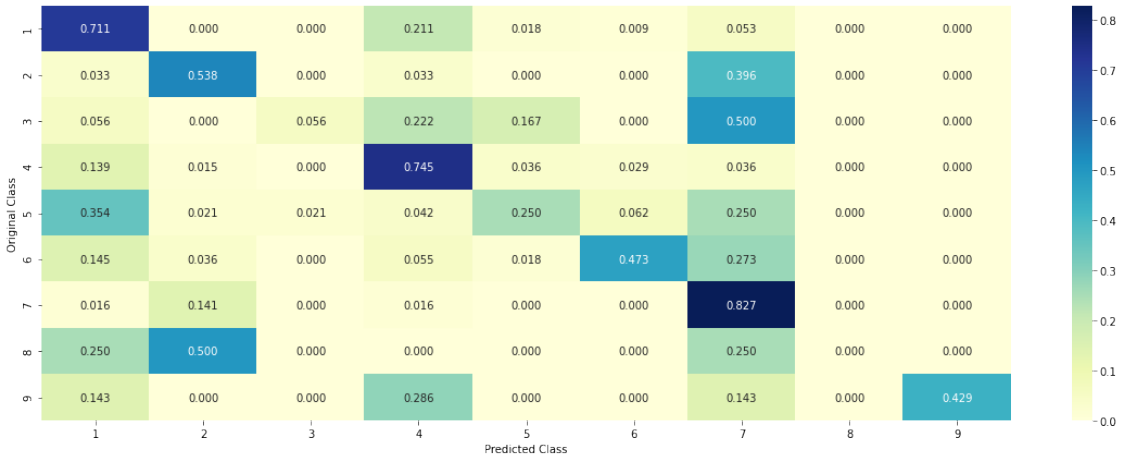
```
Log loss (train) on the stacking classifier : 0.5517685824247897
Log loss (CV) on the stacking classifier : 1.0634839759609753
Log loss (test) on the stacking classifier : 1.0881895898306622
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```



---------- Precision matrix (Columm Sum=1) ----------



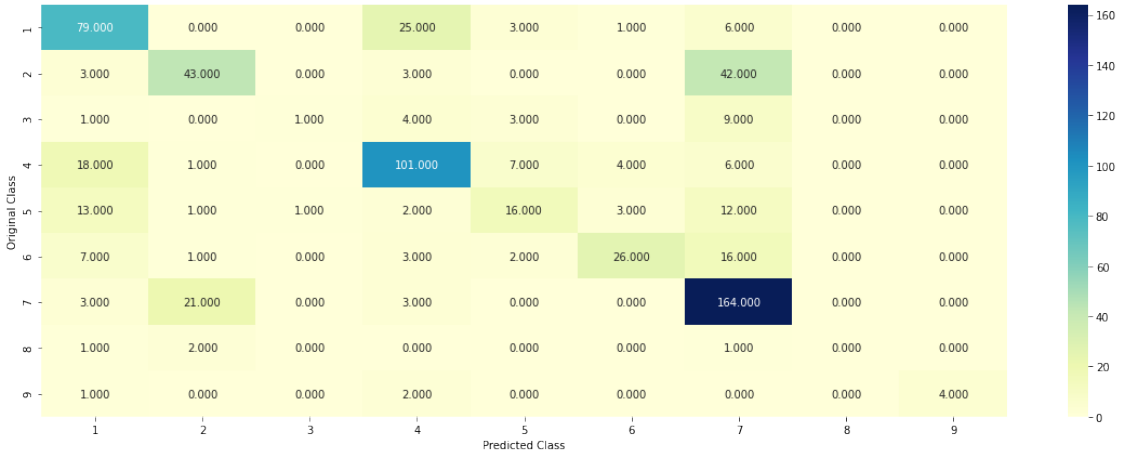
---------- Recall matrix (Row sum=1) ----------



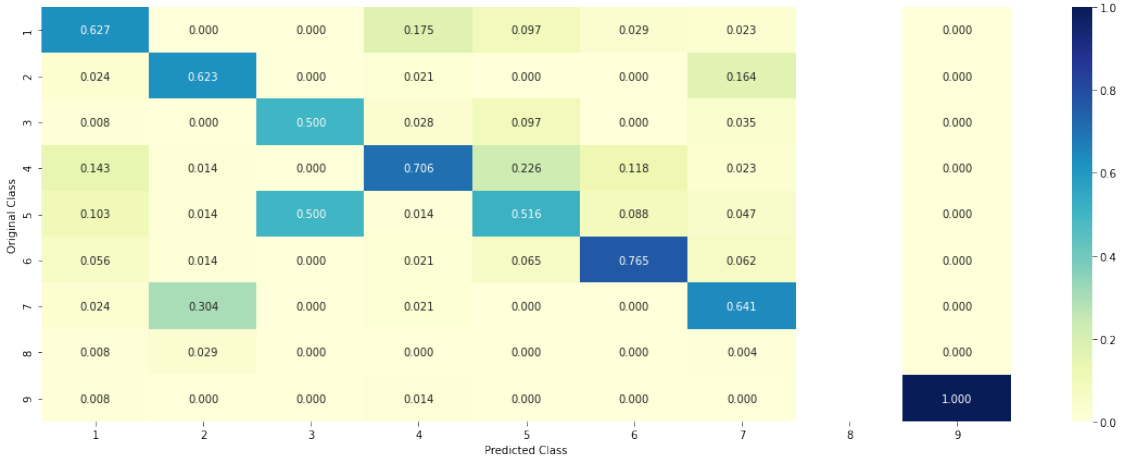
4.7.3 Maximum Voting classifier

[ ]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf',␣
 ↪sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.
 ↪predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.
 ↪predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.
 ↪predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.
 ↪predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.
 ↪predict(test_x_onehotCoding))
```

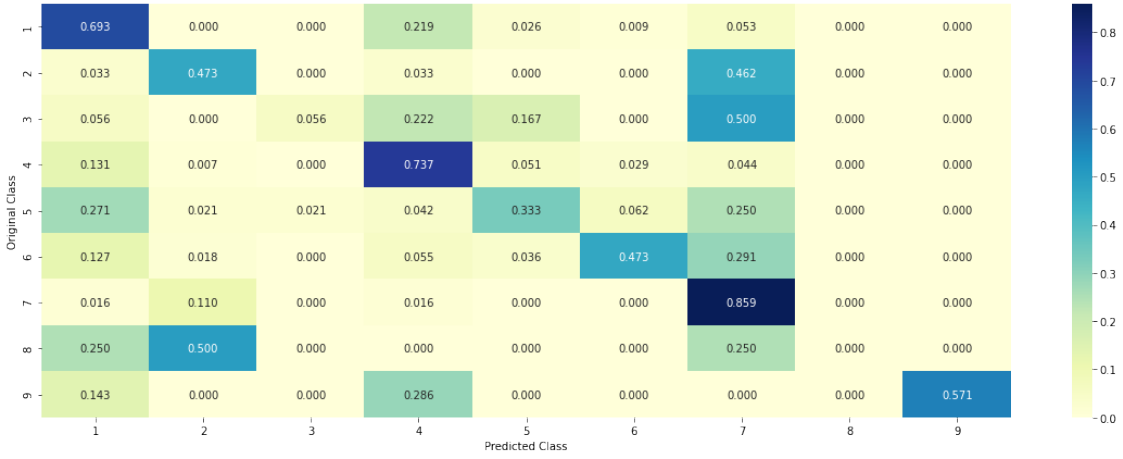
```
Log loss (train) on the VotingClassifier : 0.6763039835210329
Log loss (CV) on the VotingClassifier : 0.9914373576851239
Log loss (test) on the VotingClassifier : 1.0280567439016426
Number of missclassified point : 0.3473684210526316
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



---------- Recall matrix (Row sum=1) ----------



5. Assignments

 Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer a
 Instead of using all the words in the dataset, use only the top 1000 words based of tf-id
Apply Logistic regression with CountVectorizer Features, including both unigrams and bigra
 Try any of the feature engineering techniques discussed in the course to reduce the CV an

1 Feature Engineering

1.0.1 Defining a new feature that has a sense of count of the number of times the given Gene and Variation where mentioned in The corrosponding text.

[151]:
```
#defining the gene and variation corpus  from train data to fit␣
 ↪CountVectorizer on
corpus = list()

corpus.extend([i for i in train_df['Gene'].unique()])
corpus.extend([i for i in train_df['Variation'].unique()])
```

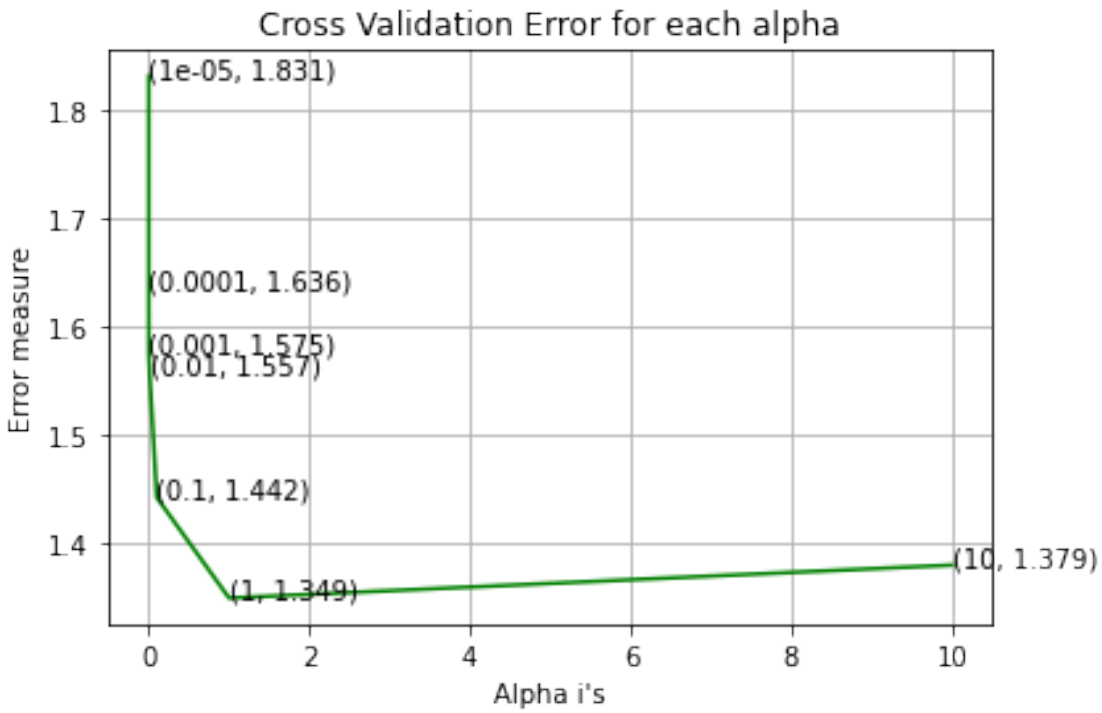
[152]:
```
cv = CountVectorizer()
countGV = cv.fit(corpus)
gene_variation_features = countGV.get_feature_names()
#keeping a sense of count of Gene and Varitions mentions in the corrosponding␣
 ↪text field
train_text = cv.transform(train_df['TEXT'])
test_text = cv.transform(test_df['TEXT'])
cv_text = cv.transform(cv_df['TEXT'])
```

Checking importance of this feature

[153]:
```
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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text, y_train)
```

```
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text, y_train)
    predict_y = sig_clf.predict_proba(cv_text)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,␣
 ↪predict_y, labels=clf.classes_, eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_text, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text, y_train)

predict_y = sig_clf.predict_proba(train_text)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.8308894970074714
For values of alpha =  0.0001 The log loss is: 1.6355335358699359
For values of alpha =  0.001 The log loss is: 1.5751255914450568
For values of alpha =  0.01 The log loss is: 1.5566201120526322
For values of alpha =  0.1 The log loss is: 1.4415609721193456
For values of alpha =  1 The log loss is: 1.3486345667912825
For values of alpha =  10 The log loss is: 1.379064685976835
```




```
For values of best alpha =  1 The train log loss is: 1.1976893698209352
For values of best alpha =  1 The cross validation log loss is:
1.3486345667912825
For values of best alpha =  1 The test log loss is: 1.4096286011622041
```

4. Machine Learning Models

[98]:
```
#Data preparation for ML models.

#Misc. functionns for ML models

def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities␣
 ↪belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y-␣
 ↪test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

[58]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

[59]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(max_features = 1000)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".
 ↪format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point␣
 ↪[{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".
 ↪format(word,yes_no))
```

```
    print("Out of the top ",no_features," features ", word_present, "are␣
 ↪present in query point")
```

Stacking of extracted features with prior ones

[154]:
```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding =␣
 ↪hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =␣
 ↪hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding =␣
 ↪hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text)).tocsr()
train_x_onehotCoding = hstack((train_x_onehotCoding,␣
 ↪train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text)).tocsr()
test_x_onehotCoding = hstack((test_x_onehotCoding,␣
 ↪test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text)).tocsr()
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).
 ↪tocsr()
cv_y = np.array(list(cv_df['Class']))
```

[155]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",␣
 ↪train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ",␣
 ↪test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data␣
 ↪=", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 4410)
(number of data points * number of features) in test data =  (665, 4410)
(number of data points * number of features) in cross validation data = (532,
4410)
```

4.1. Logistic Regression

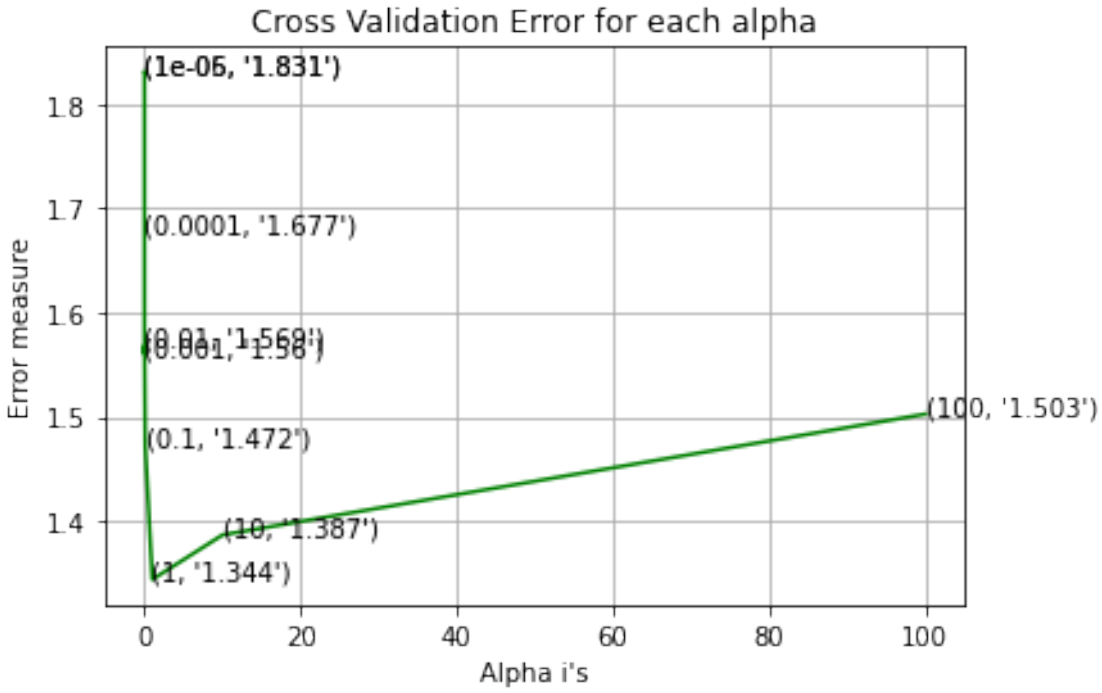
4.1.1. With Class balancing

4.1.1.1. Hyper paramter tuning

[156]:
```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',␣
 ↪loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use␣
 ↪log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.8308894970074714
for alpha = 1e-05
Log Loss : 1.8308894970074714
for alpha = 0.0001
Log Loss : 1.6773568599568798
for alpha = 0.001
Log Loss : 1.5602105098779244
for alpha = 0.01
Log Loss : 1.5686318728117905
for alpha = 0.1
Log Loss : 1.4724489665432063
for alpha = 1
Log Loss : 1.3441209096726794
for alpha = 10
Log Loss : 1.3868747383782516
for alpha = 100
Log Loss : 1.5027439378609453
```

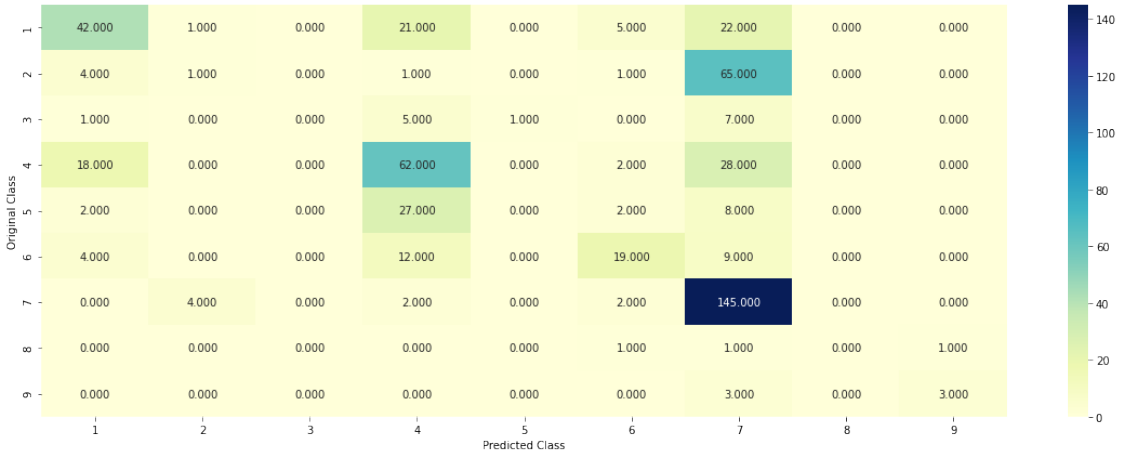


```
For values of best alpha =  1 The train log loss is: 1.195774970817202
For values of best alpha =  1 The cross validation log loss is:
1.3441209096726794
For values of best alpha =  1 The test log loss is: 1.397381574671639
```

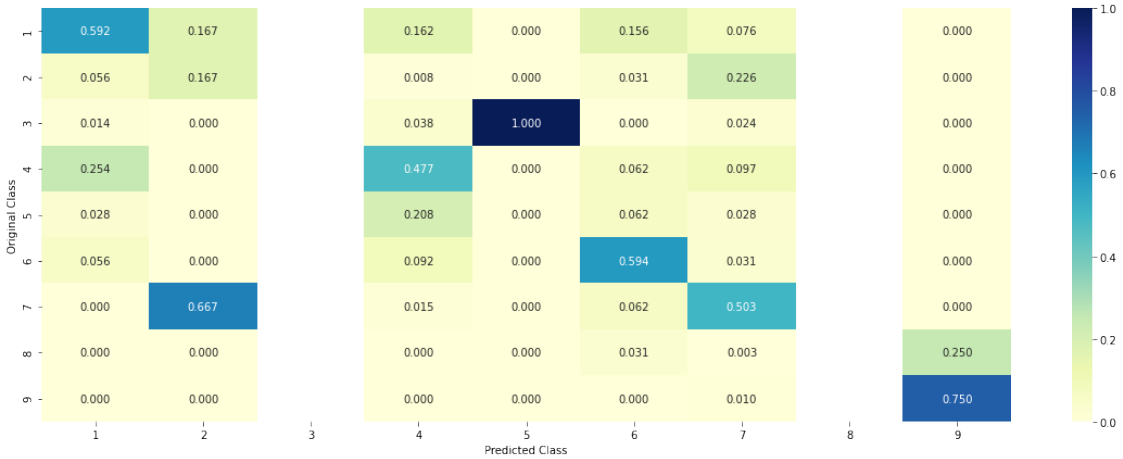
4.1.1.2. Testing the model with best hyper paramters

[157]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,␣
 ↪cv_x_onehotCoding, cv_y, clf)
```

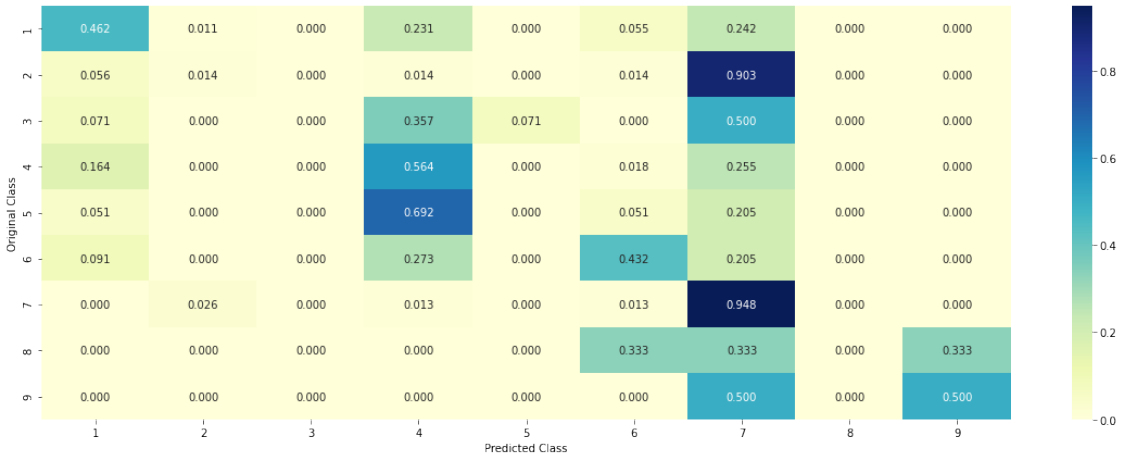
```
Log loss : 1.3441209096726794
Number of mis-classified points : 0.48872180451127817
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Column Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.1.2. Without Class balancing

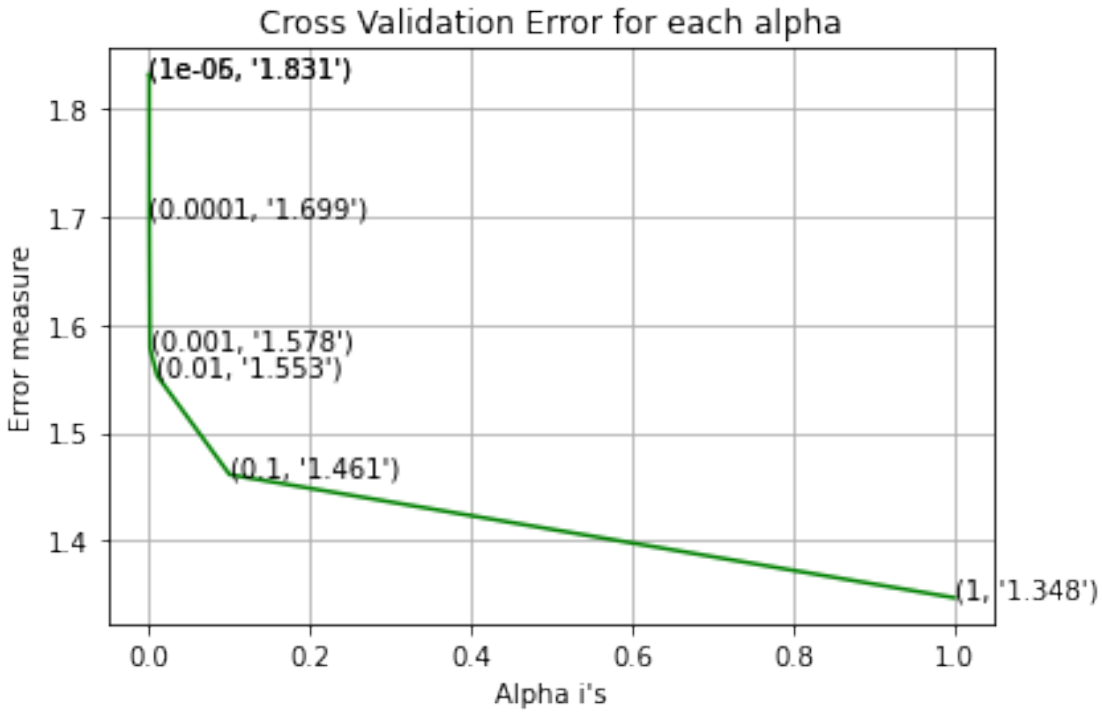
4.1.2.1. Hyper paramter tuning

[158]:
```
# 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#-------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
```

```
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None,␣
 ↪method=sigmoid, cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#--------------------------------------
# video link:
#--------------------------------------

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_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(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
```

```
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.8308894970074714
for alpha = 1e-05
Log Loss : 1.8308894970074714
for alpha = 0.0001
Log Loss : 1.6994684958565733
for alpha = 0.001
Log Loss : 1.5779003576224142
for alpha = 0.01
Log Loss : 1.5525898492794907
for alpha = 0.1
Log Loss : 1.4612554091206744
for alpha = 1
Log Loss : 1.347599465070658
```



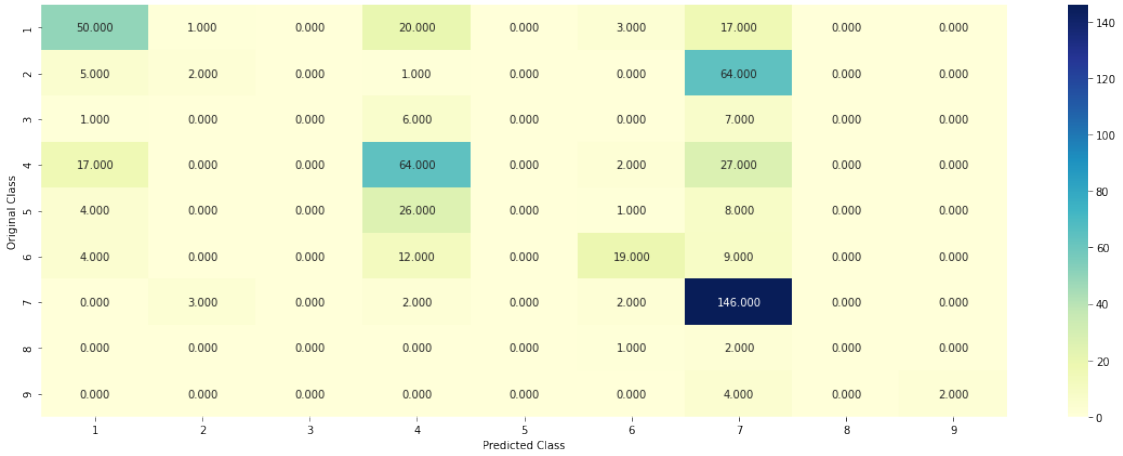

```
For values of best alpha =  1 The train log loss is: 1.1966491522270384
For values of best alpha =  1 The cross validation log loss is:
```

```
1.347599465070658
For values of best alpha =  1 The test log loss is: 1.4088421732523462
```

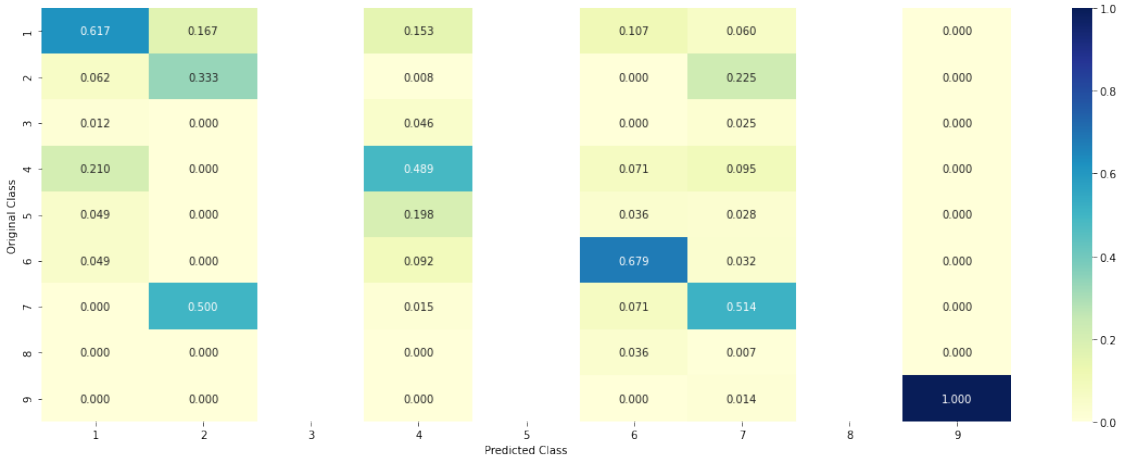
4.1.2.2. Testing model with best hyper parameters

[159]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,␣
 ↪cv_x_onehotCoding, cv_y, clf)
```

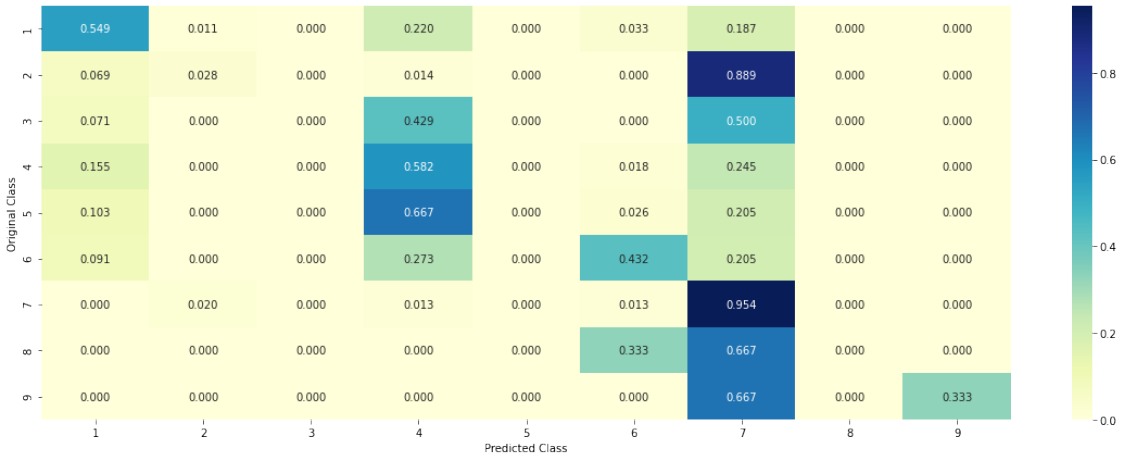
```
Log loss : 1.347599465070658
Number of mis-classified points : 0.4680451127819549
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



```
-------------------- Recall matrix (Row sum=1) --------------------
```



1.1 workflow

The presented notebook shows a simple work flow to answere a kaggle compition, Personalized Cancer Diagnosis. The task of automating this will help pathologists alot by reducing thier burdens to go through texts to classify these gene and varitions into one of the 9 classes. 1. The overview after merging files shows the data contains 5 columns : id, Gene, Variation, Text, class with 3321 data-points. 2. This is a multi Classification problem which tries to predict one of the nine class given gene-variation with corrosponding text belongs to. 3. Starting with simple EDA on data to check the distribution of Classes, and unique number of gene and variation. 4. ploting histogram of gene and variation. 5. Given gene and variation and textare featurize with repsonse coding and tfidf with top 1000 words, also with count vectorizer(having bi grams) . 4. The gene and variation are tested for importance with a simple logistic regression model and the log loss is put against a random model. 5. The Text data is preprocessed for stopword removal, lemetization and etc before featurization. 6. Model training starts from here. 7. A Logistic Regression model is trained with bag of words representation having bi and tri grams. 8. with Tfidf representation Naive Bayes, KNN, Logistic Regression, Linear SVM and Random forest is trained. 9. Random forest is aslo trained with Response coding. 10. All the Log loss is recored.

11. Another featurization is tried on Text by considering a sense of number of times gene and variation gets mentioned in their corrosponding text field using CountVectorizer and a logistic regression model is trained on this feature concatinated with prior onces.

[163]:
```
from prettytable import PrettyTable

##############################################################################
print('Performance Table of Models')
x = PrettyTable()
x.field_names =["Models","Train","CV","Test","Misclassified(%)"]
```

```
x.add_row(["LR(Class balanced) BOW(bi grams)",0.7335,1.0410,1.0984,0.3477])
x.add_row(["LR(Class unbalanced)  BOW(bi grams)",0.7272,1.0412,1.0895,0.3533])
x.add_row(["NB TFIDF",0.7804,1.1437,1.1790,0.3665])
x.add_row(["KNN TFIDF",0.6616,1.0031,1.0897,0.35528])
x.add_row(["LR(Class balanced) TFIDF",0.5479,0.9628,0.9800,0.3383])
x.add_row(["LR(Class Unbalanced) TFIDF",0.5371,1.004,1.0059,0.3402])
x.add_row(["Linear SVM TFIDF",0.7966,1.0795,1.1097,0.3345])
x.add_row(["Random forest TFIDF",0.8395,1.1834,1.1822,0.4097])
x.add_row(["Random forest Response coding",0.0660,1.2127,1.3088,0.4436])
x.add_row(["LR(Class balanced) feature engineering",1.1457,1.3441,1.3973,0.
 ↪4807])
x.add_row(["LR(Class Unbalanced) feature engineering",1.1908,1.3475,1.4088,0.
 ↪4680])

print(x)
```

```
Performance Table of Models
+-----------------------------------------+--------+--------+--------+---------
---------+
|                  Models                 | Train  |   CV   |  Test  |
Misclassified(%) |
+-----------------------------------------+--------+--------+--------+---------
---------+
|     LR(Class balanced) BOW(bi grams)     | 0.7335 | 1.041  | 1.0984 |
0.3477      |
|   LR(Class unbalanced)  BOW(bi grams)    | 0.7272 | 1.0412 | 1.0895 |
0.3533      |
|                NB TFIDF                  | 0.7804 | 1.1437 | 1.179  |
0.3665      |
|               KNN TFIDF                 | 0.6616 | 1.0031 | 1.0897 |
0.35528      |
|         LR(Class balanced) TFIDF         | 0.5479 | 0.9628 |  0.98  |
0.3383      |
|        LR(Class Unbalanced) TFIDF        | 0.5371 | 1.004  | 1.0059 |
0.3402      |
|            Linear SVM TFIDF              | 0.7966 | 1.0795 | 1.1097 |
0.3345      |
|           Random forest TFIDF            | 0.8395 | 1.1834 | 1.1822 |
0.4097      |
|      Random forest Response coding       | 0.066  | 1.2127 | 1.3088 |
0.4436      |
|  LR(Class balanced) feature engineering  | 1.1457 | 1.3441 | 1.3973 |
0.4807      |
| LR(Class Unbalanced) feature engineering | 1.1908 | 1.3475 | 1.4088 |
0.468       |
+-----------------------------------------+--------+--------+--------+---------
```

```
---------+
```