Cancer diagnosis using machine learning

December 1, 2021

[5]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.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 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
```

[23]:
```
data = pd.read_csv('training/training_variants')
print('The number of data points ' , data.shape[0])
print('The number of features ' , data.shape[1])
```

```
The number of data points  3321
The number of features  4
```

[24]:
```
print("Features are : " , data.columns.values)
```

```
Features are :  ['ID' 'Gene' 'Variation' 'Class']
```

[25]:
```
data.head()
```

[25]:
```
   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
```

1 Reading text data

[26]:
```
data_text =pd.read_csv("training/
 ↪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']
```

[26]:
```
   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…
```

[27]:
```
import nltk
```

[28]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Himanshu\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[28]:

```
True
```

[33]:

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

[34]:

```
stop_words
```

[34]:

```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
 'been',
 'before',
 'being',
 'below',
 'between',
 'both',
 'but',
 'by',
 'can',
 'couldn',
 "couldn't",
 'd',
 'did',
 'didn',
 "didn't",
 'do',
 'does',
 'doesn',
 "doesn't",
 'doing',
 'don',
 "don't",
 'down',
 'during',
 'each',
 'few',
 'for',
 'from',
 'further',
 'had',
 'hadn',
 "hadn't",
 'has',
 'hasn',
 "hasn't",
 'have',
 'haven',
 "haven't",
 'having',
 'he',
 'her',
 'here',
 'hers',
 'herself',
 'him',
 'himself',
 'his',
 'how',
 'i',
 'if',
 'in',
 'into',
 'is',
 'isn',
 "isn't",
 'it',
 "it's",
 'its',
 'itself',
 'just',
 'll',
 'm',
 'ma',
 'me',
 'mightn',
 "mightn't",
 'more',
 'most',
 'mustn',
 "mustn't",
 'my',
 'myself',
 'needn',
 "needn't",
 'no',
 'nor',
 'not',
 'now',
 'o',
 'of',
 'off',
 'on',
 'once',
 'only',
 'or',
 'other',
 'our',
 'ours',
 'ourselves',
 'out',
 'over',
 'own',
 're',
 's',
 'same',
 'shan',
 "shan't",
 'she',
 "she's",
 'should',
 "should've",
 'shouldn',
 "shouldn't",
 'so',
 'some',
 'such',
 't',
 'than',
 'that',
 "that'll",
 'the',
 'their',
 'theirs',
 'them',
 'themselves',
 'then',
 'there',
 'these',
 'they',
 'this',
 'those',
 'through',
 'to',
 'too',
 'under',
 'until',
 'up',
 've',
 'very',
 'was',
 'wasn',
 "wasn't",
 'we',
 'were',
 'weren',
 "weren't",
 'what',
 'when',
 'where',
 'which',
 'while',
 'who',
 'whom',
 'why',
 'will',
 'with',
 'won',
 "won't",
 'wouldn',
 "wouldn't",
 'y',
 'you',
 "you'd",
 "you'll",
 "you're",
 "you've",
 'your',
 'yours',
 'yourself',
 'yourselves'}
```

[35]:

```
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
```

[36]:

```
#text processing stage.
start_time = time.process_time()
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.process_time() -␣
 ↪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 : 20.921875 seconds
```

[37]:

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

[37]:

```
   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

                                                TEXT
0  cyclin dependent kinases cdks regulate variety…
1  abstract background non small cell lung cancer…
```

```
2  abstract background non small cell lung cancer…
3  recent evidence demonstrated acquired uniparen…
4  oncogenic mutations monomeric casitas b lineag…
```

[50]:

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

[50]:

```
        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
```

[51]:

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

[52]:

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

[52]:

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

2 Test, Train and Cross Validation Split|

[64]:

```
y_true = result['Class'].values #the true values of y are stored here.
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)
```

[65]:

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

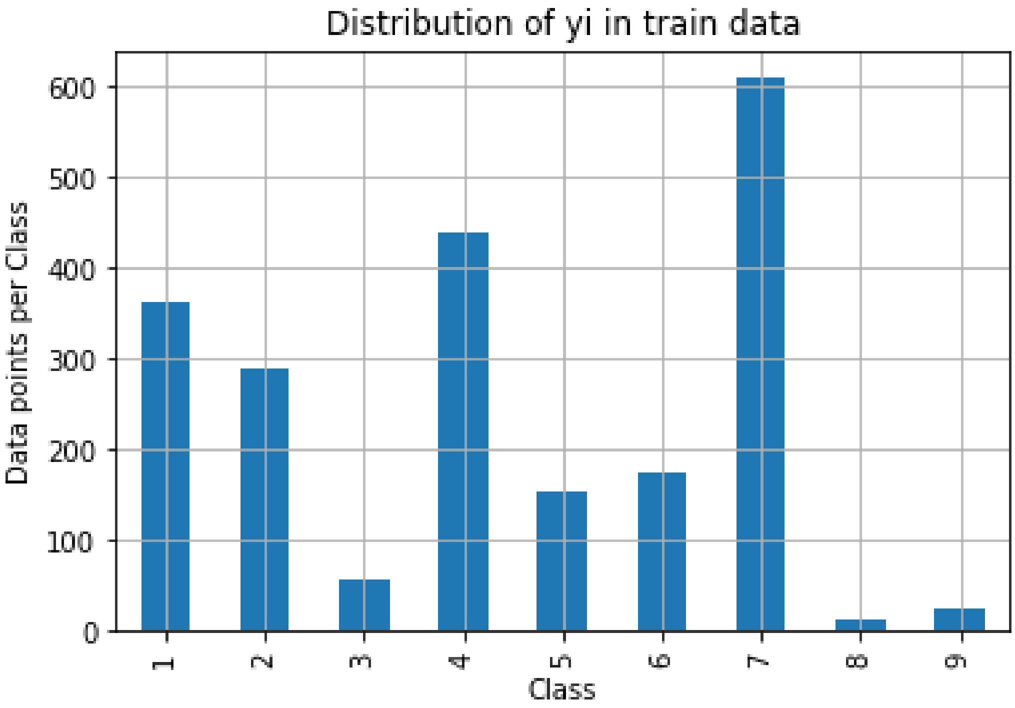
```
Number of points in train data 2124
Number of points in cross validation data 532
Number of points in test data 665
```

2.0.1 Distribution of Yis in the train test and cross validation data

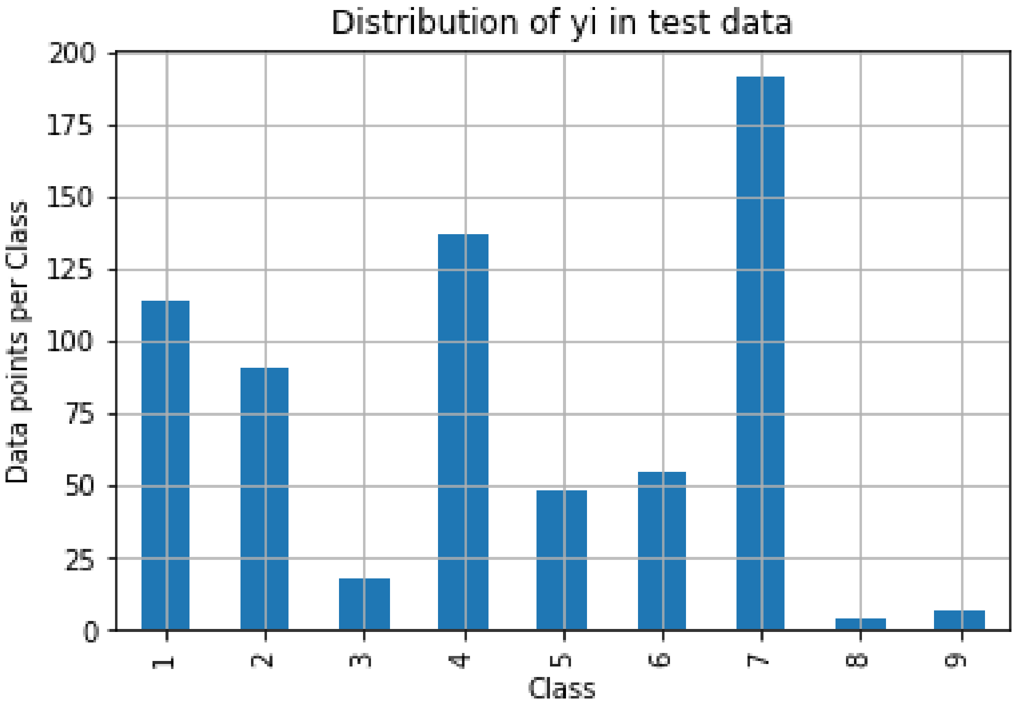
[66]:

```
# 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().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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)
```

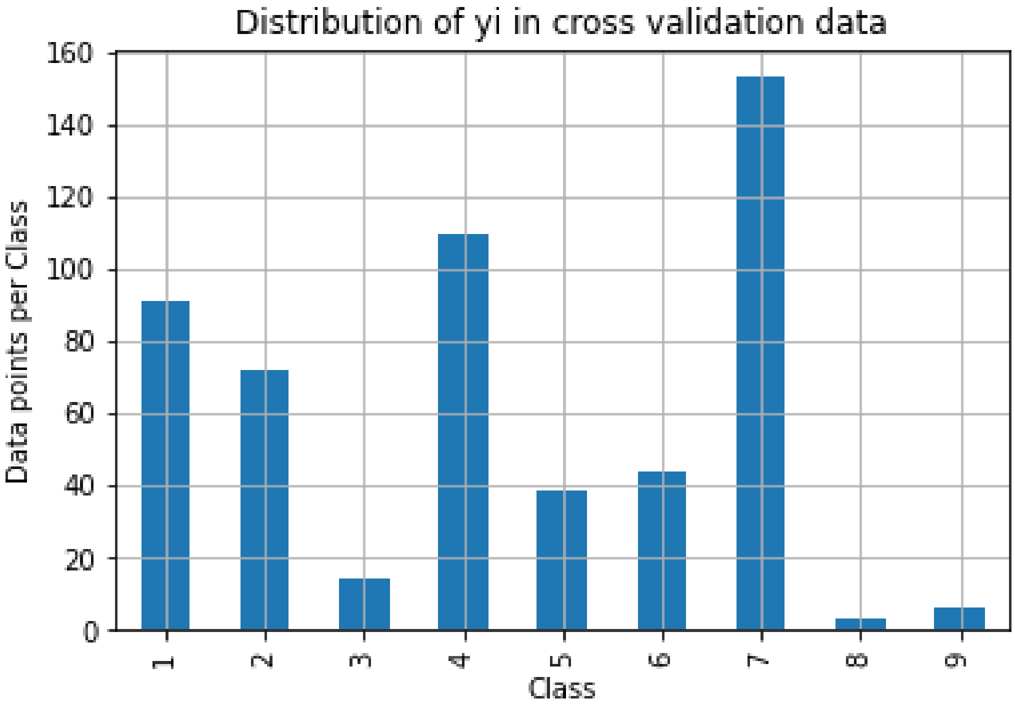
```
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# 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 Creating a random model

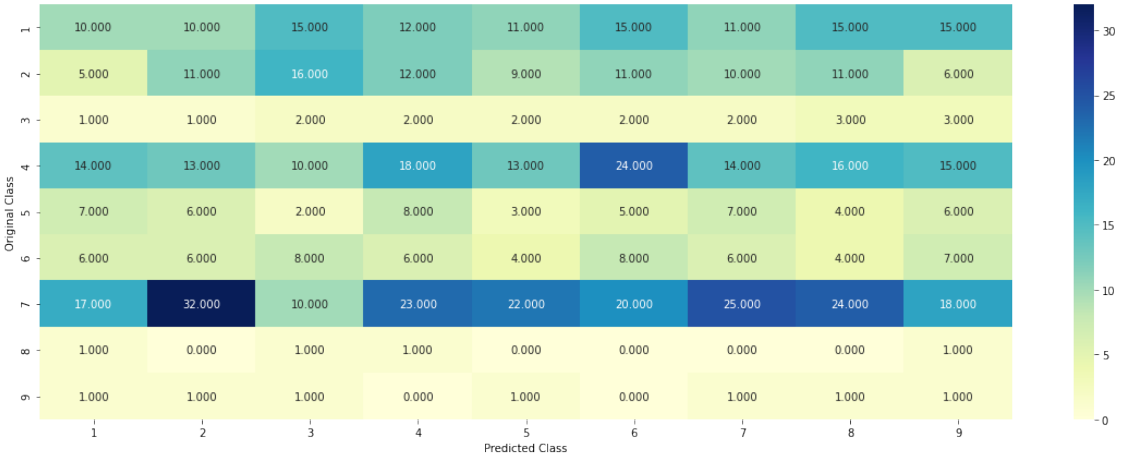
[68]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i␣
 ↪are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)


    B =(C/C.sum(axis=0))
    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()
```

[69]:

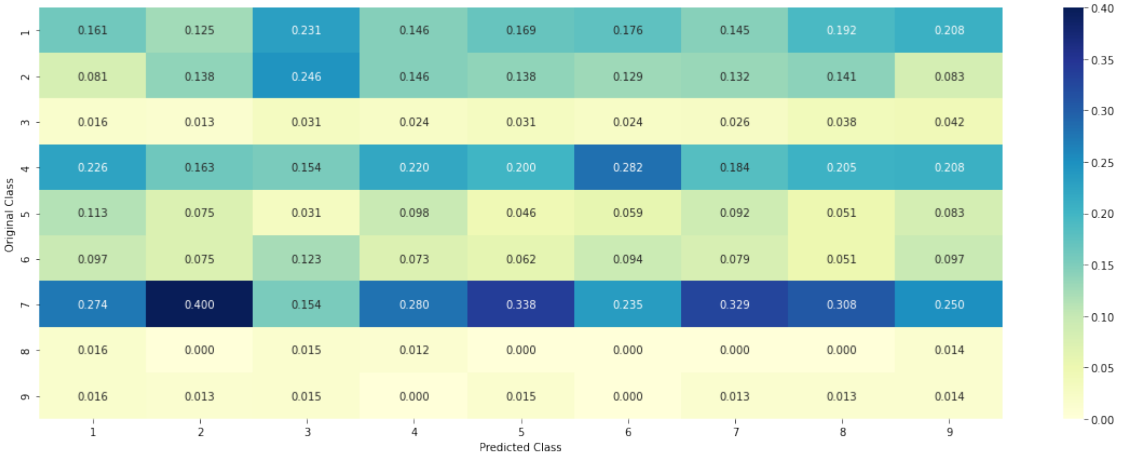
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their␣
 ↪sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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)
```

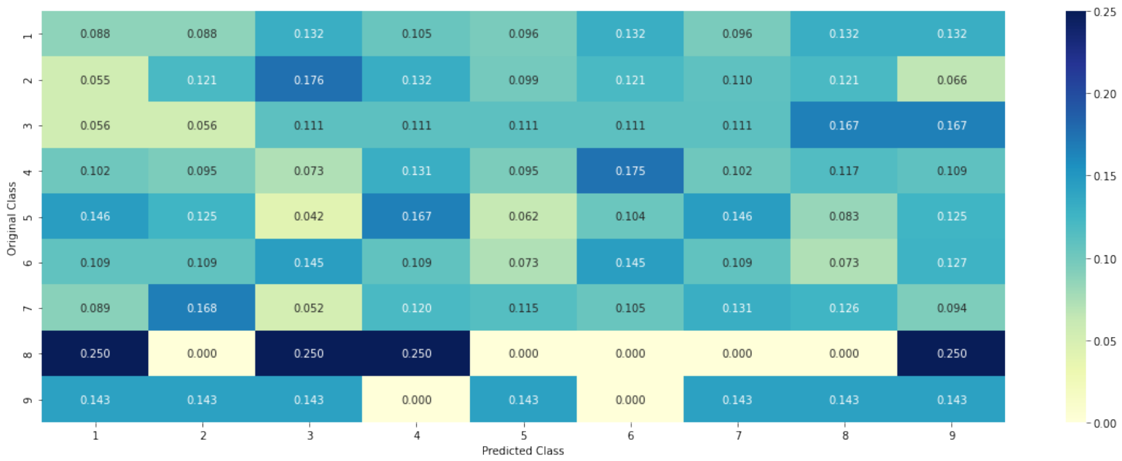
```
Log loss on Cross Validation Data using Random Model 2.445833388837183
Log loss on Test Data using Random Model 2.4391721050764392
-------------------- Confusion matrix --------------------
```



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



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



4 Univariate analysis

[91]:

```
# going to check wheather a feature is important or not
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': [0.200757575757575757, 0.037878787878787878, 0.
 ↪068181818181818177, 0.136363636363636363635, 0.25, 0.193181818181818181818, 0.
 ↪037878787878787878788, 0.037878787878787878788, 0.037878787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.
 ↪061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.
 ↪066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.
 ↪056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.
 ↪068181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.
 ↪0625, 0.056818181818181816],
    #      '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
```

[101]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
```

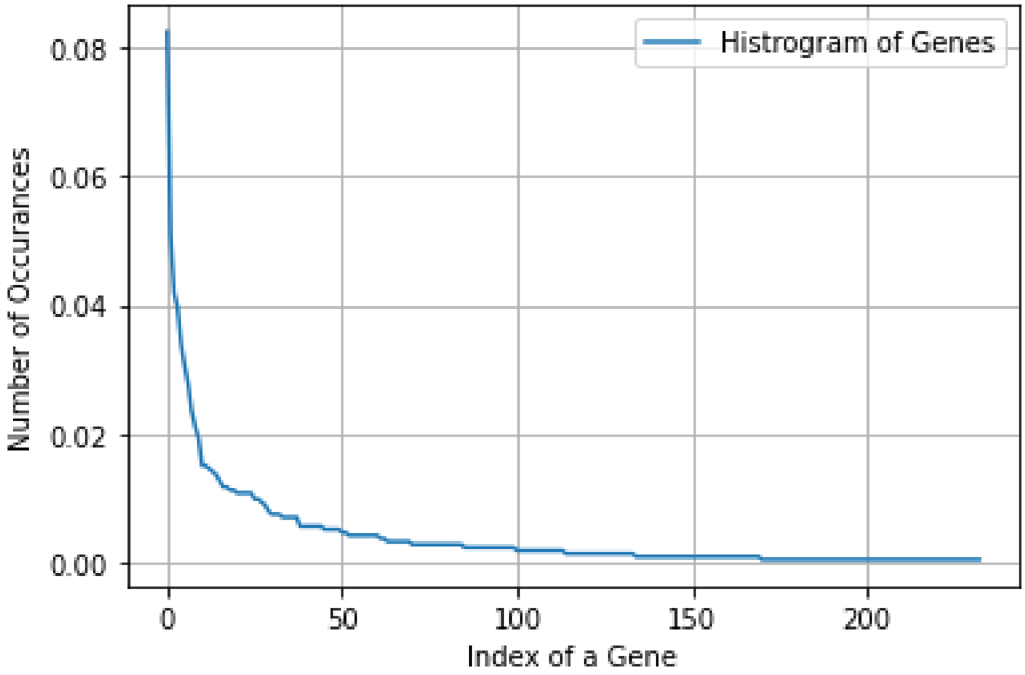
```
print(unique_genes.head(10))
```

```
Number of Unique Genes : 233
BRCA1     175
TP53      108
EGFR       89
BRCA2      84
PTEN       72
KIT        65
BRAF       59
ALK        50
ERBB2      45
PDGFRA     41
Name: Gene, dtype: int64
```

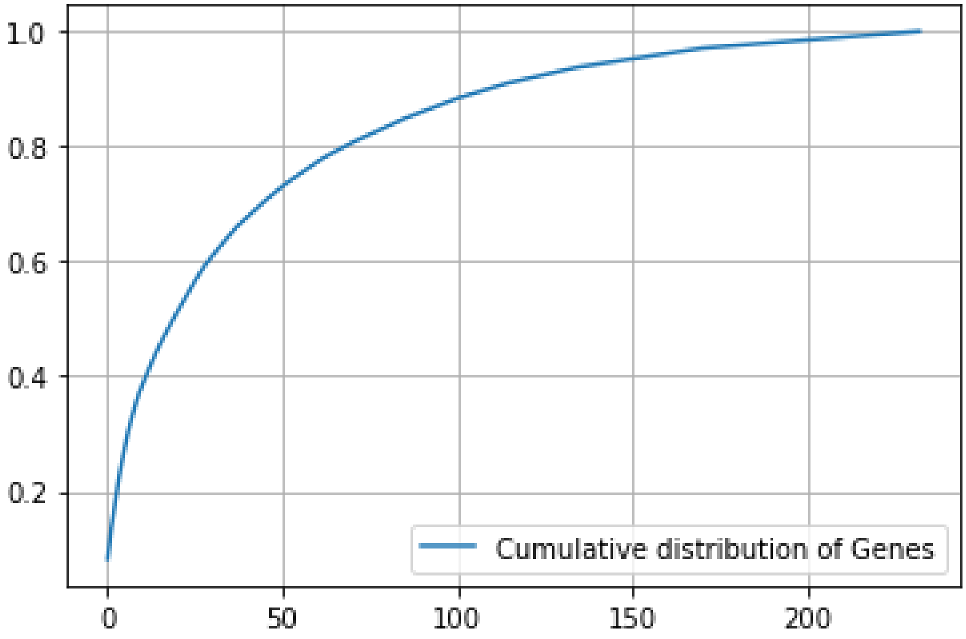
```
[102]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes␣
        ↪in the train data, and they are distibuted as follows",)
```

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

```
[103]: 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()
```



[104]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



[105]:

```
#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))
```

[106]:

```
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)
```

[107]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
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'])
```

[108]:

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

[108]:

```
403       TP53
1344      AKT1
62      PTPRT
3243      DDR2
2662     BRCA1
Name: Gene, dtype: object
```

[109]:

```
gene_vectorizer.get_feature_names()
```

[109]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 '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',
 'foxp1',
 'gata3',
 'gnaq',
 'gnas',
 'h3f3a',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikzf1',
 'il7r',
 'inpp4b',
 'jak1',
 'jak2',
 'kdm5a',
 'kdm5c',
 'kdr',
 'keap1',
 'kit',
 'kmt2a',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats1',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pax8',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad50',
 'rad51b',
 'rad51c',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 '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']
```

[110]:

```python
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)
```

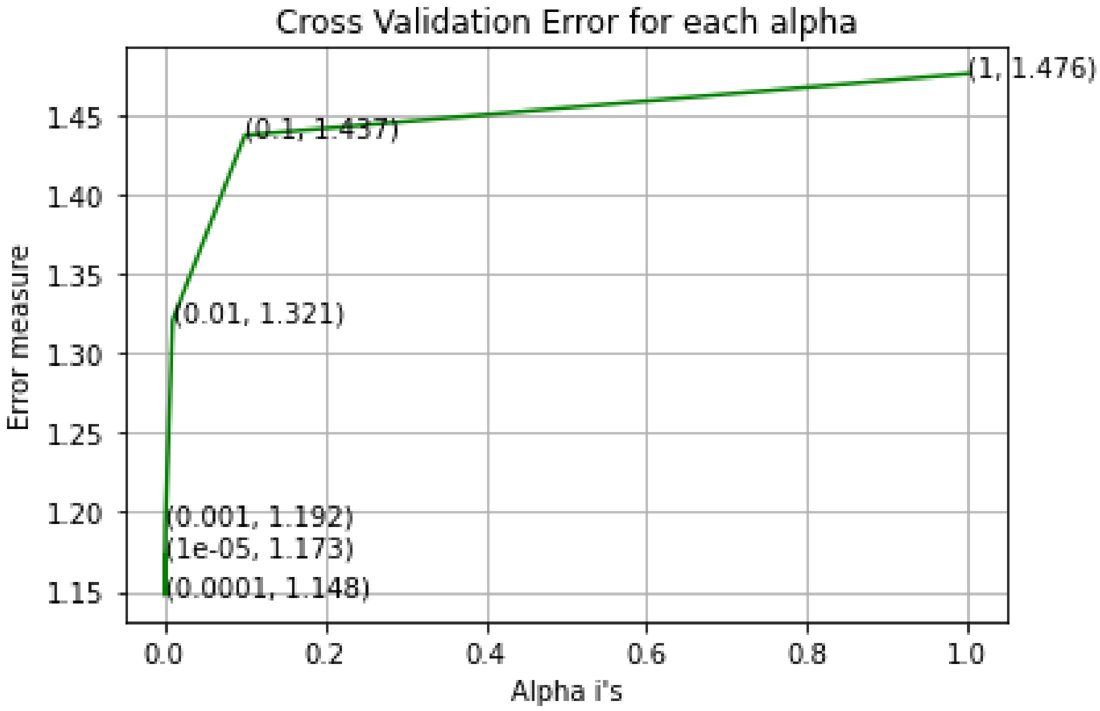
[111]:

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

[112]:

```
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: 1.1729777736036173
For values of alpha =  0.0001 The log loss is: 1.1477142315941569
For values of alpha =  0.001 The log loss is: 1.192401412977906
For values of alpha =  0.01 The log loss is: 1.3206330370303472
For values of alpha =  0.1 The log loss is: 1.4370589372442595
For values of alpha =  1 The log loss is: 1.4758297960949913
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0073788729689954
For values of best alpha =  0.0001 The cross validation log loss is:
1.1477142315941569
For values of best alpha =  0.0001 The test log loss is: 1.1746521916389614
```

[113]:
```
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  233  genes
in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 512 out of  532 : 96.2406015037594
```

5 Univariate analysis on variation feature

[114]:

```
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 : 1935
Truncating_Mutations    64
Deletion                45
Amplification           44
Fusions                 20
Overexpression           5
Q61L                     2
Q22K                     2
F384L                    2
R841K                    2
R170W                    2
Name: Variation, dtype: int64
```

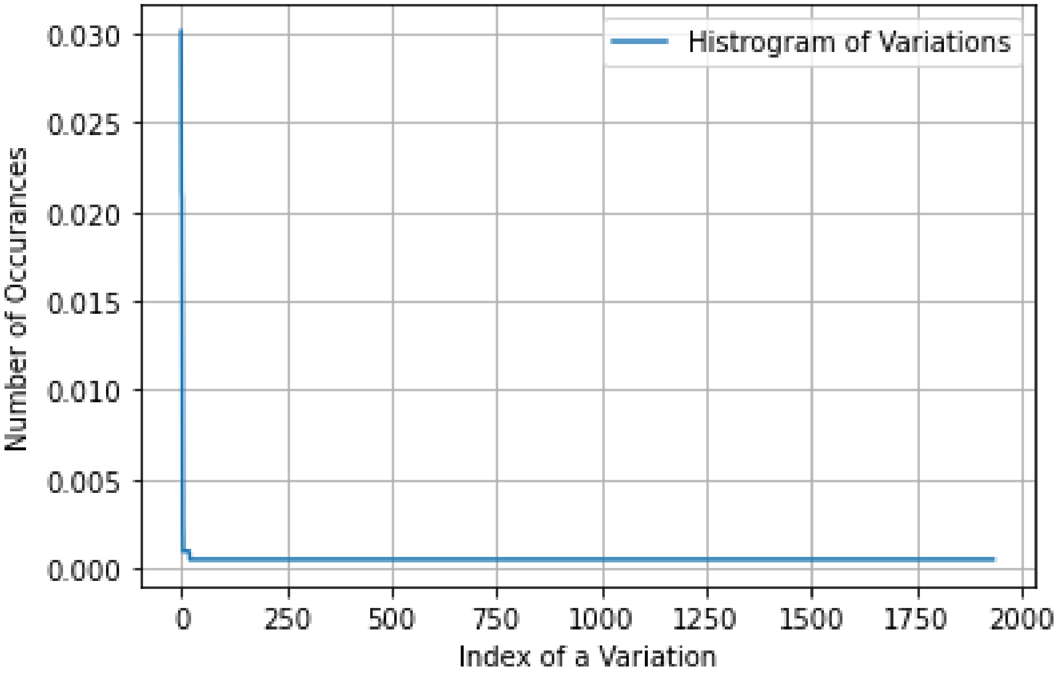
[115]:

```
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 1935 different categories of variations in the train data, and
they are distibuted as follows
```

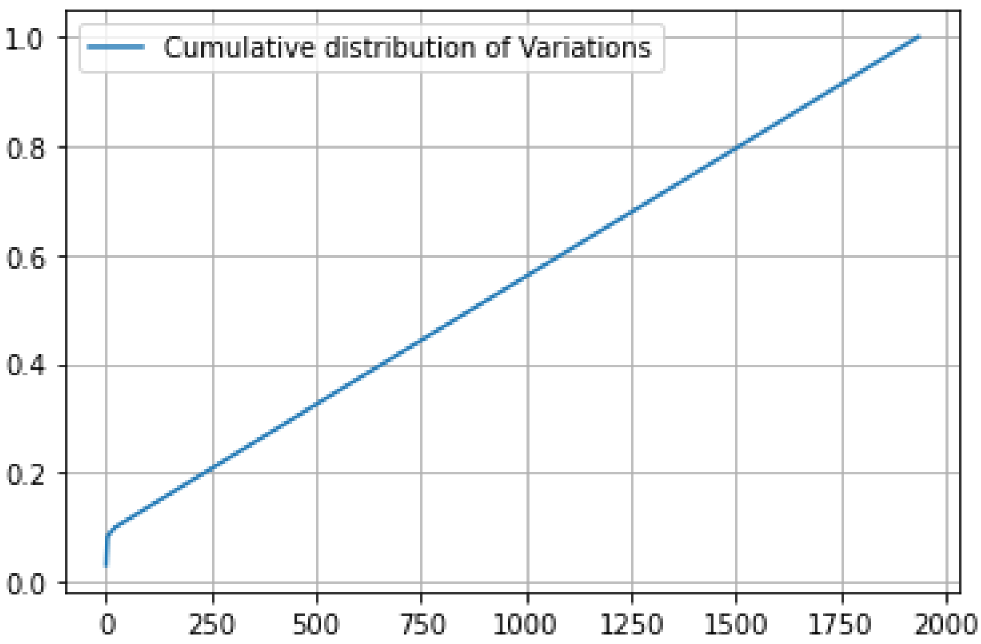
[116]:

```
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()
```



[117]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.03013183 0.05131827 0.0720339  … 0.99905838 0.99952919 1.        ]
```



[118]:
```
# 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))
```

[119]:
```
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)
```

[120]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
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'])
```

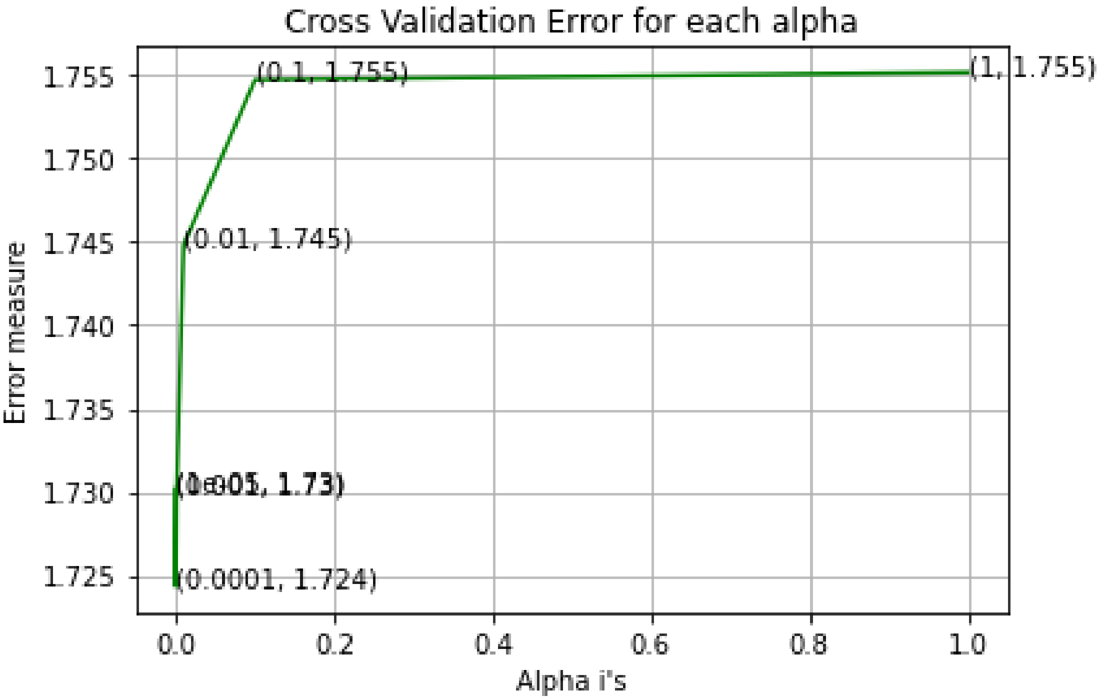
[121]:
```
print("train_variation_feature_onehotEncoded is converted feature using the␣
 ↪onne-hot encoding method. The shape of Variation feature:",␣
 ↪train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1969)
```

[123]:
```
alpha = [10 ** x for x in range(-5, 1)]

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: 1.7301681353883596
For values of alpha =  0.0001 The log loss is: 1.7243251717698724
For values of alpha =  0.001 The log loss is: 1.7299754097633104
For values of alpha =  0.01 The log loss is: 1.744806577457411
For values of alpha =  0.1 The log loss is: 1.7547215807542174
For values of alpha =  1 The log loss is: 1.755169665523679
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6610656550861532
For values of best alpha =  0.0001 The cross validation log loss is:
1.7243251717698724
For values of best alpha =  0.0001 The test log loss is: 1.6755727819003285
```

[125]:
```
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  1935  genes in test and cross
validation data sets?
Ans
1. In test data 81 out of 665 : 12.180451127819548
2. In cross validation data 49 out of  532 : 9.210526315789473
```

[126]:

```
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
```

[127]:

```
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
```

[128]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer = CountVectorizer(min_df=3)
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 : 54016
```

[129]:

```
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)
```

[130]:

```
#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)
```

[131]:

```
# 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
```

[132]:
```
# 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)
```

[133]:
```
#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()))
```

[134]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

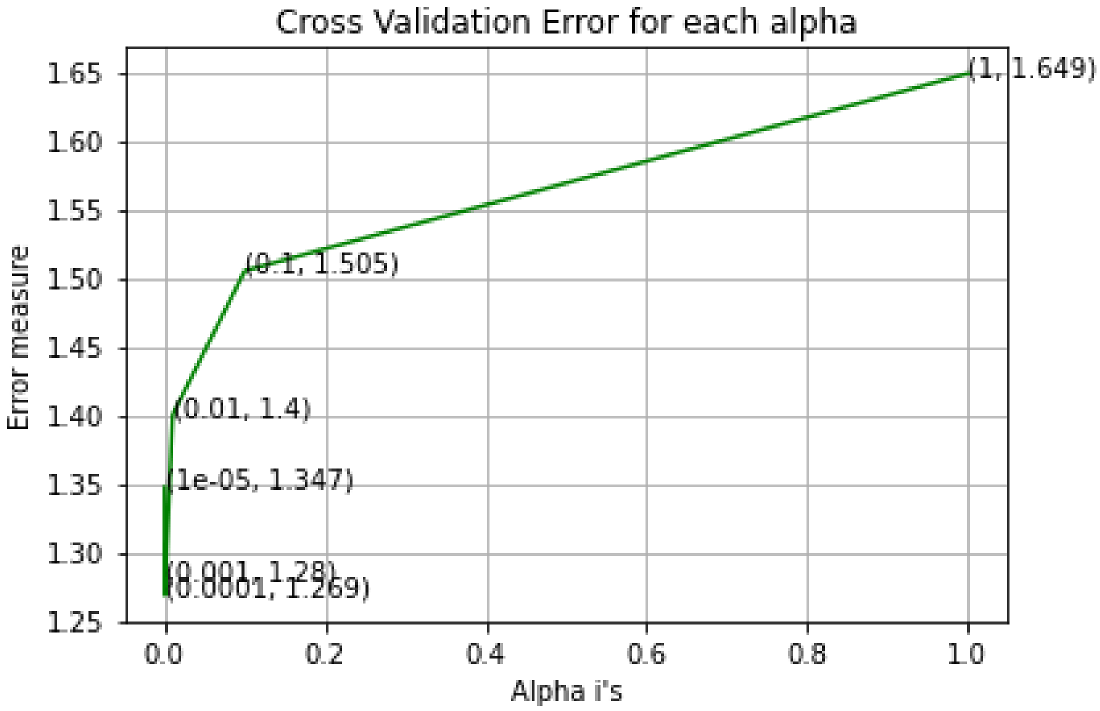
```
Counter({3: 5777, 4: 3848, 5: 2617, 6: 2612, 7: 2081, 9: 1979, 8: 1920, 10:
1584, 12: 1373, 11: 1211, 13: 1127, 14: 955, 15: 890, 18: 734, 16: 715, 20: 654,
17: 612, 21: 543, 22: 493, 24: 482, 19: 457, 25: 433, 30: 399, 27: 377, 26: 373,
23: 367, 28: 355, 41: 350, 29: 302, 33: 291, 49: 278, 32: 273, 36: 263, 37: 258,
40: 256, 31: 249, 34: 243, 35: 234, 42: 207, 38: 203, 39: 198, 50: 181, 46: 176,
44: 174, 48: 167, 43: 164, 51: 158, 45: 158, 54: 155, 56: 151, 47: 140, 55: 137,
52: 137, 58: 124, 53: 123, 69: 120, 61: 120, 57: 119, 74: 118, 64: 118, 59: 116,
63: 115, 60: 115, 73: 107, 66: 105, 72: 102, 78: 101, 65: 101, 76: 96, 62: 93,
71: 92, 70: 90, 80: 87, 82: 86, 75: 85, 68: 83, 67: 83, 86: 82, 84: 76, 81: 75,
77: 74, 92: 73, 79: 68, 91: 67, 94: 66, 85: 66, 83: 66, 98: 65, 87: 65, 89: 64,
88: 63, 90: 62, 96: 61, 101: 60, 112: 57, 114: 55, 103: 55, 100: 55, 117: 54,
115: 54, 104: 54, 99: 54, 111: 52, 110: 52, 113: 51, 102: 51, 124: 50, 106: 50,
119: 49, 93: 49, 97: 48, 95: 48, 126: 47, 122: 47, 147: 44, 133: 43, 107: 42,
135: 41, 109: 41, 138: 40, 134: 40, 129: 40, 131: 38, 130: 38, 127: 38, 143: 37,
120: 37, 108: 37, 105: 37, 144: 36, 136: 36, 146: 35, 128: 35, 137: 34, 116: 34,
157: 33, 125: 33, 123: 33, 182: 32, 158: 32, 142: 32, 139: 32, 132: 32, 165: 31,
149: 31, 171: 30, 161: 30, 151: 30, 140: 30, 189: 29, 169: 29, 160: 29, 141: 29,
175: 28, 150: 28, 210: 27, 162: 27, 154: 27, 148: 27, 121: 27, 197: 26, 179: 26,
176: 26, 174: 26, 172: 26, 156: 26, 145: 26, 226: 25, 215: 25, 181: 25, 180: 25,
170: 25, 168: 25, 163: 25, 159: 25, 152: 25, 244: 24, 209: 24, 184: 24, 177: 24,
155: 24, 219: 23, 212: 23, 183: 23, 153: 23, 196: 22, 193: 22, 164: 22, 236: 21,
205: 21, 202: 21, 188: 21, 187: 21, 118: 21, 198: 20, 186: 20, 255: 19, 228: 19,
222: 19, 216: 19, 203: 19, 201: 19, 191: 19, 166: 19, 252: 18, 225: 18, 217: 18,
206: 18, 199: 18, 194: 18, 192: 18, 185: 18, 307: 17, 300: 17, 294: 17, 292: 17,
260: 17, 257: 17, 245: 17, 232: 17, 231: 17, 167: 17, 328: 16, 312: 16, 287: 16,
265: 16, 246: 16, 242: 16, 238: 16, 237: 16, 233: 16, 227: 16, 207: 16, 204: 16,
200: 16, 178: 16, 330: 15, 285: 15, 273: 15, 254: 15, 214: 15, 195: 15, 173: 15,
389: 14, 326: 14, 309: 14, 308: 14, 286: 14, 253: 14, 243: 14, 218: 14, 190: 14,
360: 13, 317: 13, 299: 13, 297: 13, 291: 13, 278: 13, 277: 13, 275: 13, 274: 13,
264: 13, 259: 13, 258: 13, 220: 13, 208: 13, 473: 12, 419: 12, 346: 12, 345: 12,
324: 12, 319: 12, 295: 12, 290: 12, 288: 12, 276: 12, 272: 12, 271: 12, 268: 12,
262: 12, 248: 12, 247: 12, 230: 12, 224: 12, 223: 12, 221: 12, 475: 11, 391: 11,
357: 11, 338: 11, 335: 11, 329: 11, 306: 11, 305: 11, 304: 11, 298: 11, 289: 11,
269: 11, 263: 11, 251: 11, 250: 11, 249: 11, 239: 11, 234: 11, 229: 11, 213: 11,
211: 11, 547: 10, 504: 10, 415: 10, 411: 10, 396: 10, 386: 10, 385: 10, 376: 10,
374: 10, 364: 10, 348: 10, 340: 10, 336: 10, 333: 10, 332: 10, 322: 10, 315: 10,
314: 10, 303: 10, 301: 10, 284: 10, 283: 10, 266: 10, 256: 10, 600: 9, 515: 9,
462: 9, 440: 9, 408: 9, 352: 9, 342: 9, 341: 9, 334: 9, 321: 9, 316: 9, 296: 9,
270: 9, 261: 9, 241: 9, 240: 9, 696: 8, 637: 8, 616: 8, 593: 8, 588: 8, 514: 8,
513: 8, 480: 8, 478: 8, 455: 8, 452: 8, 449: 8, 446: 8, 444: 8, 443: 8, 439: 8,
406: 8, 398: 8, 382: 8, 363: 8, 358: 8, 351: 8, 327: 8, 320: 8, 282: 8, 280: 8,
279: 8, 756: 7, 644: 7, 579: 7, 574: 7, 565: 7, 555: 7, 521: 7, 497: 7, 491: 7,
481: 7, 463: 7, 461: 7, 456: 7, 433: 7, 428: 7, 426: 7, 418: 7, 414: 7, 412: 7,
410: 7, 404: 7, 397: 7, 395: 7, 387: 7, 384: 7, 383: 7, 380: 7, 379: 7, 378: 7,
377: 7, 372: 7, 371: 7, 368: 7, 366: 7, 365: 7, 362: 7, 359: 7, 356: 7, 354: 7,
350: 7, 344: 7, 331: 7, 313: 7, 310: 7, 302: 7, 267: 7, 1130: 6, 943: 6, 928: 6,
802: 6, 799: 6, 797: 6, 794: 6, 706: 6, 667: 6, 657: 6, 649: 6, 628: 6, 617: 6,
613: 6, 602: 6, 601: 6, 599: 6, 598: 6, 590: 6, 571: 6, 560: 6, 558: 6, 545: 6,
541: 6, 538: 6, 534: 6, 516: 6, 503: 6, 493: 6, 472: 6, 453: 6, 451: 6, 448: 6,
441: 6, 438: 6, 437: 6, 430: 6, 424: 6, 423: 6, 420: 6, 407: 6, 405: 6, 373: 6,
370: 6, 355: 6, 347: 6, 343: 6, 339: 6, 325: 6, 323: 6, 293: 6, 281: 6, 1582: 5,
1300: 5, 1289: 5, 1208: 5, 1044: 5, 984: 5, 951: 5, 921: 5, 897: 5, 892: 5, 834:
5, 827: 5, 808: 5, 806: 5, 792: 5, 739: 5, 682: 5, 674: 5, 671: 5, 654: 5, 648:
5, 647: 5, 638: 5, 633: 5, 629: 5, 620: 5, 594: 5, 591: 5, 564: 5, 563: 5, 554:
5, 552: 5, 551: 5, 540: 5, 536: 5, 528: 5, 523: 5, 520: 5, 519: 5, 517: 5, 510:
5, 509: 5, 506: 5, 500: 5, 499: 5, 494: 5, 490: 5, 487: 5, 483: 5, 471: 5, 464:
5, 454: 5, 450: 5, 447: 5, 436: 5, 432: 5, 427: 5, 413: 5, 403: 5, 399: 5, 394:
5, 390: 5, 381: 5, 369: 5, 367: 5, 349: 5, 337: 5, 235: 5, 3507: 4, 1873: 4,
1548: 4, 1524: 4, 1501: 4, 1400: 4, 1287: 4, 1216: 4, 1200: 4, 1168: 4, 1119: 4,
1080: 4, 1074: 4, 1042: 4, 1028: 4, 1027: 4, 1012: 4, 1004: 4, 1003: 4, 995: 4,
987: 4, 982: 4, 966: 4, 959: 4, 948: 4, 947: 4, 942: 4, 899: 4, 890: 4, 882: 4,
859: 4, 858: 4, 844: 4, 836: 4, 828: 4, 809: 4, 783: 4, 781: 4, 779: 4, 767: 4,
761: 4, 755: 4, 752: 4, 733: 4, 732: 4, 722: 4, 714: 4, 704: 4, 700: 4, 690: 4,
684: 4, 680: 4, 677: 4, 673: 4, 668: 4, 666: 4, 665: 4, 646: 4, 645: 4, 642: 4,
631: 4, 626: 4, 625: 4, 624: 4, 621: 4, 609: 4, 597: 4, 596: 4, 582: 4, 580: 4,
578: 4, 576: 4, 570: 4, 561: 4, 556: 4, 550: 4, 549: 4, 548: 4, 539: 4, 537: 4,
535: 4, 531: 4, 525: 4, 524: 4, 508: 4, 502: 4, 501: 4, 498: 4, 492: 4, 489: 4,
488: 4, 484: 4, 477: 4, 469: 4, 466: 4, 465: 4, 460: 4, 442: 4, 434: 4, 422: 4,
421: 4, 417: 4, 416: 4, 402: 4, 401: 4, 392: 4, 388: 4, 361: 4, 353: 4, 311: 4,
4120: 3, 3573: 3, 3261: 3, 3171: 3, 2755: 3, 2569: 3, 2497: 3, 2414: 3, 2388: 3,
2117: 3, 2037: 3, 1985: 3, 1948: 3, 1936: 3, 1930: 3, 1900: 3, 1840: 3, 1830: 3,
1813: 3, 1665: 3, 1653: 3, 1650: 3, 1640: 3, 1619: 3, 1564: 3, 1560: 3, 1532: 3,
1521: 3, 1520: 3, 1505: 3, 1437: 3, 1435: 3, 1388: 3, 1376: 3, 1367: 3, 1366: 3,
1362: 3, 1357: 3, 1355: 3, 1351: 3, 1340: 3, 1328: 3, 1311: 3, 1279: 3, 1277: 3,
1275: 3, 1266: 3, 1264: 3, 1256: 3, 1231: 3, 1219: 3, 1214: 3, 1212: 3, 1201: 3,
1181: 3, 1180: 3, 1154: 3, 1139: 3, 1099: 3, 1095: 3, 1090: 3, 1088: 3, 1087: 3,
1083: 3, 1035: 3, 1034: 3, 1025: 3, 1022: 3, 1019: 3, 1002: 3, 988: 3, 981: 3,
980: 3, 973: 3, 968: 3, 949: 3, 944: 3, 931: 3, 925: 3, 915: 3, 913: 3, 910: 3,
900: 3, 898: 3, 893: 3, 885: 3, 881: 3, 872: 3, 869: 3, 866: 3, 865: 3, 862: 3,
857: 3, 849: 3, 846: 3, 845: 3, 842: 3, 835: 3, 833: 3, 825: 3, 824: 3, 820: 3,
818: 3, 817: 3, 813: 3, 811: 3, 810: 3, 805: 3, 803: 3, 793: 3, 791: 3, 790: 3,
786: 3, 771: 3, 766: 3, 764: 3, 762: 3, 759: 3, 751: 3, 746: 3, 743: 3, 740: 3,
738: 3, 730: 3, 719: 3, 718: 3, 712: 3, 708: 3, 701: 3, 698: 3, 693: 3, 691: 3,
687: 3, 681: 3, 676: 3, 664: 3, 663: 3, 662: 3, 661: 3, 659: 3, 658: 3, 653: 3,
652: 3, 641: 3, 632: 3, 630: 3, 622: 3, 618: 3, 615: 3, 604: 3, 595: 3, 589: 3,
587: 3, 584: 3, 575: 3, 573: 3, 572: 3, 562: 3, 559: 3, 557: 3, 546: 3, 542: 3,
530: 3, 511: 3, 496: 3, 495: 3, 486: 3, 485: 3, 474: 3, 459: 3, 458: 3, 435: 3,
431: 3, 409: 3, 375: 3, 10032: 2, 8060: 2, 7970: 2, 6206: 2, 6133: 2, 6061: 2,
6035: 2, 5774: 2, 5606: 2, 5456: 2, 5122: 2, 5073: 2, 4937: 2, 4902: 2, 4812: 2,
4400: 2, 4350: 2, 4207: 2, 3910: 2, 3803: 2, 3777: 2, 3766: 2, 3728: 2, 3695: 2,
3653: 2, 3551: 2, 3520: 2, 3420: 2, 3342: 2, 3336: 2, 3314: 2, 3278: 2, 3253: 2,
3248: 2, 3197: 2, 3174: 2, 3029: 2, 3017: 2, 3008: 2, 2999: 2, 2937: 2, 2925: 2,
2863: 2, 2841: 2, 2817: 2, 2758: 2, 2741: 2, 2732: 2, 2711: 2, 2667: 2, 2644: 2,
2594: 2, 2583: 2, 2577: 2, 2560: 2, 2548: 2, 2546: 2, 2543: 2, 2533: 2, 2509: 2,
2505: 2, 2495: 2, 2484: 2, 2479: 2, 2423: 2, 2410: 2, 2405: 2, 2385: 2, 2353: 2,
2352: 2, 2316: 2, 2315: 2, 2309: 2, 2290: 2, 2289: 2, 2281: 2, 2276: 2, 2251: 2,
2246: 2, 2240: 2, 2238: 2, 2233: 2, 2220: 2, 2217: 2, 2215: 2, 2206: 2, 2180: 2,
2163: 2, 2132: 2, 2116: 2, 2104: 2, 2098: 2, 2074: 2, 2066: 2, 2058: 2, 2046: 2,
2035: 2, 2031: 2, 2014: 2, 1995: 2, 1982: 2, 1977: 2, 1976: 2, 1972: 2, 1956: 2,
1955: 2, 1946: 2, 1941: 2, 1940: 2, 1926: 2, 1917: 2, 1914: 2, 1910: 2, 1908: 2,
1894: 2, 1879: 2, 1863: 2, 1857: 2, 1832: 2, 1831: 2, 1829: 2, 1823: 2, 1819: 2,
1818: 2, 1812: 2, 1795: 2, 1778: 2, 1776: 2, 1759: 2, 1757: 2, 1750: 2, 1749: 2,
1737: 2, 1712: 2, 1706: 2, 1693: 2, 1691: 2, 1687: 2, 1683: 2, 1672: 2, 1657: 2,
1656: 2, 1649: 2, 1645: 2, 1634: 2, 1631: 2, 1624: 2, 1622: 2, 1618: 2, 1616: 2,
1608: 2, 1595: 2, 1584: 2, 1579: 2, 1556: 2, 1551: 2, 1550: 2, 1546: 2, 1544: 2,
1530: 2, 1523: 2, 1518: 2, 1515: 2, 1509: 2, 1500: 2, 1498: 2, 1492: 2, 1488: 2,
1485: 2, 1467: 2, 1466: 2, 1462: 2, 1448: 2, 1446: 2, 1443: 2, 1442: 2, 1439: 2,
1431: 2, 1423: 2, 1422: 2, 1418: 2, 1415: 2, 1411: 2, 1410: 2, 1404: 2, 1402: 2,
1371: 2, 1370: 2, 1368: 2, 1356: 2, 1349: 2, 1346: 2, 1339: 2, 1336: 2, 1330: 2,
1325: 2, 1323: 2, 1322: 2, 1321: 2, 1315: 2, 1314: 2, 1313: 2, 1312: 2, 1305: 2,
1302: 2, 1295: 2, 1288: 2, 1283: 2, 1274: 2, 1268: 2, 1259: 2, 1252: 2, 1248: 2,
1247: 2, 1246: 2, 1236: 2, 1232: 2, 1229: 2, 1228: 2, 1225: 2, 1222: 2, 1220: 2,
1215: 2, 1207: 2, 1205: 2, 1198: 2, 1197: 2, 1194: 2, 1193: 2, 1191: 2, 1178: 2,
1175: 2, 1170: 2, 1165: 2, 1161: 2, 1160: 2, 1158: 2, 1153: 2, 1152: 2, 1149: 2,
1144: 2, 1143: 2, 1141: 2, 1136: 2, 1134: 2, 1133: 2, 1129: 2, 1128: 2, 1116: 2,
1115: 2, 1111: 2, 1110: 2, 1109: 2, 1106: 2, 1102: 2, 1089: 2, 1084: 2, 1082: 2,
1077: 2, 1075: 2, 1070: 2, 1066: 2, 1064: 2, 1060: 2, 1059: 2, 1053: 2, 1052: 2,
1051: 2, 1050: 2, 1039: 2, 1036: 2, 1020: 2, 1018: 2, 1010: 2, 998: 2, 994: 2,
992: 2, 989: 2, 985: 2, 983: 2, 979: 2, 977: 2, 976: 2, 975: 2, 972: 2, 971: 2,
970: 2, 964: 2, 963: 2, 962: 2, 955: 2, 954: 2, 953: 2, 945: 2, 941: 2, 939: 2,
936: 2, 935: 2, 933: 2, 932: 2, 930: 2, 919: 2, 917: 2, 916: 2, 914: 2, 911: 2,
908: 2, 907: 2, 905: 2, 903: 2, 901: 2, 895: 2, 891: 2, 888: 2, 887: 2, 886: 2,
878: 2, 867: 2, 863: 2, 854: 2, 853: 2, 852: 2, 850: 2, 843: 2, 841: 2, 839: 2,
832: 2, 829: 2, 826: 2, 823: 2, 816: 2, 815: 2, 814: 2, 812: 2, 804: 2, 798: 2,
796: 2, 795: 2, 789: 2, 784: 2, 782: 2, 776: 2, 772: 2, 770: 2, 769: 2, 768: 2,
763: 2, 760: 2, 758: 2, 757: 2, 745: 2, 737: 2, 734: 2, 731: 2, 728: 2, 727: 2,
725: 2, 724: 2, 723: 2, 720: 2, 715: 2, 710: 2, 709: 2, 707: 2, 699: 2, 697: 2,
695: 2, 694: 2, 692: 2, 688: 2, 686: 2, 683: 2, 675: 2, 672: 2, 670: 2, 669: 2,
655: 2, 650: 2, 643: 2, 640: 2, 635: 2, 627: 2, 612: 2, 610: 2, 606: 2, 605: 2,
603: 2, 586: 2, 585: 2, 583: 2, 577: 2, 568: 2, 567: 2, 553: 2, 543: 2, 533: 2,
529: 2, 526: 2, 522: 2, 518: 2, 512: 2, 507: 2, 505: 2, 482: 2, 479: 2, 476: 2,
470: 2, 468: 2, 429: 2, 425: 2, 400: 2, 393: 2, 318: 2, 153589: 1, 119831: 1,
82952: 1, 69070: 1, 67224: 1, 67068: 1, 65581: 1, 63923: 1, 63550: 1, 55154: 1,
54139: 1, 49627: 1, 49462: 1, 47413: 1, 46783: 1, 45091: 1, 43107: 1, 42996: 1,
42917: 1, 41283: 1, 40666: 1, 40300: 1, 40211: 1, 38382: 1, 38114: 1, 37632: 1,
36510: 1, 36387: 1, 36354: 1, 35244: 1, 34707: 1, 34189: 1, 33782: 1, 33460: 1,
32619: 1, 31788: 1, 29454: 1, 28480: 1, 28161: 1, 28033: 1, 26479: 1, 26310: 1,
25831: 1, 25674: 1, 25038: 1, 25034: 1, 25032: 1, 24852: 1, 24783: 1, 24569: 1,
24311: 1, 24068: 1, 23765: 1, 23152: 1, 22403: 1, 22367: 1, 21857: 1, 21285: 1,
21182: 1, 21148: 1, 20842: 1, 20736: 1, 20654: 1, 20617: 1, 20516: 1, 20164: 1,
19776: 1, 19611: 1, 19579: 1, 19275: 1, 19162: 1, 19145: 1, 19053: 1, 18979: 1,
18945: 1, 18917: 1, 18813: 1, 18689: 1, 18248: 1, 18106: 1, 17946: 1, 17898: 1,
17801: 1, 17799: 1, 17669: 1, 17558: 1, 17510: 1, 17445: 1, 17275: 1, 17179: 1,
17149: 1, 17148: 1, 17040: 1, 16987: 1, 16969: 1, 16949: 1, 16664: 1, 16553: 1,
16395: 1, 16113: 1, 16011: 1, 15866: 1, 15764: 1, 15669: 1, 15665: 1, 15502: 1,
15479: 1, 15413: 1, 15412: 1, 15283: 1, 14952: 1, 14774: 1, 14772: 1, 14771: 1,
14766: 1, 14606: 1, 14605: 1, 14556: 1, 14374: 1, 14283: 1, 14164: 1, 13950: 1,
13918: 1, 13722: 1, 13711: 1, 13707: 1, 13593: 1, 13567: 1, 13541: 1, 13515: 1,
13374: 1, 13355: 1, 13173: 1, 13145: 1, 13024: 1, 12970: 1, 12928: 1, 12927: 1,
12902: 1, 12864: 1, 12826: 1, 12798: 1, 12787: 1, 12721: 1, 12544: 1, 12536: 1,
12505: 1, 12490: 1, 12459: 1, 12441: 1, 12429: 1, 12422: 1, 12407: 1, 12363: 1,
12355: 1, 12295: 1, 12164: 1, 12155: 1, 12133: 1, 12077: 1, 12071: 1, 11972: 1,
11952: 1, 11895: 1, 11866: 1, 11857: 1, 11849: 1, 11801: 1, 11795: 1, 11753: 1,
11687: 1, 11651: 1, 11391: 1, 11351: 1, 11334: 1, 11262: 1, 11230: 1, 11155: 1,
11075: 1, 11067: 1, 10905: 1, 10746: 1, 10714: 1, 10707: 1, 10679: 1, 10672: 1,
10671: 1, 10624: 1, 10593: 1, 10559: 1, 10557: 1, 10519: 1, 10474: 1, 10460: 1,
10438: 1, 10374: 1, 10326: 1, 10324: 1, 10305: 1, 10288: 1, 10281: 1, 10210: 1,
10206: 1, 10159: 1, 10108: 1, 10070: 1, 10040: 1, 10036: 1, 9945: 1, 9898: 1,
9884: 1, 9800: 1, 9759: 1, 9741: 1, 9678: 1, 9616: 1, 9556: 1, 9544: 1, 9496: 1,
9435: 1, 9406: 1, 9397: 1, 9392: 1, 9370: 1, 9337: 1, 9308: 1, 9298: 1, 9185: 1,
9176: 1, 9168: 1, 9126: 1, 9113: 1, 9063: 1, 8998: 1, 8961: 1, 8958: 1, 8950: 1,
8932: 1, 8929: 1, 8914: 1, 8899: 1, 8885: 1, 8875: 1, 8843: 1, 8796: 1, 8750: 1,
8727: 1, 8653: 1, 8555: 1, 8552: 1, 8512: 1, 8491: 1, 8411: 1, 8399: 1, 8396: 1,
8389: 1, 8333: 1, 8321: 1, 8320: 1, 8293: 1, 8263: 1, 8250: 1, 8242: 1, 8203: 1,
8202: 1, 8179: 1, 8163: 1, 8151: 1, 8144: 1, 8097: 1, 8087: 1, 8057: 1, 8042: 1,
8026: 1, 7937: 1, 7905: 1, 7895: 1, 7883: 1, 7870: 1, 7837: 1, 7832: 1, 7790: 1,
7732: 1, 7723: 1, 7720: 1, 7711: 1, 7701: 1, 7679: 1, 7673: 1, 7668: 1, 7640: 1,
7616: 1, 7556: 1, 7545: 1, 7542: 1, 7522: 1, 7471: 1, 7447: 1, 7399: 1, 7394: 1,
7364: 1, 7339: 1, 7329: 1, 7317: 1, 7287: 1, 7283: 1, 7273: 1, 7268: 1, 7243: 1,
7242: 1, 7224: 1, 7161: 1, 7153: 1, 7152: 1, 7144: 1, 7104: 1, 7048: 1, 7011: 1,
6995: 1, 6988: 1, 6981: 1, 6962: 1, 6947: 1, 6945: 1, 6934: 1, 6898: 1, 6897: 1,
6894: 1, 6892: 1, 6864: 1, 6858: 1, 6829: 1, 6828: 1, 6803: 1, 6768: 1, 6762: 1,
6745: 1, 6722: 1, 6698: 1, 6693: 1, 6678: 1, 6658: 1, 6643: 1, 6637: 1, 6605: 1,
6598: 1, 6591: 1, 6584: 1, 6565: 1, 6564: 1, 6549: 1, 6535: 1, 6531: 1, 6520: 1,
6509: 1, 6496: 1, 6489: 1, 6481: 1, 6469: 1, 6416: 1, 6382: 1, 6379: 1, 6372: 1,
6362: 1, 6353: 1, 6337: 1, 6306: 1, 6303: 1, 6278: 1, 6263: 1, 6218: 1, 6198: 1,
6188: 1, 6176: 1, 6156: 1, 6139: 1, 6100: 1, 6093: 1, 6066: 1, 6063: 1, 6052: 1,
6029: 1, 6027: 1, 6023: 1, 6019: 1, 6003: 1, 5998: 1, 5987: 1, 5985: 1, 5960: 1,
5940: 1, 5934: 1, 5932: 1, 5925: 1, 5924: 1, 5908: 1, 5861: 1, 5831: 1, 5820: 1,
5817: 1, 5811: 1, 5806: 1, 5796: 1, 5715: 1, 5702: 1, 5693: 1, 5680: 1, 5669: 1,
5640: 1, 5637: 1, 5631: 1, 5624: 1, 5618: 1, 5617: 1, 5596: 1, 5590: 1, 5586: 1,
5574: 1, 5522: 1, 5521: 1, 5520: 1, 5510: 1, 5494: 1, 5491: 1, 5484: 1, 5474: 1,
5455: 1, 5436: 1, 5435: 1, 5432: 1, 5417: 1, 5409: 1, 5408: 1, 5404: 1, 5401: 1,
5385: 1, 5382: 1, 5371: 1, 5365: 1, 5344: 1, 5340: 1, 5339: 1, 5317: 1, 5288: 1,
5272: 1, 5238: 1, 5223: 1, 5199: 1, 5182: 1, 5168: 1, 5157: 1, 5142: 1, 5133: 1,
5130: 1, 5112: 1, 5107: 1, 5099: 1, 5094: 1, 5089: 1, 5080: 1, 5046: 1, 5044: 1,
5041: 1, 5035: 1, 5024: 1, 5019: 1, 5017: 1, 4985: 1, 4971: 1, 4969: 1, 4968: 1,
4961: 1, 4953: 1, 4939: 1, 4938: 1, 4936: 1, 4934: 1, 4927: 1, 4916: 1, 4914: 1,
4910: 1, 4896: 1, 4890: 1, 4847: 1, 4831: 1, 4829: 1, 4822: 1, 4807: 1, 4801: 1,
4790: 1, 4786: 1, 4775: 1, 4773: 1, 4772: 1, 4771: 1, 4767: 1, 4764: 1, 4760: 1,
4732: 1, 4700: 1, 4697: 1, 4683: 1, 4661: 1, 4654: 1, 4616: 1, 4607: 1, 4603: 1,
4582: 1, 4562: 1, 4557: 1, 4555: 1, 4550: 1, 4539: 1, 4535: 1, 4532: 1, 4524: 1,
4523: 1, 4518: 1, 4510: 1, 4507: 1, 4504: 1, 4503: 1, 4494: 1, 4488: 1, 4483: 1,
4468: 1, 4464: 1, 4457: 1, 4454: 1, 4446: 1, 4433: 1, 4424: 1, 4406: 1, 4393: 1,
4380: 1, 4373: 1, 4367: 1, 4349: 1, 4338: 1, 4336: 1, 4334: 1, 4333: 1, 4331: 1,
4328: 1, 4326: 1, 4318: 1, 4314: 1, 4313: 1, 4311: 1, 4307: 1, 4290: 1, 4273: 1,
4263: 1, 4262: 1, 4255: 1, 4234: 1, 4231: 1, 4227: 1, 4224: 1, 4217: 1, 4211: 1,
4210: 1, 4206: 1, 4205: 1, 4202: 1, 4181: 1, 4179: 1, 4173: 1, 4170: 1, 4168: 1,
4155: 1, 4148: 1, 4139: 1, 4132: 1, 4124: 1, 4110: 1, 4095: 1, 4094: 1, 4093: 1,
4092: 1, 4091: 1, 4082: 1, 4080: 1, 4073: 1, 4068: 1, 4062: 1, 4058: 1, 4056: 1,
4054: 1, 4053: 1, 4043: 1, 4032: 1, 4029: 1, 4026: 1, 4005: 1, 3990: 1, 3988: 1,
3987: 1, 3963: 1, 3949: 1, 3937: 1, 3936: 1, 3928: 1, 3919: 1, 3917: 1, 3901: 1,
3896: 1, 3892: 1, 3891: 1, 3888: 1, 3882: 1, 3870: 1, 3859: 1, 3852: 1, 3848: 1,
3846: 1, 3844: 1, 3838: 1, 3834: 1, 3826: 1, 3810: 1, 3809: 1, 3799: 1, 3793: 1,
3790: 1, 3789: 1, 3785: 1, 3782: 1, 3779: 1, 3778: 1, 3772: 1, 3764: 1, 3763: 1,
3758: 1, 3745: 1, 3741: 1, 3735: 1, 3731: 1, 3725: 1, 3724: 1, 3721: 1, 3720: 1,
3718: 1, 3713: 1, 3702: 1, 3701: 1, 3700: 1, 3690: 1, 3683: 1, 3674: 1, 3670: 1,
3664: 1, 3658: 1, 3655: 1, 3648: 1, 3647: 1, 3645: 1, 3631: 1, 3627: 1, 3625: 1,
3620: 1, 3616: 1, 3610: 1, 3607: 1, 3604: 1, 3599: 1, 3592: 1, 3589: 1, 3588: 1,
3587: 1, 3580: 1, 3577: 1, 3576: 1, 3564: 1, 3562: 1, 3557: 1, 3553: 1, 3549: 1,
3546: 1, 3545: 1, 3538: 1, 3525: 1, 3515: 1, 3512: 1, 3510: 1, 3506: 1, 3505: 1,
3500: 1, 3498: 1, 3496: 1, 3488: 1, 3487: 1, 3479: 1, 3474: 1, 3461: 1, 3460: 1,
3452: 1, 3445: 1, 3441: 1, 3432: 1, 3430: 1, 3421: 1, 3419: 1, 3416: 1, 3408: 1,
3399: 1, 3398: 1, 3397: 1, 3395: 1, 3393: 1, 3392: 1, 3391: 1, 3390: 1, 3376: 1,
3374: 1, 3372: 1, 3369: 1, 3366: 1, 3364: 1, 3359: 1, 3347: 1, 3345: 1, 3337: 1,
3318: 1, 3315: 1, 3311: 1, 3302: 1, 3297: 1, 3293: 1, 3289: 1, 3279: 1, 3277: 1,
3271: 1, 3260: 1, 3254: 1, 3252: 1, 3245: 1, 3243: 1, 3239: 1, 3221: 1, 3217: 1,
3211: 1, 3209: 1, 3204: 1, 3203: 1, 3188: 1, 3183: 1, 3178: 1, 3170: 1, 3166: 1,
3155: 1, 3154: 1, 3153: 1, 3152: 1, 3150: 1, 3149: 1, 3146: 1, 3143: 1, 3137: 1,
3131: 1, 3115: 1, 3114: 1, 3109: 1, 3103: 1, 3099: 1, 3097: 1, 3096: 1, 3094: 1,
3090: 1, 3088: 1, 3087: 1, 3085: 1, 3082: 1, 3079: 1, 3077: 1, 3076: 1, 3072: 1,
3060: 1, 3055: 1, 3053: 1, 3046: 1, 3041: 1, 3038: 1, 3035: 1, 3032: 1, 3020: 1,
3009: 1, 3006: 1, 3004: 1, 3001: 1, 2998: 1, 2992: 1, 2991: 1, 2983: 1, 2978: 1,
2975: 1, 2972: 1, 2969: 1, 2968: 1, 2964: 1, 2956: 1, 2954: 1, 2952: 1, 2950: 1,
2949: 1, 2944: 1, 2940: 1, 2936: 1, 2929: 1, 2918: 1, 2905: 1, 2902: 1, 2901: 1,
2896: 1, 2887: 1, 2880: 1, 2879: 1, 2873: 1, 2862: 1, 2855: 1, 2851: 1, 2836: 1,
2835: 1, 2831: 1, 2824: 1, 2820: 1, 2819: 1, 2811: 1, 2799: 1, 2792: 1, 2788: 1,
2762: 1, 2760: 1, 2753: 1, 2752: 1, 2746: 1, 2740: 1, 2736: 1, 2730: 1, 2728: 1,
2723: 1, 2718: 1, 2717: 1, 2709: 1, 2703: 1, 2700: 1, 2697: 1, 2692: 1, 2683: 1,
2681: 1, 2678: 1, 2671: 1, 2668: 1, 2666: 1, 2665: 1, 2664: 1, 2663: 1, 2661: 1,
2660: 1, 2659: 1, 2657: 1, 2655: 1, 2649: 1, 2634: 1, 2632: 1, 2631: 1, 2629: 1,
2628: 1, 2626: 1, 2622: 1, 2620: 1, 2618: 1, 2616: 1, 2614: 1, 2601: 1, 2599: 1,
2597: 1, 2590: 1, 2587: 1, 2584: 1, 2576: 1, 2574: 1, 2573: 1, 2566: 1, 2556: 1,
2555: 1, 2553: 1, 2541: 1, 2539: 1, 2531: 1, 2518: 1, 2516: 1, 2513: 1, 2507: 1,
2498: 1, 2493: 1, 2491: 1, 2485: 1, 2481: 1, 2474: 1, 2466: 1, 2457: 1, 2453: 1,
2447: 1, 2445: 1, 2435: 1, 2434: 1, 2432: 1, 2430: 1, 2429: 1, 2426: 1, 2425: 1,
2416: 1, 2412: 1, 2411: 1, 2402: 1, 2398: 1, 2395: 1, 2394: 1, 2392: 1, 2390: 1,
2389: 1, 2386: 1, 2378: 1, 2376: 1, 2372: 1, 2369: 1, 2367: 1, 2366: 1, 2365: 1,
2362: 1, 2357: 1, 2350: 1, 2349: 1, 2336: 1, 2334: 1, 2333: 1, 2330: 1, 2327: 1,
2325: 1, 2324: 1, 2322: 1, 2321: 1, 2318: 1, 2308: 1, 2307: 1, 2305: 1, 2304: 1,
2302: 1, 2301: 1, 2284: 1, 2283: 1, 2271: 1, 2269: 1, 2262: 1, 2245: 1, 2241: 1,
2237: 1, 2232: 1, 2229: 1, 2226: 1, 2222: 1, 2219: 1, 2218: 1, 2214: 1, 2213: 1,
2209: 1, 2207: 1, 2205: 1, 2203: 1, 2200: 1, 2197: 1, 2194: 1, 2191: 1, 2178: 1,
2177: 1, 2173: 1, 2172: 1, 2169: 1, 2167: 1, 2165: 1, 2161: 1, 2156: 1, 2154: 1,
2153: 1, 2152: 1, 2151: 1, 2150: 1, 2148: 1, 2145: 1, 2143: 1, 2141: 1, 2136: 1,
2134: 1, 2131: 1, 2129: 1, 2126: 1, 2121: 1, 2119: 1, 2118: 1, 2113: 1, 2111: 1,
2109: 1, 2107: 1, 2105: 1, 2099: 1, 2094: 1, 2089: 1, 2088: 1, 2087: 1, 2086: 1,
2084: 1, 2079: 1, 2077: 1, 2076: 1, 2073: 1, 2071: 1, 2069: 1, 2067: 1, 2065: 1,
2061: 1, 2060: 1, 2057: 1, 2054: 1, 2052: 1, 2049: 1, 2047: 1, 2043: 1, 2036: 1,
2034: 1, 2030: 1, 2025: 1, 2024: 1, 2023: 1, 2020: 1, 2019: 1, 2013: 1, 2012: 1,
2009: 1, 2006: 1, 2004: 1, 1999: 1, 1998: 1, 1997: 1, 1996: 1, 1990: 1, 1989: 1,
1988: 1, 1986: 1, 1984: 1, 1978: 1, 1975: 1, 1971: 1, 1964: 1, 1963: 1, 1960: 1,
1959: 1, 1958: 1, 1954: 1, 1953: 1, 1952: 1, 1950: 1, 1949: 1, 1942: 1, 1937: 1,
1935: 1, 1920: 1, 1915: 1, 1913: 1, 1912: 1, 1906: 1, 1904: 1, 1902: 1, 1901: 1,
1893: 1, 1891: 1, 1890: 1, 1888: 1, 1887: 1, 1886: 1, 1885: 1, 1880: 1, 1877: 1,
1875: 1, 1870: 1, 1868: 1, 1867: 1, 1866: 1, 1864: 1, 1862: 1, 1856: 1, 1855: 1,
1852: 1, 1849: 1, 1848: 1, 1846: 1, 1845: 1, 1843: 1, 1841: 1, 1838: 1, 1833: 1,
1828: 1, 1825: 1, 1824: 1, 1817: 1, 1815: 1, 1806: 1, 1802: 1, 1798: 1, 1797: 1,
1796: 1, 1794: 1, 1793: 1, 1791: 1, 1784: 1, 1782: 1, 1774: 1, 1771: 1, 1768: 1,
1767: 1, 1765: 1, 1763: 1, 1760: 1, 1758: 1, 1756: 1, 1755: 1, 1754: 1, 1751: 1,
1748: 1, 1747: 1, 1746: 1, 1744: 1, 1742: 1, 1741: 1, 1739: 1, 1736: 1, 1735: 1,
1734: 1, 1733: 1, 1729: 1, 1728: 1, 1727: 1, 1726: 1, 1725: 1, 1724: 1, 1721: 1,
1718: 1, 1709: 1, 1708: 1, 1705: 1, 1704: 1, 1698: 1, 1697: 1, 1696: 1, 1690: 1,
1682: 1, 1681: 1, 1680: 1, 1679: 1, 1676: 1, 1675: 1, 1674: 1, 1673: 1, 1670: 1,
1669: 1, 1667: 1, 1666: 1, 1664: 1, 1658: 1, 1651: 1, 1648: 1, 1647: 1, 1646: 1,
1644: 1, 1643: 1, 1642: 1, 1641: 1, 1636: 1, 1635: 1, 1632: 1, 1630: 1, 1629: 1,
1627: 1, 1620: 1, 1617: 1, 1615: 1, 1614: 1, 1610: 1, 1609: 1, 1607: 1, 1604: 1,
1603: 1, 1600: 1, 1598: 1, 1596: 1, 1592: 1, 1591: 1, 1590: 1, 1589: 1, 1588: 1,
1587: 1, 1585: 1, 1580: 1, 1577: 1, 1576: 1, 1575: 1, 1574: 1, 1572: 1, 1571: 1,
1567: 1, 1565: 1, 1557: 1, 1554: 1, 1553: 1, 1552: 1, 1545: 1, 1542: 1, 1541: 1,
1537: 1, 1536: 1, 1535: 1, 1534: 1, 1531: 1, 1529: 1, 1527: 1, 1525: 1, 1522: 1,
1519: 1, 1517: 1, 1514: 1, 1513: 1, 1512: 1, 1510: 1, 1508: 1, 1507: 1, 1506: 1,
1502: 1, 1499: 1, 1493: 1, 1489: 1, 1487: 1, 1481: 1, 1480: 1, 1472: 1, 1471: 1,
1470: 1, 1469: 1, 1468: 1, 1465: 1, 1461: 1, 1456: 1, 1455: 1, 1453: 1, 1452: 1,
1451: 1, 1449: 1, 1445: 1, 1438: 1, 1436: 1, 1433: 1, 1432: 1, 1428: 1, 1425: 1,
1421: 1, 1420: 1, 1419: 1, 1416: 1, 1414: 1, 1409: 1, 1407: 1, 1406: 1, 1403: 1,
1399: 1, 1397: 1, 1395: 1, 1392: 1, 1391: 1, 1389: 1, 1387: 1, 1386: 1, 1381: 1,
1380: 1, 1379: 1, 1378: 1, 1377: 1, 1375: 1, 1374: 1, 1369: 1, 1363: 1, 1361: 1,
1360: 1, 1359: 1, 1358: 1, 1352: 1, 1348: 1, 1345: 1, 1344: 1, 1337: 1, 1335: 1,
1332: 1, 1331: 1, 1329: 1, 1327: 1, 1326: 1, 1319: 1, 1316: 1, 1306: 1, 1304: 1,
1299: 1, 1298: 1, 1297: 1, 1293: 1, 1292: 1, 1291: 1, 1290: 1, 1286: 1, 1282: 1,
1281: 1, 1276: 1, 1273: 1, 1272: 1, 1267: 1, 1263: 1, 1262: 1, 1261: 1, 1260: 1,
1254: 1, 1253: 1, 1249: 1, 1245: 1, 1244: 1, 1242: 1, 1239: 1, 1237: 1, 1234: 1,
1233: 1, 1226: 1, 1224: 1, 1223: 1, 1221: 1, 1218: 1, 1213: 1, 1211: 1, 1209: 1,
1203: 1, 1202: 1, 1199: 1, 1196: 1, 1195: 1, 1192: 1, 1189: 1, 1188: 1, 1186: 1,
1182: 1, 1179: 1, 1176: 1, 1172: 1, 1171: 1, 1167: 1, 1164: 1, 1163: 1, 1162: 1,
1157: 1, 1155: 1, 1150: 1, 1148: 1, 1146: 1, 1140: 1, 1138: 1, 1137: 1, 1132: 1,
1131: 1, 1127: 1, 1126: 1, 1125: 1, 1124: 1, 1118: 1, 1117: 1, 1114: 1, 1113: 1,
1112: 1, 1107: 1, 1105: 1, 1104: 1, 1103: 1, 1098: 1, 1097: 1, 1096: 1, 1094: 1,
1093: 1, 1091: 1, 1086: 1, 1085: 1, 1081: 1, 1079: 1, 1076: 1, 1072: 1, 1071: 1,
1069: 1, 1068: 1, 1067: 1, 1065: 1, 1061: 1, 1058: 1, 1056: 1, 1055: 1, 1049: 1,
1048: 1, 1047: 1, 1046: 1, 1045: 1, 1037: 1, 1029: 1, 1026: 1, 1024: 1, 1023: 1,
1017: 1, 1016: 1, 1015: 1, 1013: 1, 1011: 1, 1007: 1, 1005: 1, 999: 1, 993: 1,
991: 1, 990: 1, 974: 1, 969: 1, 967: 1, 965: 1, 961: 1, 960: 1, 958: 1, 956: 1,
952: 1, 946: 1, 940: 1, 938: 1, 937: 1, 934: 1, 929: 1, 927: 1, 922: 1, 920: 1,
918: 1, 909: 1, 906: 1, 904: 1, 902: 1, 896: 1, 884: 1, 883: 1, 880: 1, 879: 1,
877: 1, 876: 1, 875: 1, 874: 1, 873: 1, 871: 1, 864: 1, 861: 1, 860: 1, 856: 1,
855: 1, 847: 1, 838: 1, 837: 1, 831: 1, 830: 1, 822: 1, 821: 1, 819: 1, 807: 1,
800: 1, 788: 1, 787: 1, 785: 1, 780: 1, 778: 1, 773: 1, 765: 1, 754: 1, 753: 1,
750: 1, 749: 1, 744: 1, 742: 1, 741: 1, 736: 1, 729: 1, 726: 1, 721: 1, 717: 1,
716: 1, 705: 1, 703: 1, 702: 1, 689: 1, 679: 1, 678: 1, 656: 1, 651: 1, 639: 1,
634: 1, 623: 1, 619: 1, 614: 1, 611: 1, 607: 1, 581: 1, 569: 1, 566: 1, 544: 1,
532: 1, 527: 1, 467: 1, 457: 1, 445: 1})
```

6 Training logistic regression model with gene feature

[135]:

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

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: 1.3471857619819732
For values of alpha =  0.0001 The log loss is: 1.268583740473084
For values of alpha =  0.001 The log loss is: 1.280287233714018
For values of alpha =  0.01 The log loss is: 1.399949375879958
For values of alpha =  0.1 The log loss is: 1.5050602179577912
For values of alpha =  1 The log loss is: 1.6487259328758859
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6529251757939964
For values of best alpha =  0.0001 The cross validation log loss is:
1.268583740473084
For values of best alpha =  0.0001 The test log loss is: 1.1836703982084613
```

[136]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    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
```

[137]:
```
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")
```

```
97.297 % of word of test data appeared in train data
97.761 % of word of Cross Validation appeared in train data
```

7 Machine learning models

[138]:

```
#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)
```

[139]:

```
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)
```

[140]:

```
# 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 = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    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")
```

7.1 Stacking three types of features

[141]:

```
# 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']))
```

[142]:

```
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, 56217)
(number of data points * number of features) in test data =  (665, 56217)
(number of data points * number of features) in cross validation data = (532,
56217)
```

[143]:

```
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))
```

[144]:

```
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)
```

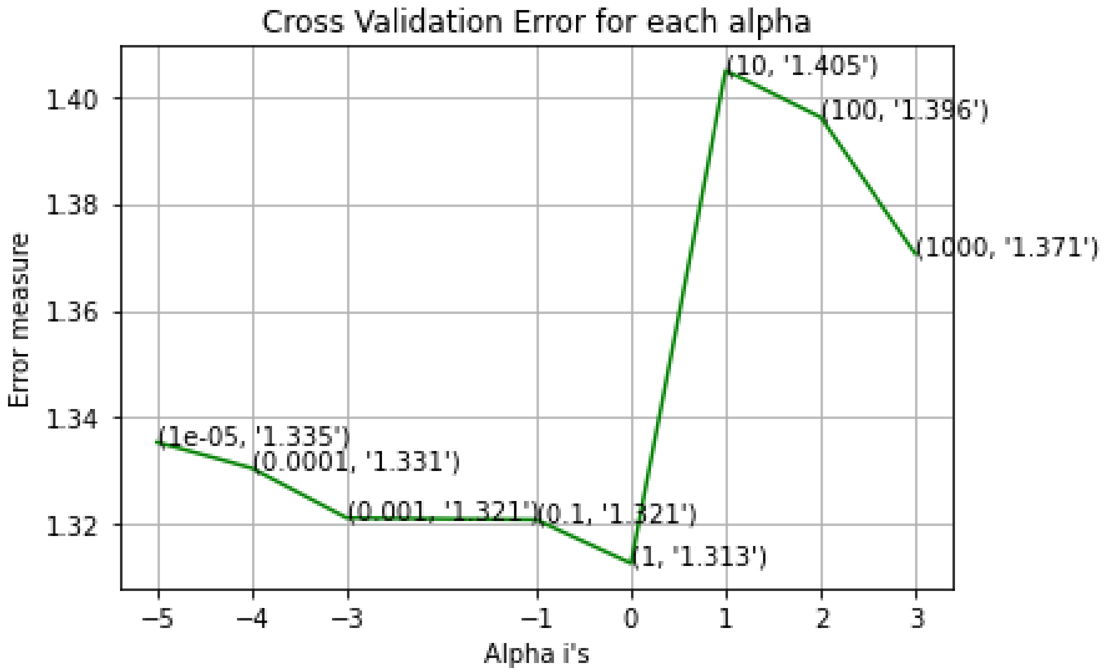
8 Naive bayes

[145]:

```
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 : 1.3353169598157526
for alpha = 0.0001
Log Loss : 1.3305172673208208
for alpha = 0.001
Log Loss : 1.321218056709718
for alpha = 0.1
Log Loss : 1.3208902216598462
for alpha = 1
Log Loss : 1.3127574719118553
for alpha = 10
Log Loss : 1.4048565148497405
for alpha = 100
Log Loss : 1.3960643368672683
for alpha = 1000
Log Loss : 1.370558024032899
```

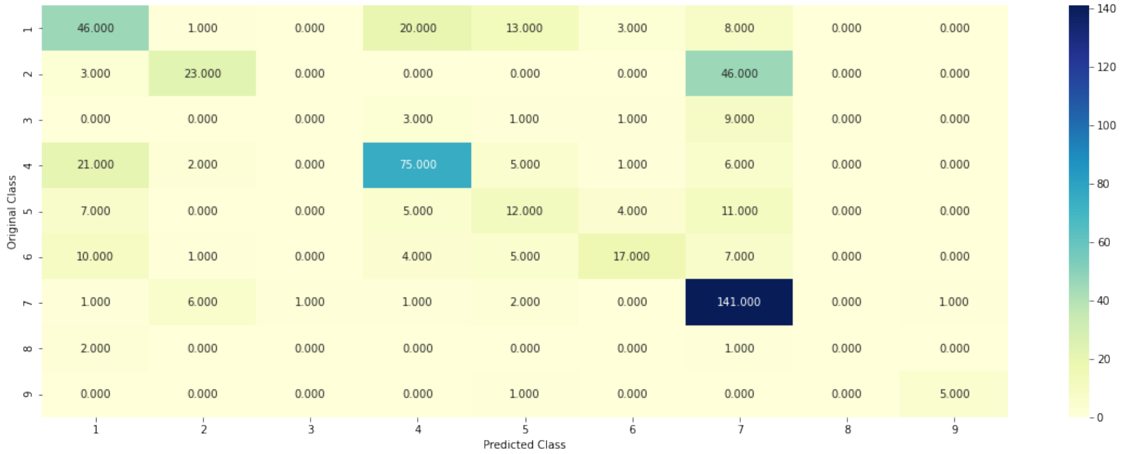


```
For values of best alpha =  1 The train log loss is: 0.891126616225253
For values of best alpha =  1 The cross validation log loss is:
1.3127574719118553
For values of best alpha =  1 The test log loss is: 1.2661135239608725
```

8.0.1 Have to test models with best hyperparameters

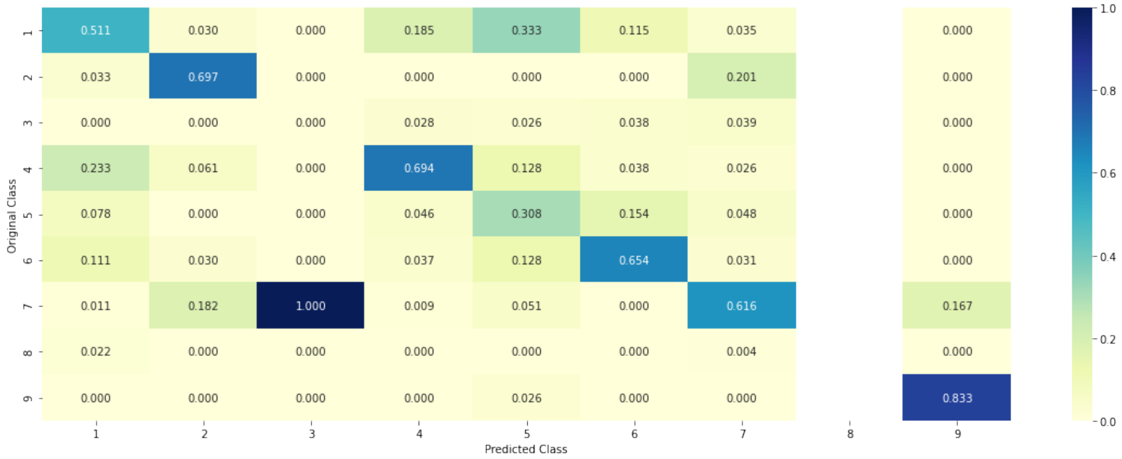
[146]:
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites 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 : 1.3127574719118553
Number of missclassified point : 0.40037593984962405
-------------------- Confusion matrix --------------------
```

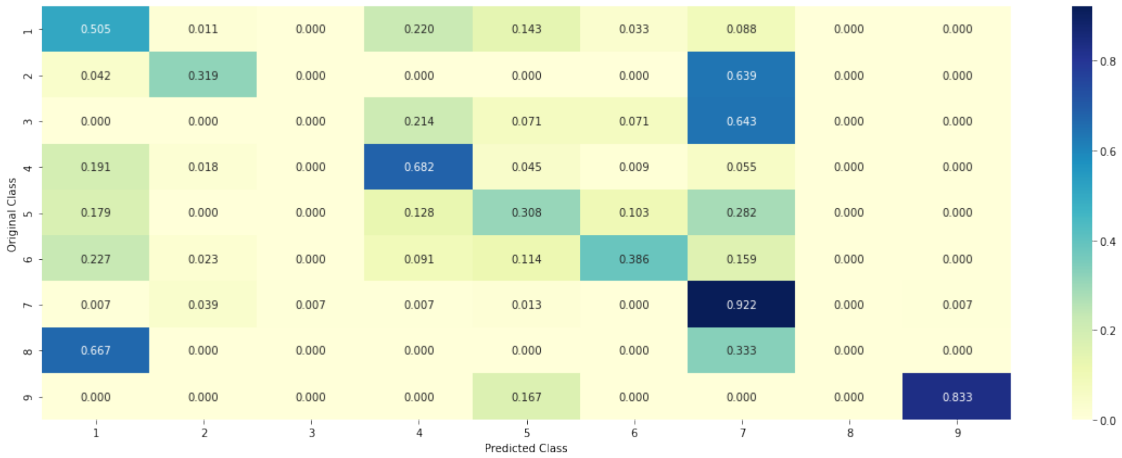


	1	2	3	4	5	6	7	8	9
1	46.000	1.000	0.000	20.000	13.000	3.000	8.000	0.000	0.000
2	3.000	23.000	0.000	0.000	0.000	0.000	46.000	0.000	0.000
3	0.000	0.000	0.000	3.000	1.000	1.000	9.000	0.000	0.000
4	21.000	2.000	0.000	75.000	5.000	1.000	6.000	0.000	0.000
5	7.000	0.000	0.000	5.000	12.000	4.000	11.000	0.000	0.000
6	10.000	1.000	0.000	4.000	5.000	17.000	7.000	0.000	0.000
7	1.000	6.000	1.000	1.000	2.000	0.000	141.000	0.000	1.000
8	2.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	5.000

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



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



feature importance no of incorrectly classified points

[149]:
```
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*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: [[0.0637 0.3303 0.0235 0.0868 0.0431 0.037
0.4048 0.0066 0.0043]]
Actual Class : 2
--------------------------------------------------
17 Text feature [presence] present in test data point [True]
18 Text feature [activating] present in test data point [True]
19 Text feature [well] present in test data point [True]
20 Text feature [recently] present in test data point [True]
21 Text feature [cell] present in test data point [True]
23 Text feature [kinase] present in test data point [True]
26 Text feature [higher] present in test data point [True]
27 Text feature [also] present in test data point [True]
29 Text feature [inhibitor] present in test data point [True]
30 Text feature [showed] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [found] present in test data point [True]
34 Text feature [however] present in test data point [True]
35 Text feature [cells] present in test data point [True]
37 Text feature [may] present in test data point [True]
38 Text feature [mutations] present in test data point [True]
43 Text feature [10] present in test data point [True]
44 Text feature [expressing] present in test data point [True]
45 Text feature [obtained] present in test data point [True]
46 Text feature [previously] present in test data point [True]
47 Text feature [treated] present in test data point [True]
49 Text feature [identified] present in test data point [True]
50 Text feature [factor] present in test data point [True]
51 Text feature [including] present in test data point [True]
52 Text feature [observed] present in test data point [True]
54 Text feature [reported] present in test data point [True]
57 Text feature [12] present in test data point [True]
58 Text feature [described] present in test data point [True]
59 Text feature [although] present in test data point [True]
65 Text feature [respectively] present in test data point [True]
66 Text feature [without] present in test data point [True]
70 Text feature [mutation] present in test data point [True]
71 Text feature [small] present in test data point [True]
72 Text feature [two] present in test data point [True]
73 Text feature [using] present in test data point [True]
75 Text feature [due] present in test data point [True]
77 Text feature [different] present in test data point [True]
78 Text feature [recent] present in test data point [True]
```

```
80 Text feature [followed] present in test data point [True]
81 Text feature [15] present in test data point [True]
85 Text feature [therapeutic] present in test data point [True]
86 Text feature [demonstrated] present in test data point [True]
87 Text feature [suggests] present in test data point [True]
88 Text feature [inhibitors] present in test data point [True]
89 Text feature [three] present in test data point [True]
94 Text feature [discussion] present in test data point [True]
95 Text feature [consistent] present in test data point [True]
Out of the top  100  features  47 are present in query point
```

[150]:
```
test_df['TEXT'].iloc[test_point_index]
```

[150]:
```
'analyze multi institutional series type c thymic carcinomas tcs including
neuroendocrine tumors focusing expression mutations c kit materials methods
immunohistochemical expression c kit cd117 p63 cd5 neuroendocrine markers well
mutational analysis c kit exons 9 11 13 14 17 direct sequencing 48 cases tcs
immunohistochemical molecular data statistically crossed clinicopathological
features results overall 29 tumors 60 expressed cd117 69 positive cd5 85 41
cases p63 neuroendocrine markers stained six atypical carcinoids five poorly
differentiated thymic squamous cell carcinomas overall six cd117 positive cases
12 5 showed c kit mutation mutation detected cd117 negative tumors carcinoids
mutations found poorly differentiated thymic squamous cell carcinomas expressing
cd117 cd5 p63 lacking neuroendocrine markers 6 12 cases features mutations
involved exon 11 four cases v559a l576p y553n w557r exon 9 e490k exon 17 d820e
conclusions tcs need immunohistochemical screening cd117 c kit mutation analysis
mandatory cd117 positive cases particularly coexpressing cd5 p63 lacking
neuroendocrine differentiation finding c kit mutation predict efficacy different
c kit inhibitors carcinoma cd117 c kit immunohistochemistry mutation thymus
topic mutation carcinoid tumor squamous cell carcinoma mutation analysis exons
mandibulofacial dysostosis neurosecretory systems proto oncogene protein c kit
neoplasms thymic carcinoma c kit mutation tp63 gene issue section translational
research introduction thymic carcinomas tcs type c rare malignant neoplasms poor
prognosis limited effective therapeutic options 1 4 half tcs express cd117
product proto oncogene c kit subset tcs harbor c kit mutations 5 14 previous
works demonstrated high frequency cd117 expression tc 80 whereas thymomas
usually express cd117 c kit mutation detected thymomas 5 6 8 9 11 12 note
different drugs acting inhibitors receptor tyrosine kinases rtks including c kit
demonstrated significant clinical benefit selected patients c kit mutated tc 7
10 12 13 prevalence role c kit mutations multi institutional series consecutive
cd117 positive cd117 negative tcs investigated based experience review pertinent
literature practical suggestions rationale use rtk inhibitors tc also provided
materials methods forty eight tcs thymic neuroendocrine tumors retrospectively
collected four different institutions 23 cases modena 17 reggio emilia 6 padova
2 cremona cases reviewed reclassified tcs 42 cases thymic neuroendocrine tumors
6 cases features atypical carcinoid according morphological criteria 2004 world
health organisation classification thymic tumors 2 material consisted 31
```

mediastinal biopsies 64 5 17 surgical resections two cases included present study previously published case reports 10 14 clinicopathological data recorded pathological reports clinical charts following variables registered present study sex age histotype immunohistochemical expression cd117 cd5 p63 chromogranin synaptophysin presence c kit mutations patients survival considered study relatively limited number cases different tumor stage variable therapeutic approaches study conducted accordance precepts helsinki declaration according national laws study require ethical committee approval since data handled anonymously immunohistochemical analysis case 4 thick sections obtained representative block sections air dried overnight 37 c deparaffinized xylene rehydrated decreasing concentration alcohol water endogenous peroxidase activity blocked immersion 10 min 3 hydrogen peroxide h2o2 methanol incubation primary antibodies accomplished modified streptavidin biotin peroxidase technique using automated immunostainer ventana benchmark tucson az 3 3diaminobenzidine used chromogene harris hematoxylin counterstain panel antibodies used study technical characteristics following cd117 clone a4502 dakopatts glostrup denmark 1 200 dilution without antigen retrieval cd5 clone sp19 ventana prediluted microwave antigen retrieval p63 clone 4a4 neomarkers fremont ca 1 400 dilution microwave antigen retrieval chromogranin clone lk2h10 ventana prediluted microwave antigen retrieval synaptophysin polyclonal ventana prediluted microwave antigen retrieval negative positive controls included batch negative controls specificity staining carried immunostaining duplicate sections nonimmune mouse igg concentration corresponding primary antibody normal tonsil tissue used positive control cd117 mast cells cd5 lymphocytes p63 squamous epithelium pulmonary typical carcinoid serve positive control chromogranin synaptophysin percentage positive cells intensity staining 0 negative 1 weak 2 moderate 3 strong recorded lesion considered positive least 10 cells reacted moderate strong intensity relevant subcellular localization nuclear p63 cytoplasmic chromogranin synaptophysin cytoplasmic membranous cd117 cd5 mutational analysis molecular analysis carried formalin fixed paraffin embedded tissues 43 cases frozen samples 5 cases dna content extracted tumor cells pcr carried 20 l reactions containing 50 200 ng dna 2 l commercial pcr buffer final concentration 1x applied biosystems foster city ca 1 0 2 0 mm mgcl2 400 dntp 40 pmol primer 3 units amplitaq gold polymerase applied biosystems pcr reaction carried uno ii thermoblock biometra gottingen germany initial denaturation 94 c 10 min followed 41 cycles final extension step 7 min 72 c cycles included denaturation 95 c 1 min annealing 53 c 66 c 1 min extension 72 c 2 min amplified dna electrophoresed 2 agarose gel 1 h 110 v amplification products purified using minelute pcr purification kit qiagen hilden germany indicated manufacturer instructions pcr products sequenced directions abi prism bigdye terminator v1 1 cycle sequencing kit applied biosystems using primers employed pcr cycle sequencing products finally purified centri sep spin columns applied biosystems subsequently runned abi prism 310 automatic sequencer applied biosystems data analyzed sequencing analysis 5 2 software applied biosystems forward reverse oligonucleotide primers used amplify c kit exons 9 11 13 14 17 listed supplemental table s1 available annals oncology online statistical analysis correlation clinicopathological variables immunohistochemical molecular results carried using contingency table methods tested significance using pearson 2 test spss version 16 0 chicago inc chicago il difference probability p values 0 05 considered significant results clinicopathological features summarized table 1 briefly case series consisted 26 males 54 22 females mean age 61 years apart 6 atypical carcinoids non neuroendocrine tcs mainly consisted squamous cell histotype 38 cases poorly differentiated morphology 24 cases overall 42 cases 87 tumors stage iii iv diagnosis overall survival available 38 patients ranging 8 89 months mean 25 months table 1 clinicopathologic characteristics thymic neoplasms characteristics cases n 48 n age mean 61 years median 63 5 years range 34 84 years sex male 26 54 female 22 46 type material biopsy 31 64 5 resection 17 33 5 histotype squamous cell 38 79 adenocarcinoma 1 2 mucoepidermoid 1 2 atypical carcinoid 6 13 lymphoepitelioma like 1 2 myoepithelial 1 2 stage ii 6 13 iii 11 23 iva 20 41 ivb 11 23 immunohistochemical molecular characteristics summarized table 2 note 29 cases 60 showed immunostaining cd117 carcinoids 23 60 5 squamous cell carcinomas 33 cases 69 positive cd5 32 84 squamous cell carcinomas lymphoepithelioma like carcinoma 41 cases 85 immunoreacted p63 squamous cell carcinomas mucoepidermoid carcinoma lymphoepithelioma like carcinoma myoepithelial carcinoma figure 1 positive staining chromogranin synaptophysin observed six atypical carcinoids one four poorly differentiated thymic squamous cell carcinomas respectively table 2 immunohistochemical molecular features 48 thymic carcinomas feature cases n 48 n cd117 positive 29 60 negative 19 40 cd5 positive 33 69 negative 15 31 p63 positive 41 85 negative 7 15 synaptophysin positive 10 21 negative 38 79 chromogranin positive 7 15 negative 41 85 c kit wild type 42 87 5 mutated 6 12 5 exon 11 v559a l576p y553n w557r 4 66 exon 17 d820e 1 17 exon 9 e490k 1 17 figure 1 example poorly differentiated thymic carcinoma haematoxylin eosin staining 200 expressing cd5 b immunohistochemistry 200 p63 c immunohistochemistry 200 cd117 immunohistochemistry 200 view largedownload slide example poorly differentiated thymic carcinoma haematoxylin eosin staining 200 expressing cd5 b immunohistochemistry 200 p63 c immunohistochemistry 200 cd117 immunohistochemistry 200 overall activating c kit mutations observed six cases 12 5 considering cd117 positive neoplasms rate c kit mutations raised 21 6 29 cases important despite positivity cd117 mutation detected atypical carcinoids excluding atypical carcinoids observed frequency c kit mutations cd117 positive tcs 26 6 23 cases gene alterations consisted missense mutations heterozygosis involving exon 11 four cases exon 9 one case exon 17 one case detail mutations exon 11 v559a l576p y553n w557r supplemental figure s1 available annals oncology online three mutations unprecedented tcs l576p previously found poorly differentiated tc 9 also type c kit mutation detected exon 9 e490k previously reported tc supplemental figure s1 available annals oncology online statistical analysis cd117 expression significantly correlated c kit mutations p 0 034 6 c kit mutated cases robustly positive score 3 50 tumor cells cd117 among wild type cases n 42 23 tumors stained cd117 19 cases completely negative finally striking relationship morphology immunoprofile c kit mutations observed c kit mutated tcs consisted poorly differentiated squamous cell carcinomas showing solid growth monomorphic cells moderate cytoplasm nuclei single prominent nucleolus dissected bands dense collagen tumor cells strongly expressed cd117 cd5 p63 stain neuroendocrine markers taking consideration tcs displaying latter characteristics 12 cases overall c kit mutations detected half cases 6 cases discussion tcs aggressive neoplasms presenting unresectable mediastinal masses majority cases multimodal chemoradiotherapy resulting often ineffective advanced stage 1 2 despite tcs may show several genetic alterations reliable molecular targets relevant targeted therapies far identified 3 15 17 among different molecular pathways consistent body evidence suggests critical role proto oncogene c kit tc 16 17 fact many tcs characterized high levels c kit protein transcripts overexpression cd117 product c kit 5 6 biomarker useful differential diagnosis thymomas mimicking neoplasms e squamous cell carcinoma lung typically negative 5 6 importantly subgroup tcs cd117 expression related constitutive somatic activating c kit mutation 7 9 10 13 14 well known among cd117 positive tumors namely gastrointestinal stromal tumors gist small cell lung cancer seminoma melanoma adenoid cystic carcinoma showing c kit mutations clinical benefit selective c kit inhibitors e imatinib sunitinib sorafenib 18 several works analyzed c kit mutations thymomas tcs results summarized table 3 supplemental table s2 available annals oncology online pan et al 5 evidenced cd117 expression 86 tcs direct sequencing c kit juxtamembrane exons 9 11 tyrosine kinase exons 13 17 domains failed evidence mutational alterations 22 tcs similarly tsuchida et al 9 demonstrated cd117 immunostaining 65 17 tcs mutations detected 13 analyzed cases petrini et al 12 recently tested c kit mutations eight tcs five thymomas one tc cell line t1889 direct sequencing analysis exon 1 exon 20 despite significant difference cd117 expression tcs 46 thymomas 4 authors find c kit mutations 12 note cd117 expression observed primary relapsed tumors significantly associated worse overall progression free survival 12 yoh et al 8 collected 24 thymomas 17 tcs detecting epidermal growth factor receptor mutations 2 thymomas c kit missense mutation exon 11 1 tc l576p table 3 summary clinicopathological features published thymic carcinomas c kit mutations treated c kit inhibitors reference age sex histologic type c kit mutation stage therapy drug clinical response strobel et al 7 54 tc squamous cell g3 v560del ex11 metastatic none imatinib sd 6 months bisagni et al 10 46 tc squamous cell g3 d820e ex17 pt3 n2 m1 ct rt sorafenib pr 15 months disel et al 13 47 f tc squamous cell g3 del577 578 579 ex11 iva ct rt sorafenib sd buti et al 14 48 tc squamous cell g3 y553n iv ct imatinib pr 8 months li et al 19 46 tc squamous cell g3 nd iv ct sorafenib sd 9 months chuah et al 20 na type b2 nd imatinib ct dasatinib lr hamada et al 21 case 1 62 atypical carcinoid none invasive ct imatinib good clinical response case 2 58 atypical carcinoid nd invasive rt nessuno recurrence metastasis giaccone et al 22 case 1 36 tc nd ivb ct imatinib pd case 2 67 type b3 none iva rt ct imatinib sd case 3 47 type b2 3 nd iva ct imatinib sd case 4 76 tc nd ivb none imatinib pd case 5 36 tc nd ivb ct imatinib pd case 6 71 tc none ivb none imatinib pd case 7 69 f tc squamous cell type none ivb none imatinib pd strobel et al 23 case 1 35 tc squamous cell type none ivb ct imatinib sunitinib pr case 2 69 tc squamous cell type none iva rt ct sunitinib pr case 3 77 tc squamous cell type none ii sunitinib pr case 4 28 f tc undifferentiated none ivb ct rt sunitinib pr 2 months palmieri et al 24 15 cases 4 type b2 none na na imatinib pd 2 type b2 b3 none na na imatinib pd 6 type b3 none na na imatinib 1 sd 3 tc none na na imatinib pd ct chemotherapy f female g3 grade 3 poorly differentiated male na available nd done pd progression disease pr partial response rt radiotherapy surgery sd stable disease tc thymic carcinoma finally girard et al 11 evidenced 2 c kit mutated 7 tcs 1 exon 14 h697y 1 exon 11 v560del particular novel exon 14 missense mutation h697y highly sensitive sunitinib rather imatinib noteworthy c kit mutated tcs consisted squamous cell carcinoma histotype poorly differentiated morphology grade 3 without keratinization although prospective trials using imatinib different thymic malignancies failed demonstrate clinical responses 17 22 24 handful case reports yielded promising results selected patients 7 10 13 14 21 2004 strobel et al 7 first reported case chemoresistant undifferentiated tc exon 11 v560del c kit mutation experiencing stable disease 6 months using imatinib mesylate subsequently bisagni et al 10 reported long lasting partial response tc showing missense mutation exon 17 d820e based previous clinical experience dealing kind mutation gist authors decided treat patient sorafenib small molecule inhibiting several targets c kit pdgfrs vascular endothelial growth factor receptors vegfrs flt 3 c raf b raf recently disel et al 13 reported deletion mutation exon 11 577 579del advanced tc squamous features patient stable disease using sorafenib note experiencing consistent partial response 8 months patient poorly differentiated chemoresistant tc novel missense mutation exon 11 y553n detected 14 patient treated imatinib similarly happens gist harboring mutation c kit mutations reported tcs involving exon 11 l576p exon 14 h697y supporting critical role c kit mutations tcs clinical responses registered wild type tcs treated imatinib recent trials 17 22 24 although hamada et al 21 described clinical benefit wild type thymic atypical carcinoid treated imatinib case reports highlighted effectiveness sorafenib 19 dasatinib 20 somatostatin receptor 2 25 tc metastatic thymoma strobel et al 23 recently reported partial response four patients wild type tcs three tcs squamous cell differentiation one undifferentiated tc treated sunitinib however differently imatinib sunitinib multi targeted rtk inhibitor interfering several targets c kit pdgfrs vegfrs flt 3 also antiangiogenetic role advanced gist sunitinib adopted second line therapy tumor develops resistance imatinib drug particularly effective gists harboring c kit mutations exon 13 17 pdgfr alpha mutations 26 far standard care light data previous experiences imatinib sorafenib sunitinib tc 7 9 10 13 14 practical therapeutic algorithm based c kit mutation type illustrated figure 2 briefly seems effective use imatinib mesylate tcs significantly depends presence type c kit mutation detected tumor cells pharmacological agent selectively inhibiting type iii rtks hand sorafenib sunitinib less selective imatinib effectively adopted tcs harboring imatinib resistant c kit mutations e involving exons 13 14 17 wild type tcs due antiangiogenetic role suggested recent preliminary experience strobel et al 23 figure 2 helpful therapeutic decision tree using targeted therapies thymic carcinoma based expression mutations c kit view largedownload slide helpful therapeutic decision tree using targeted therapies thymic carcinoma based expression mutations c kit conclusion opinion immunohistochemical screening including small panel antibodies cd117 cd5 p63 neuroendocrine markers mandatory cases tc cd117 positive tcs tested c kit mutations expanding molecular test exons 9 11 13 14 17 probability find mutations higher cd117 positive thymic squamous cell carcinoma poorly

```
differentiated morphology hematoxylin eosin coexpression cd5 p63 absence
neuroendocrine markers mutations involving c kit tcs seem possess biological
significance observed gist based presence absence type c kit mutation tcs may
benefit targeted therapy different rtk inhibitors '
```

```
[151]: no_feature
```

```
[151]: 100
```

```
[152]: test_df['Gene'].iloc[test_point_index]
```

```
[152]: 'KIT'
```

```
[153]: test_df['Variation'].iloc[test_point_index]
```

```
[153]: 'P577_D579del'
```

```
[154]: clf.coef_.shape
```

```
[154]: (9, 56217)
```

```
[155]: indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
        indices[0]
```

```
[155]: array([28108, 14369, 14371, 39926, 14374, 14378, 39919, 39917, 39916,
               39915, 39907, 39906, 39894, 39890, 14397, 39886, 39885, 14403,
               39855, 39858, 39859, 39864, 39867, 39868, 14368, 39872, 14411,
               14410, 39876, 39879, 39882, 39884, 39874, 39854, 39928, 14365,
               39998, 14303, 14304, 39997, 39996, 14315, 39990, 39988, 39981,
               39974, 39973, 39971, 14329, 14330, 39969, 14339, 39965, 14364,
               14363, 14362, 14361, 14360, 14359, 39934, 14358, 39936, 39941,
               39960, 14347, 39962, 14343, 14357, 14301, 39853, 39846, 14533,
               14534, 14535, 39747, 39746, 39745, 14545, 39738, 39737, 39732,
               14555, 39717, 14559, 39715, 14561, 39714, 39713, 14590, 39681,
               14588, 14587, 14586, 39683, 14526, 39684, 39692, 39695, 39696,
               39698], dtype=int64)
```

```
[156]: # 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 = CountVectorizer()
            var_count_vec = CountVectorizer()
            text_count_vec = CountVectorizer(min_df=3)

            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")
```

[167]:

```
test_point_index = 0
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))

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 : 4
Predicted Class Probabilities: [[0.0595 0.0817 0.0218 0.6849 0.0405 0.0342
0.0673 0.0061 0.0039]]
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

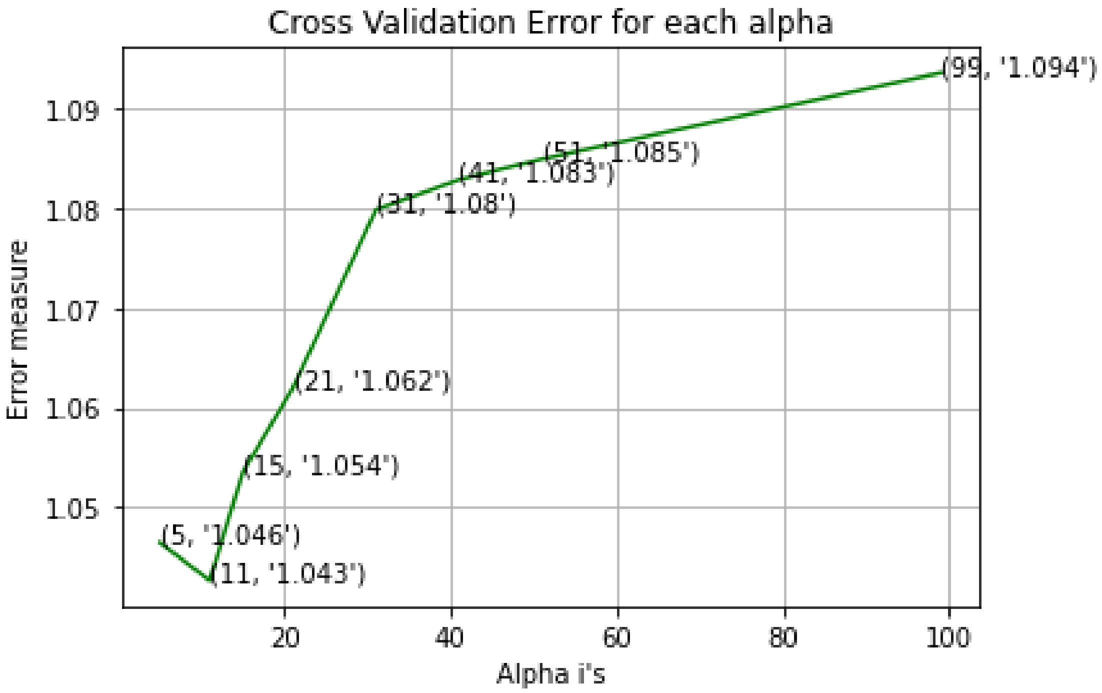
9 k nearest neighbours classification

[168]:

```
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))
```

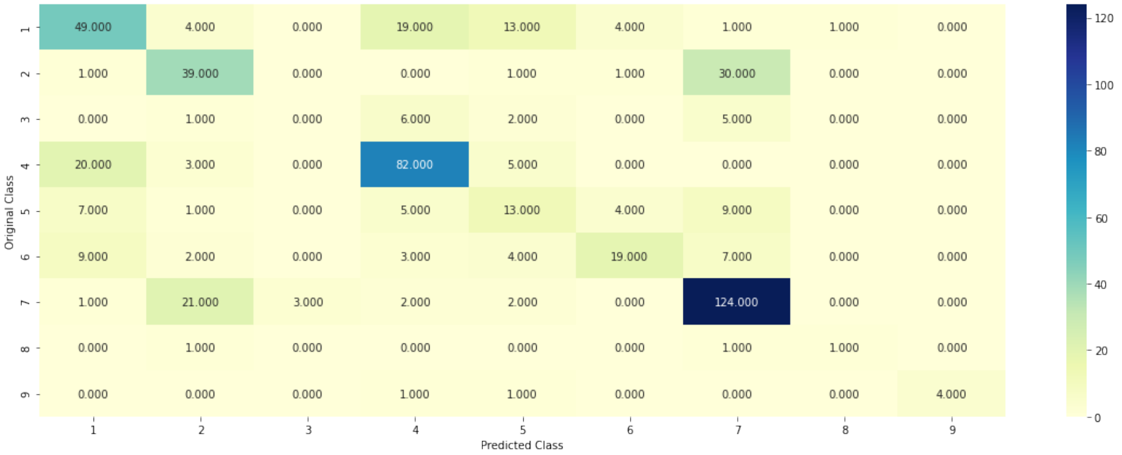
```
for alpha = 5
Log Loss : 1.0464399394561428
for alpha = 11
Log Loss : 1.042623410526951
for alpha = 15
Log Loss : 1.0535135471553312
for alpha = 21
Log Loss : 1.06192807442335
for alpha = 31
Log Loss : 1.0798344332160619
for alpha = 41
Log Loss : 1.0828543297413094
for alpha = 51
Log Loss : 1.0849359204083553
for alpha = 99
Log Loss : 1.093580219828438
```



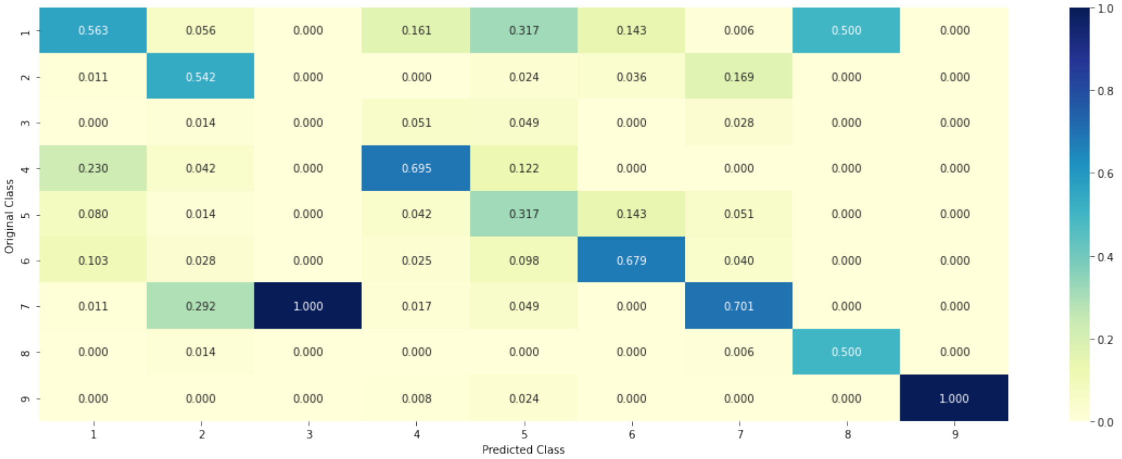
```
For values of best alpha =  11 The train log loss is: 0.6091524695560726
For values of best alpha =  11 The cross validation log loss is:
1.042623410526951
For values of best alpha =  11 The test log loss is: 1.0224742181090396
```

```
[169]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
       predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,␣
        ↪cv_x_responseCoding, cv_y, clf)
```

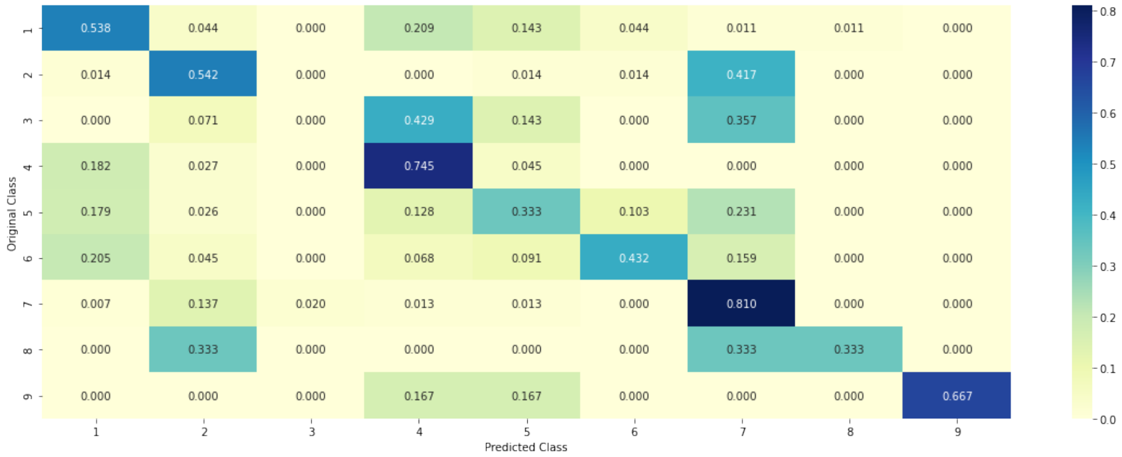
```
Log loss : 1.042623410526951
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```



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



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



[170]:

```
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 : 2
The  11  nearest neighbours of the test points belongs to classes [2 2 7 2 2 7 2
7 7 7 7]
Fequency of nearest points : Counter({7: 6, 2: 5})
```

[171]:

```
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 : 6
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belongs
to classes [2 7 7 7 2 6 6 6 6 6 6]
Fequency of nearest points : Counter({6: 6, 7: 3, 2: 2})
```

[172]:

```
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 = 13

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 : 1
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test points belongs
to classes [4 4 1 4 1 1 1 1 1 4 4]
Fequency of nearest points : Counter({1: 6, 4: 5})
```

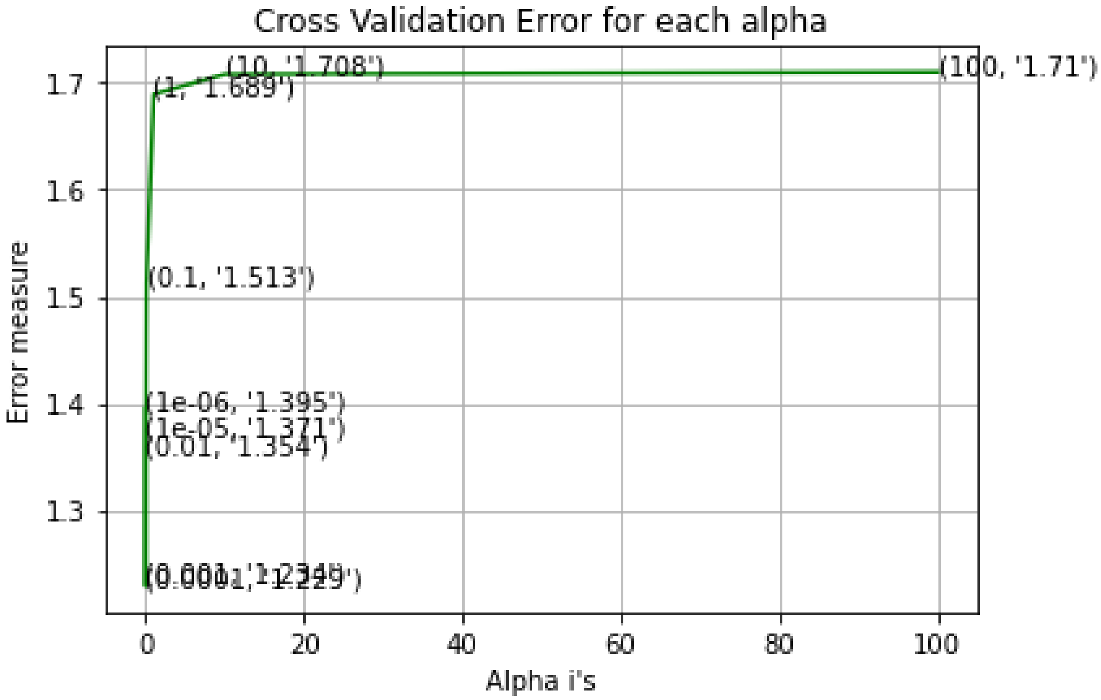
10 Logistic regression with class balancing

[173]:

```
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 : 1.3946693762778002
for alpha = 1e-05
Log Loss : 1.3709316106509564
for alpha = 0.0001
Log Loss : 1.2287379542037373
for alpha = 0.001
Log Loss : 1.2344241579421753
for alpha = 0.01
Log Loss : 1.3543126414550897
for alpha = 0.1
Log Loss : 1.512895778914386
```

```
for alpha = 1
Log Loss : 1.689266027560365
for alpha = 10
Log Loss : 1.7081954936919648
for alpha = 100
Log Loss : 1.7102047599930676
```

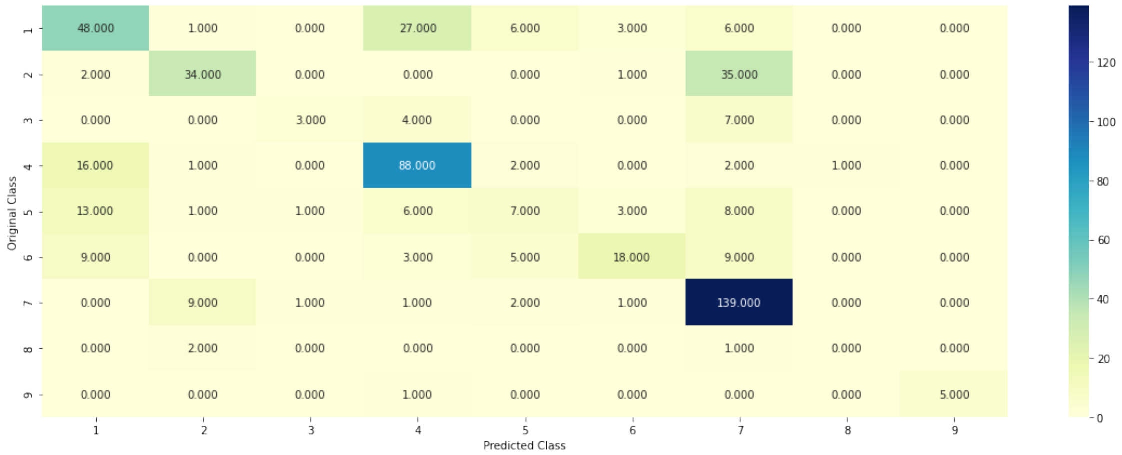


```
For values of best alpha =  0.0001 The train log loss is: 0.4919373812950623
For values of best alpha =  0.0001 The cross validation log loss is:
1.2287379542037373
For values of best alpha =  0.0001 The test log loss is: 1.131338170986897
```

[174]:

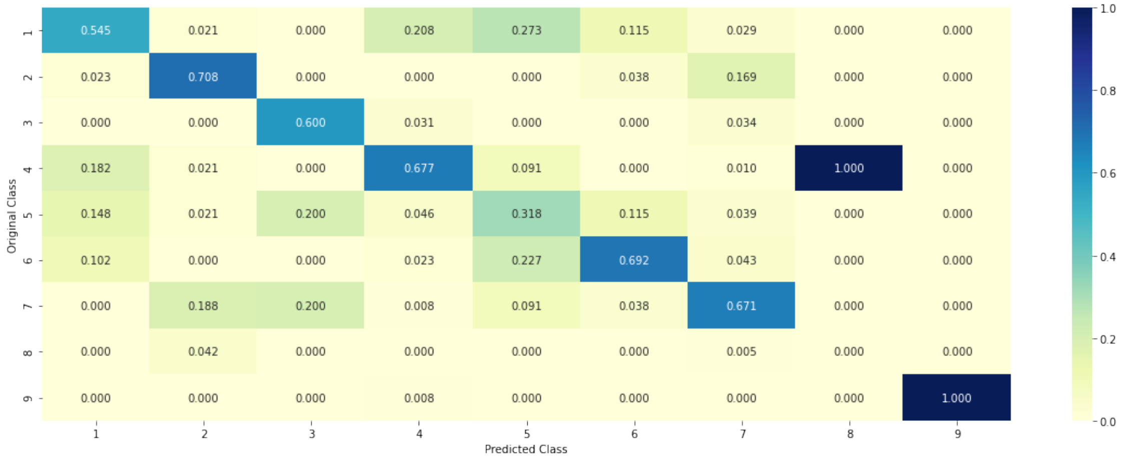
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,␣
 ↪cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.2287379542037373
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```



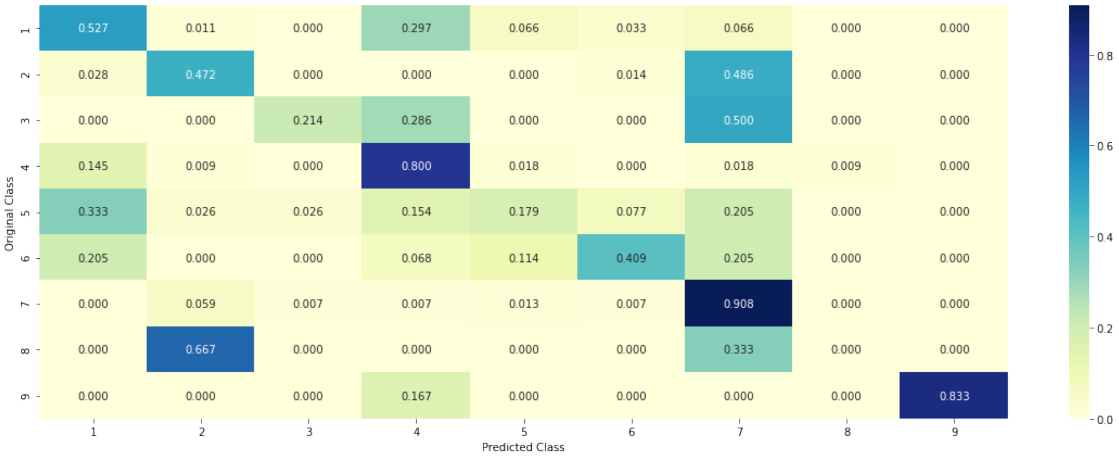
	1	2	3	4	5	6	7	8	9
1	48.000	1.000	0.000	27.000	6.000	3.000	6.000	0.000	0.000
2	2.000	34.000	0.000	0.000	0.000	1.000	35.000	0.000	0.000
3	0.000	0.000	3.000	4.000	0.000	0.000	7.000	0.000	0.000
4	16.000	1.000	0.000	88.000	2.000	0.000	2.000	1.000	0.000
5	13.000	1.000	1.000	6.000	7.000	3.000	8.000	0.000	0.000
6	9.000	0.000	0.000	3.000	5.000	18.000	9.000	0.000	0.000
7	0.000	9.000	1.000	1.000	2.000	1.000	139.000	0.000	0.000
8	0.000	2.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.545	0.021	0.000	0.208	0.273	0.115	0.029	0.000	0.000
2	0.023	0.708	0.000	0.000	0.000	0.038	0.169	0.000	0.000
3	0.000	0.000	0.600	0.031	0.000	0.000	0.034	0.000	0.000
4	0.182	0.021	0.000	0.677	0.091	0.000	0.010	1.000	0.000
5	0.148	0.021	0.200	0.046	0.318	0.115	0.039	0.000	0.000
6	0.102	0.000	0.000	0.023	0.227	0.692	0.043	0.000	0.000
7	0.000	0.188	0.200	0.008	0.091	0.038	0.671	0.000	0.000
8	0.000	0.042	0.000	0.000	0.000	0.000	0.005	0.000	0.000
9	0.000	0.000	0.000	0.008	0.000	0.000	0.000	0.000	1.000

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



[175]:

```
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']))
```

[176]:

```
# 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 = 1
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 : 2
Predicted Class Probabilities: [[0.0457 0.5251 0.0201 0.0363 0.03   0.0243
0.3059 0.0053 0.0072]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

[177]:
```
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)
```

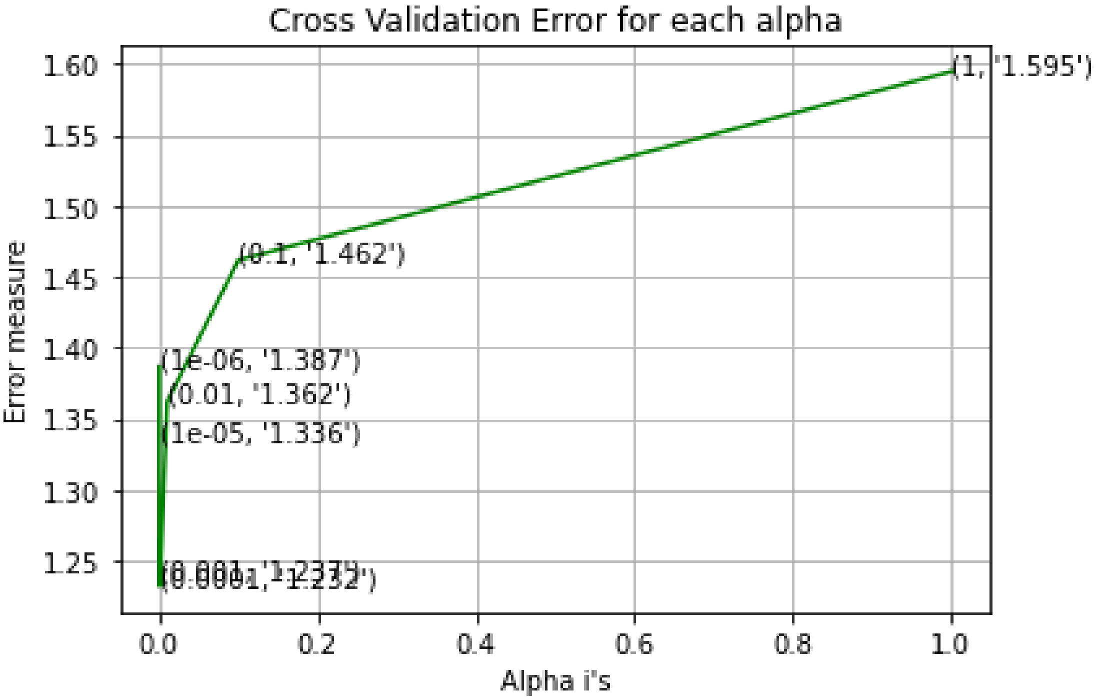
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0736 0.1421 0.0184 0.116  0.0266 0.0305 0.576
0.0078 0.009 ]]
Actual Class : 7
--------------------------------------------------
287 Text feature [jak] present in test data point [True]
305 Text feature [stat] present in test data point [True]
415 Text feature [tyr694] present in test data point [True]
489 Text feature [constitutively] present in test data point [True]
494 Text feature [constitutive] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

11 Logistic regression without class balancing

[178]:
```
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 : 1.3868235835459999
for alpha = 1e-05
Log Loss : 1.3355047214732567
```

```
for alpha = 0.0001
Log Loss : 1.232166627981432
for alpha = 0.001
Log Loss : 1.2374074620880506
for alpha = 0.01
Log Loss : 1.362142650044525
for alpha = 0.1
Log Loss : 1.4618323609175399
for alpha = 1
Log Loss : 1.594577009762644
```

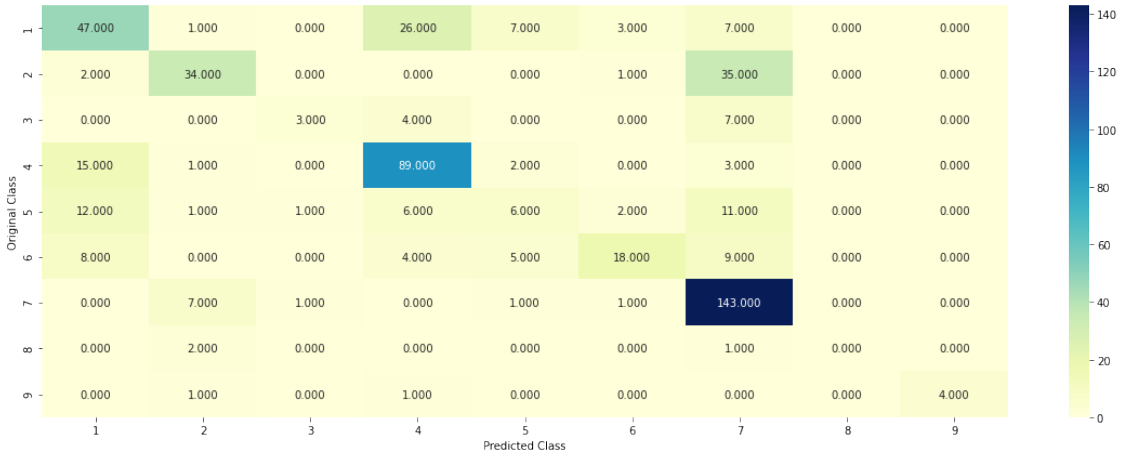


```
For values of best alpha =  0.0001 The train log loss is: 0.47622756268301025
For values of best alpha =  0.0001 The cross validation log loss is:
1.232166627981432
For values of best alpha =  0.0001 The test log loss is: 1.1413373533671145
```

[179]:

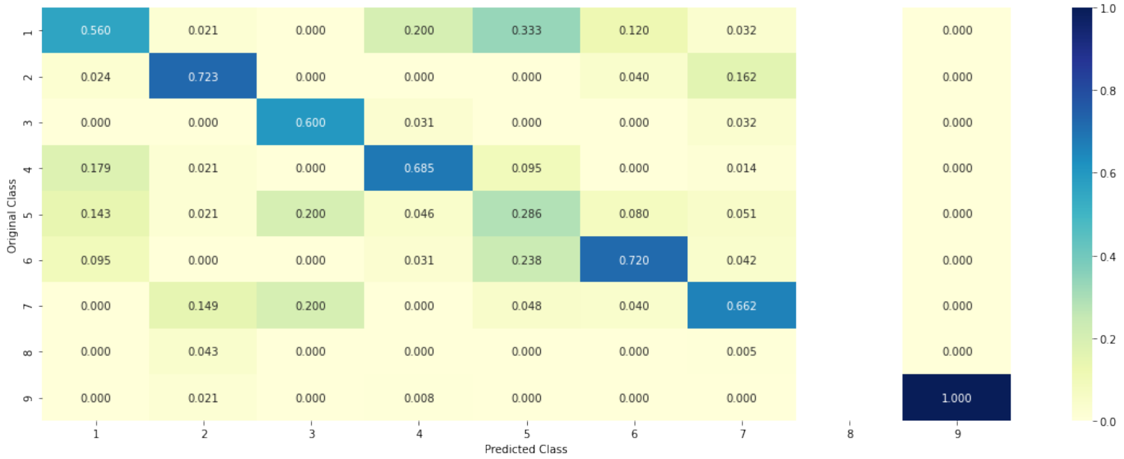
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.232166627981432
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



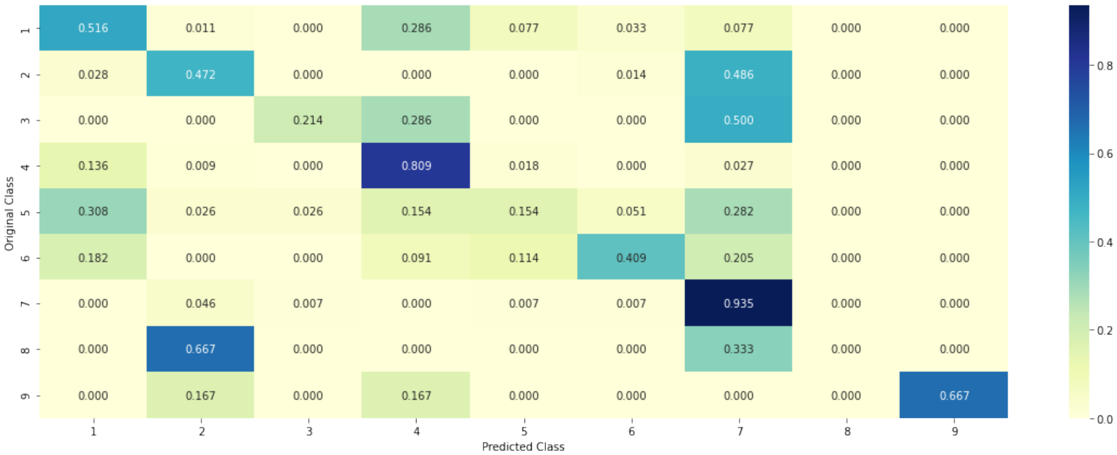
	1	2	3	4	5	6	7	8	9
1	47.000	1.000	0.000	26.000	7.000	3.000	7.000	0.000	0.000
2	2.000	34.000	0.000	0.000	0.000	1.000	35.000	0.000	0.000
3	0.000	0.000	3.000	4.000	0.000	0.000	7.000	0.000	0.000
4	15.000	1.000	0.000	89.000	2.000	0.000	3.000	0.000	0.000
5	12.000	1.000	1.000	6.000	6.000	2.000	11.000	0.000	0.000
6	8.000	0.000	0.000	4.000	5.000	18.000	9.000	0.000	0.000
7	0.000	7.000	1.000	0.000	1.000	1.000	143.000	0.000	0.000
8	0.000	2.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	1.000	0.000	1.000	0.000	0.000	0.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.560	0.021	0.000	0.200	0.333	0.120	0.032		0.000
2	0.024	0.723	0.000	0.000	0.000	0.040	0.162		0.000
3	0.000	0.000	0.600	0.031	0.000	0.000	0.032		0.000
4	0.179	0.021	0.000	0.685	0.095	0.000	0.014		0.000
5	0.143	0.021	0.200	0.046	0.286	0.080	0.051		0.000
6	0.095	0.000	0.000	0.031	0.238	0.720	0.042		0.000
7	0.000	0.149	0.200	0.000	0.048	0.040	0.662		0.000
8	0.000	0.043	0.000	0.000	0.000	0.000	0.005		0.000
9	0.000	0.021	0.000	0.008	0.000	0.000	0.000		1.000

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



[180]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
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 : 2
Predicted Class Probabilities: [[0.0489 0.5163 0.0136 0.034  0.0275 0.0242
0.3221 0.0065 0.007 ]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

[181]:
```
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: [[0.0671 0.1419 0.0146 0.1171 0.0257 0.0357
0.5814 0.0082 0.0084]]
Actual Class : 7
--------------------------------------------------
341 Text feature [jak] present in test data point [True]
353 Text feature [stat] present in test data point [True]
466 Text feature [tyr694] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

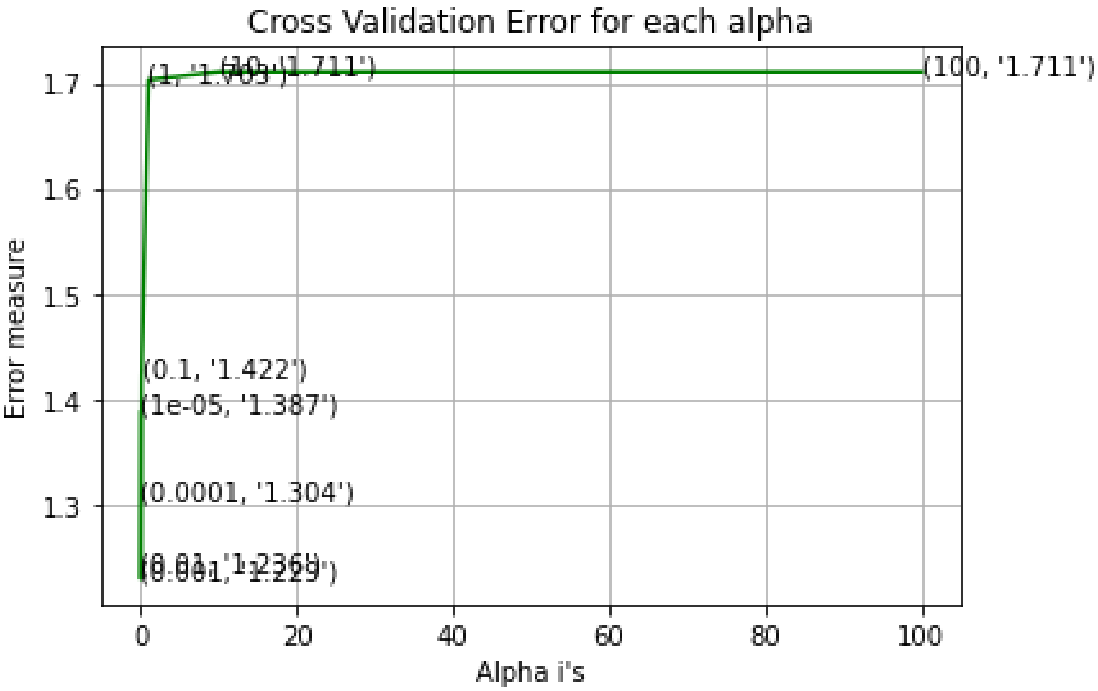
12 Linear SVMs

[182]:

```
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))
```

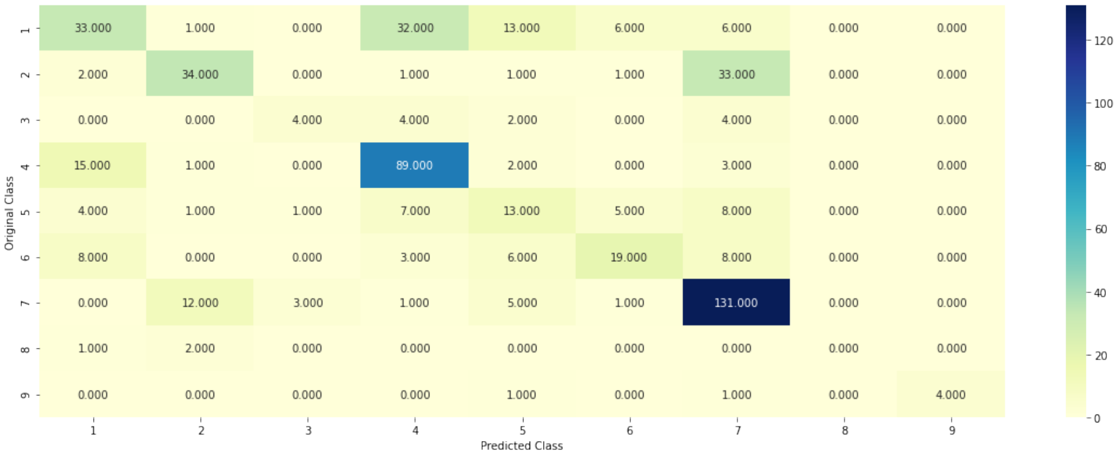
```
for C = 1e-05
Log Loss : 1.3874472394793447
for C = 0.0001
Log Loss : 1.304193084630548
for C = 0.001
Log Loss : 1.22903613149766
for C = 0.01
Log Loss : 1.2363014824047556
for C = 0.1
Log Loss : 1.422434533280168
for C = 1
Log Loss : 1.7031690409396851
for C = 10
Log Loss : 1.7105606563822395
for C = 100
Log Loss : 1.7105715142278224
```



```
For values of best alpha =  0.001 The train log loss is: 0.5339800158256987
For values of best alpha =  0.001 The cross validation log loss is:
1.22903613149766
For values of best alpha =  0.001 The test log loss is: 1.146937237837979
```

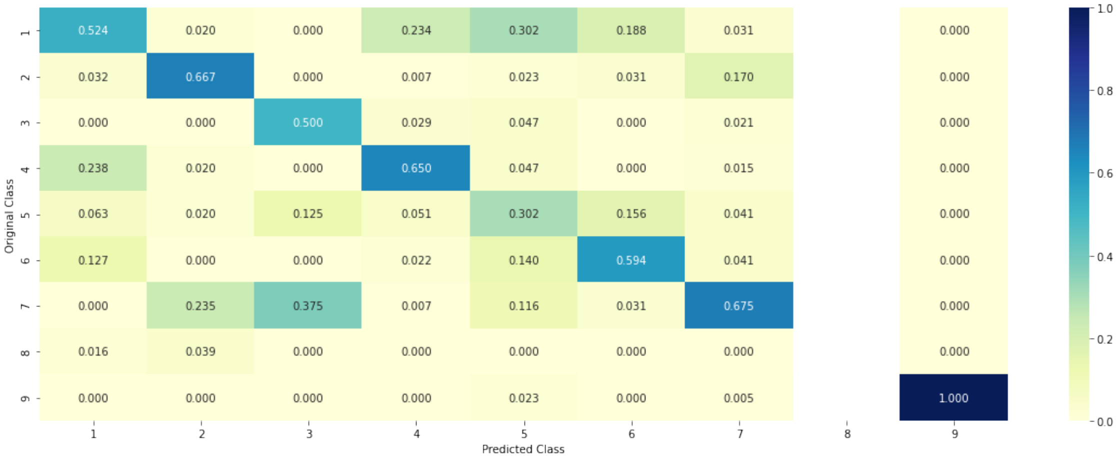
[183]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',␣
 ↪random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding,␣
 ↪train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.22903613149766
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```



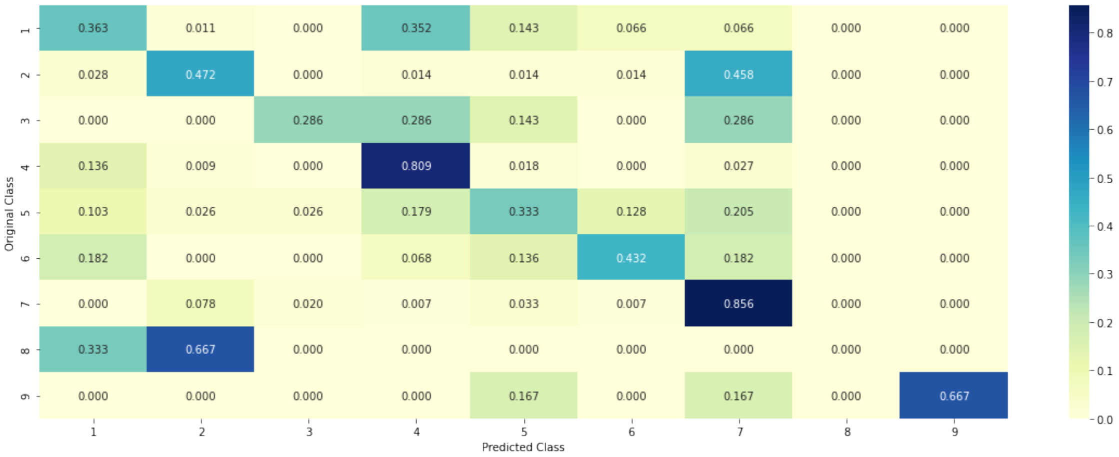
	1	2	3	4	5	6	7	8	9
1	33.000	1.000	0.000	32.000	13.000	6.000	6.000	0.000	0.000
2	2.000	34.000	0.000	1.000	1.000	1.000	33.000	0.000	0.000
3	0.000	0.000	4.000	4.000	2.000	0.000	4.000	0.000	0.000
4	15.000	1.000	0.000	89.000	2.000	0.000	3.000	0.000	0.000
5	4.000	1.000	1.000	7.000	13.000	5.000	8.000	0.000	0.000
6	8.000	0.000	0.000	3.000	6.000	19.000	8.000	0.000	0.000
7	0.000	12.000	3.000	1.000	5.000	1.000	131.000	0.000	0.000
8	1.000	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.524	0.020	0.000	0.234	0.302	0.188	0.031		0.000
2	0.032	0.667	0.000	0.007	0.023	0.031	0.170		0.000
3	0.000	0.000	0.500	0.029	0.047	0.000	0.021		0.000
4	0.238	0.020	0.000	0.650	0.047	0.000	0.015		0.000
5	0.063	0.020	0.125	0.051	0.302	0.156	0.041		0.000
6	0.127	0.000	0.000	0.022	0.140	0.594	0.041		0.000
7	0.000	0.235	0.375	0.007	0.116	0.031	0.675		0.000
8	0.016	0.039	0.000	0.000	0.000	0.000	0.000		0.000
9	0.000	0.000	0.000	0.000	0.023	0.000	0.005		1.000

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



[184]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# 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 : 2
Predicted Class Probabilities: [[0.0738 0.4863 0.0169 0.0648 0.0371 0.0364
0.2732 0.005  0.0066]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

[185]:

```
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: [[0.1112 0.1762 0.0163 0.0931 0.0361 0.0631
0.4884 0.0076 0.0081]]
Actual Class : 7
--------------------------------------------------
329 Text feature [jak] present in test data point [True]
425 Text feature [stat] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

13 Random Forest Classifier

[186]:

```
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()
'''
```

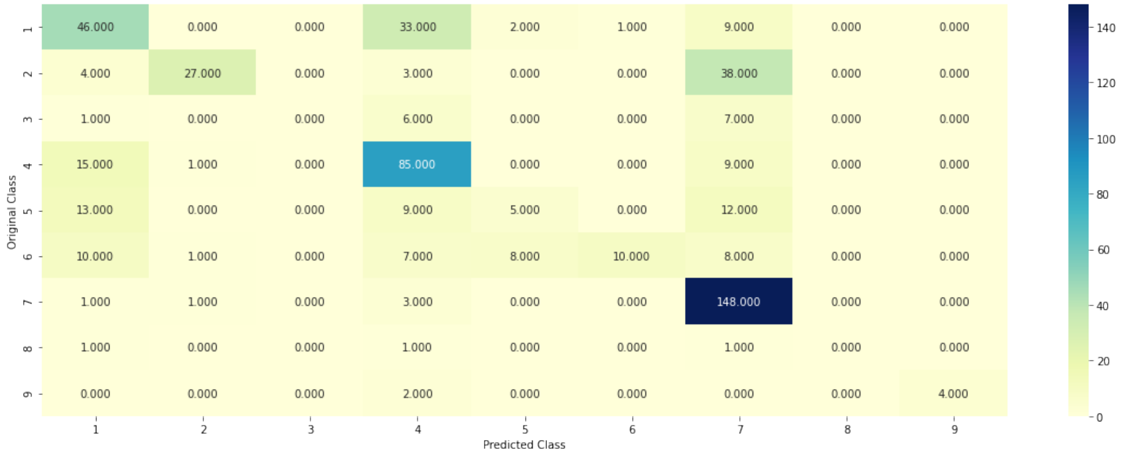
```
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
 ↪n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
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 : 1.2332222837512028
for n_estimators = 100 and max depth =  10
Log Loss : 1.1873387055544222
for n_estimators = 200 and max depth =  5
Log Loss : 1.2212569033115201
for n_estimators = 200 and max depth =  10
Log Loss : 1.1807178319552964
for n_estimators = 500 and max depth =  5
Log Loss : 1.216039029182352
for n_estimators = 500 and max depth =  10
Log Loss : 1.1768639334505706
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2199220175479633
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1758784071529003
for n_estimators = 2000 and max depth =  5
Log Loss : 1.218676958059698
for n_estimators = 2000 and max depth =  10
Log Loss : 1.175019749258833
For values of best estimator =  2000 The train log loss is: 0.6779415969745608
For values of best estimator =  2000 The cross validation log loss is:
1.175019749258833
For values of best estimator =  2000 The test log loss is: 1.1421485226394097
```

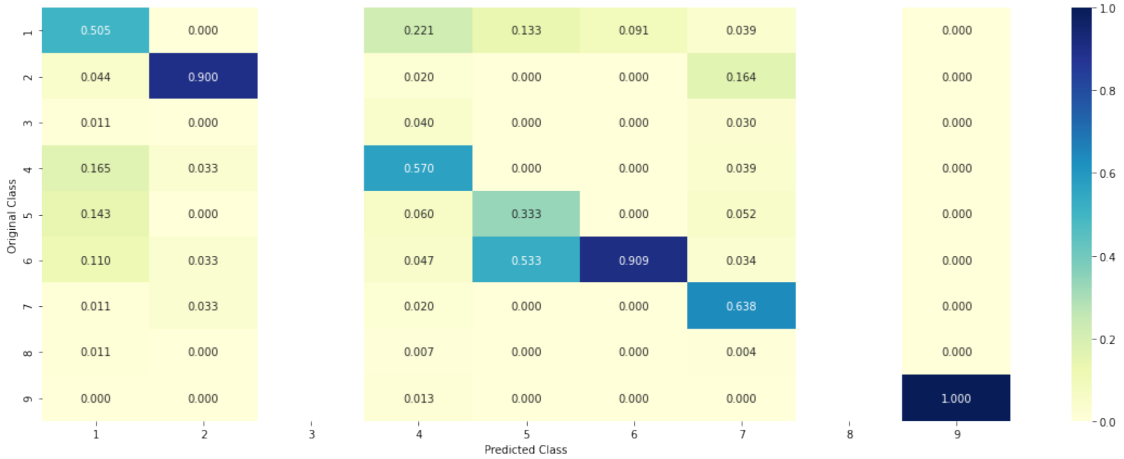
[187]:

```
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)
```

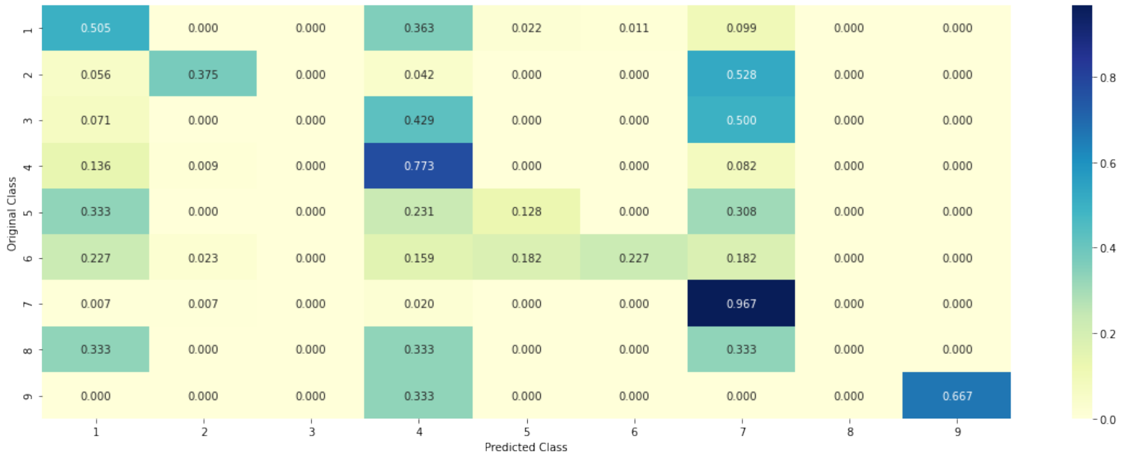
```
Log loss : 1.175019749258833
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```



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



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



[188]:

```
# test_point_index = 10
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)

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(-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 : 2
Predicted Class Probabilities: [[0.0595 0.5671 0.0165 0.058  0.0382 0.033
0.2165 0.0056 0.0057]]
Actual Class : 2
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
```

```
8 Text feature [constitutive] present in test data point [True]
9 Text feature [missense] present in test data point [True]
18 Text feature [inhibitor] present in test data point [True]
19 Text feature [therapy] present in test data point [True]
22 Text feature [therapeutic] present in test data point [True]
23 Text feature [receptor] present in test data point [True]
25 Text feature [months] present in test data point [True]
27 Text feature [patients] present in test data point [True]
30 Text feature [trials] present in test data point [True]
31 Text feature [drug] present in test data point [True]
34 Text feature [cells] present in test data point [True]
37 Text feature [clinical] present in test data point [True]
38 Text feature [resistance] present in test data point [True]
39 Text feature [growth] present in test data point [True]
40 Text feature [cell] present in test data point [True]
45 Text feature [protein] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
50 Text feature [efficacy] present in test data point [True]
55 Text feature [treated] present in test data point [True]
62 Text feature [imatinib] present in test data point [True]
64 Text feature [advanced] present in test data point [True]
69 Text feature [survival] present in test data point [True]
82 Text feature [oncogene] present in test data point [True]
88 Text feature [type] present in test data point [True]
89 Text feature [kinases] present in test data point [True]
90 Text feature [response] present in test data point [True]
92 Text feature [nuclear] present in test data point [True]
95 Text feature [lung] present in test data point [True]
98 Text feature [kit] present in test data point [True]
Out of the top  100  features  33 are present in query point
```

[189]:
```
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("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: [[0.1175 0.1157 0.0259 0.1441 0.0571 0.0538
0.4692 0.0088 0.0079]]
```

```
Actuall Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
7 Text feature [activation] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
10 Text feature [stability] present in test data point [True]
11 Text feature [erk] present in test data point [True]
15 Text feature [function] present in test data point [True]
17 Text feature [ba] present in test data point [True]
18 Text feature [inhibitor] present in test data point [True]
21 Text feature [constitutively] present in test data point [True]
23 Text feature [receptor] present in test data point [True]
27 Text feature [patients] present in test data point [True]
29 Text feature [f3] present in test data point [True]
30 Text feature [trials] present in test data point [True]
34 Text feature [cells] present in test data point [True]
35 Text feature [signaling] present in test data point [True]
37 Text feature [clinical] present in test data point [True]
39 Text feature [growth] present in test data point [True]
40 Text feature [cell] present in test data point [True]
43 Text feature [downstream] present in test data point [True]
45 Text feature [protein] present in test data point [True]
46 Text feature [lines] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
52 Text feature [inhibition] present in test data point [True]
58 Text feature [activate] present in test data point [True]
59 Text feature [erk1] present in test data point [True]
68 Text feature [ligand] present in test data point [True]
72 Text feature [proliferation] present in test data point [True]
86 Text feature [assays] present in test data point [True]
88 Text feature [type] present in test data point [True]
89 Text feature [kinases] present in test data point [True]
90 Text feature [response] present in test data point [True]
97 Text feature [core] present in test data point [True]
Out of the top  100  features  36 are present in query point
```

14 With response coding

[190]:

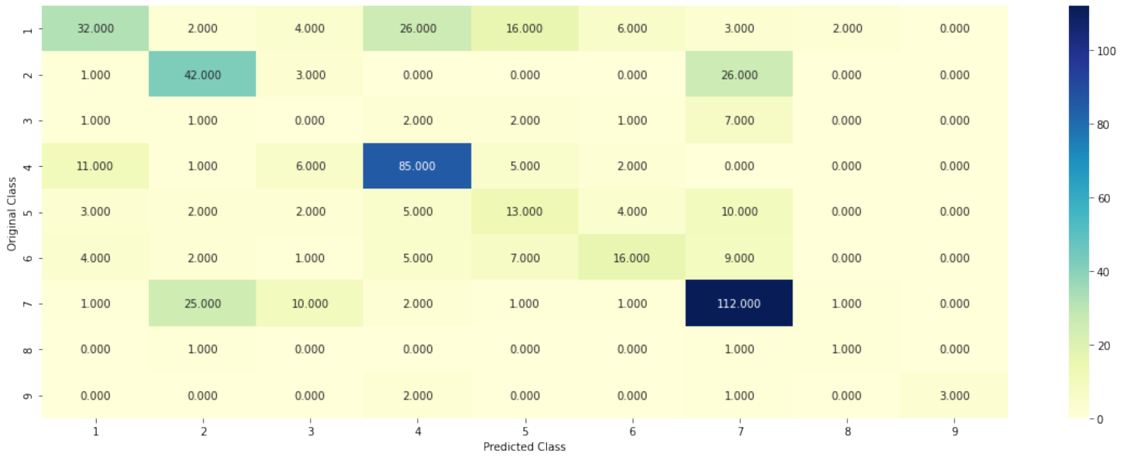
```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',␣
 ↪max_depth=j, random_state=42, n_jobs=-1)
        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 : 2.0851917489490788
for n_estimators = 10 and max depth =  3
Log Loss : 1.6184705970031164
```

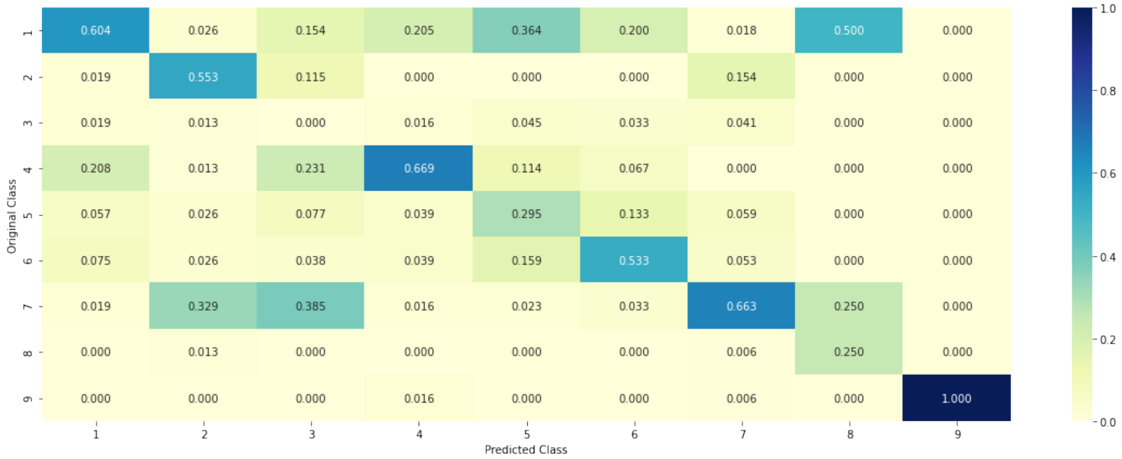
```
for n_estimators = 10 and max depth =  5
Log Loss : 1.3622452776726082
for n_estimators = 10 and max depth =  10
Log Loss : 1.8391340075807834
for n_estimators = 50 and max depth =  2
Log Loss : 1.6362165142386254
for n_estimators = 50 and max depth =  3
Log Loss : 1.4115655769719389
for n_estimators = 50 and max depth =  5
Log Loss : 1.4242095837906819
for n_estimators = 50 and max depth =  10
Log Loss : 1.795618821102472
for n_estimators = 100 and max depth =  2
Log Loss : 1.4934411618418084
for n_estimators = 100 and max depth =  3
Log Loss : 1.4457159209945227
for n_estimators = 100 and max depth =  5
Log Loss : 1.382365301234866
for n_estimators = 100 and max depth =  10
Log Loss : 1.791321675696037
for n_estimators = 200 and max depth =  2
Log Loss : 1.5731829647630942
for n_estimators = 200 and max depth =  3
Log Loss : 1.490905384603315
for n_estimators = 200 and max depth =  5
Log Loss : 1.4050812441304936
for n_estimators = 200 and max depth =  10
Log Loss : 1.7652956492223777
for n_estimators = 500 and max depth =  2
Log Loss : 1.6205515992589392
for n_estimators = 500 and max depth =  3
Log Loss : 1.5410549132312443
for n_estimators = 500 and max depth =  5
Log Loss : 1.405078165923461
for n_estimators = 500 and max depth =  10
Log Loss : 1.8110815195216445
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5754078870711825
for n_estimators = 1000 and max depth =  3
Log Loss : 1.533991898792025
for n_estimators = 1000 and max depth =  5
Log Loss : 1.392090929927875
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7930396112748088
For values of best alpha =  10 The train log loss is: 0.09487208699290473
For values of best alpha =  10 The cross validation log loss is:
1.3622452776726082
For values of best alpha =  10 The test log loss is: 1.2453872281244254
```

```
[191]: 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)
```

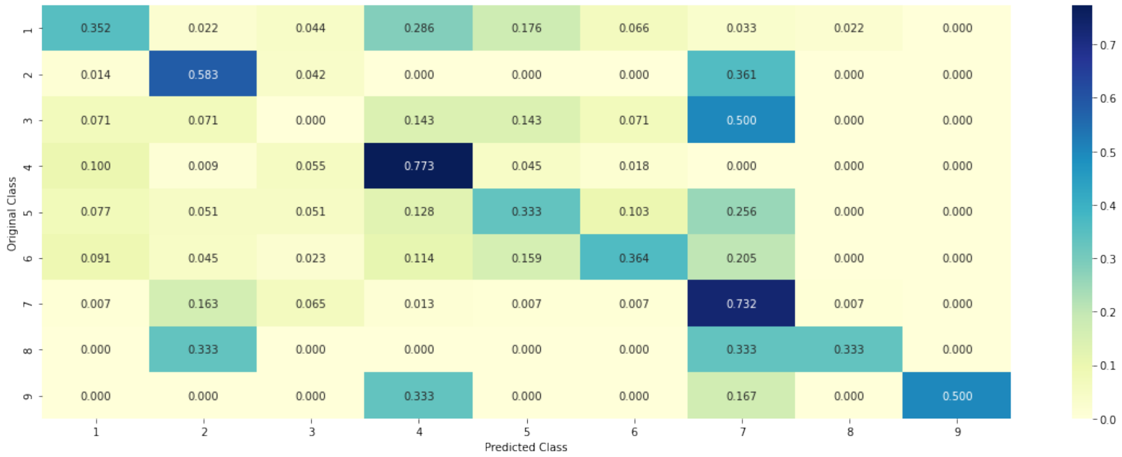
```
Log loss : 1.362245277672608
Number of mis-classified points : 0.42857142857142855
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.352	0.022	0.044	0.286	0.176	0.066	0.033	0.022	0.000
2	0.014	0.583	0.042	0.000	0.000	0.000	0.361	0.000	0.000
3	0.071	0.071	0.000	0.143	0.143	0.071	0.500	0.000	0.000
4	0.100	0.009	0.055	0.773	0.045	0.018	0.000	0.000	0.000
5	0.077	0.051	0.051	0.128	0.333	0.103	0.256	0.000	0.000
6	0.091	0.045	0.023	0.114	0.159	0.364	0.205	0.000	0.000
7	0.007	0.163	0.065	0.013	0.007	0.007	0.732	0.007	0.000
8	0.000	0.333	0.000	0.000	0.000	0.000	0.333	0.333	0.000
9	0.000	0.000	0.000	0.333	0.000	0.000	0.167	0.000	0.500

[192]:
```
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)


test_point_index = 1
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 : 2
Predicted Class Probabilities: [[0.0149 0.3938 0.1427 0.0175 0.08   0.0266
0.2857 0.0162 0.0226]]
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
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
```

[193]:

```
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("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: [[0.0027 0.0049 0.0052 0.005  0.0032 0.0068
0.9649 0.003  0.0042]]
```

```
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
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
```

15 Stacking all the models

[194]:

```
clf1 = SGDClassifier(alpha=0.001, 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=1, 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=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
```

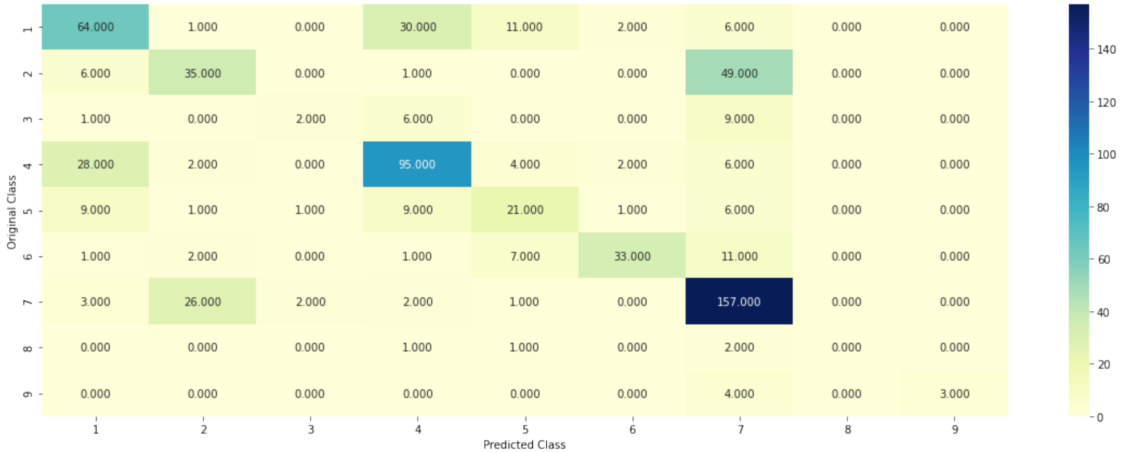
```
sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.
 ↪predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.
 ↪predict_proba(cv_x_onehotCoding))))
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: 1.23
Support vector machines : Log Loss: 1.70
Naive Bayes : Log Loss: 1.32
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.818
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.725
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.353
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.254
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.538
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.859
```

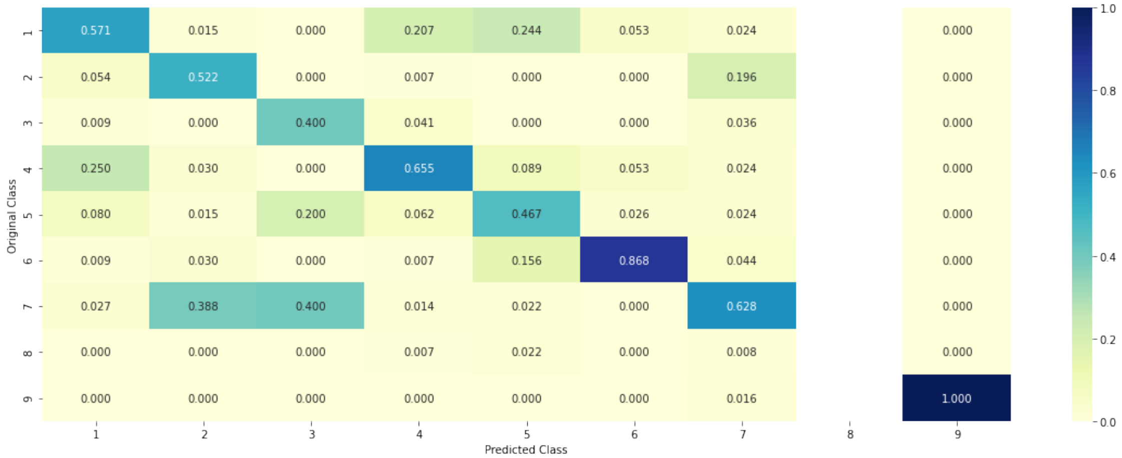
[195]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3],␣
 ↪meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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))
```

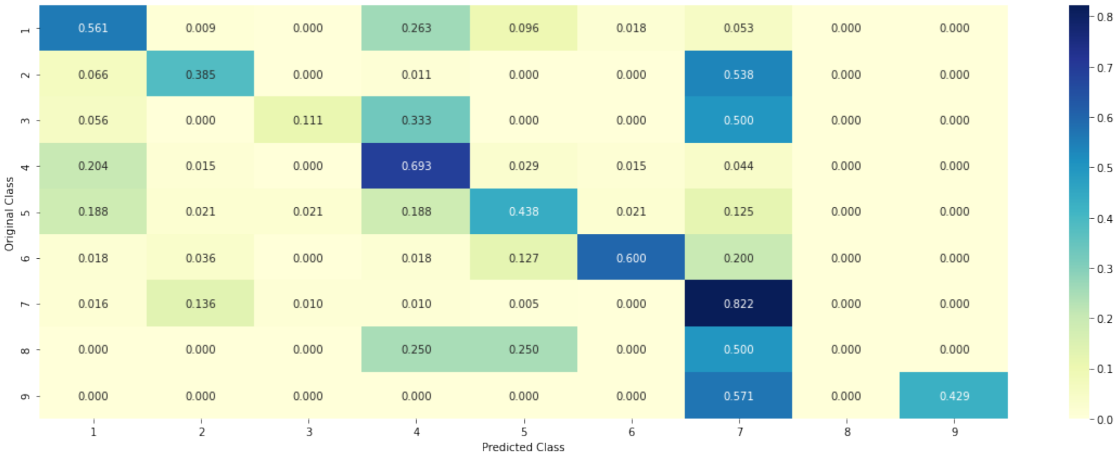
```
Log loss (train) on the stacking classifier : 0.49384508011202294
Log loss (CV) on the stacking classifier : 1.2538824276164817
Log loss (test) on the stacking classifier : 1.189855952174517
Number of missclassified point : 0.38345864661654133
-------------------- Confusion matrix --------------------
```



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



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

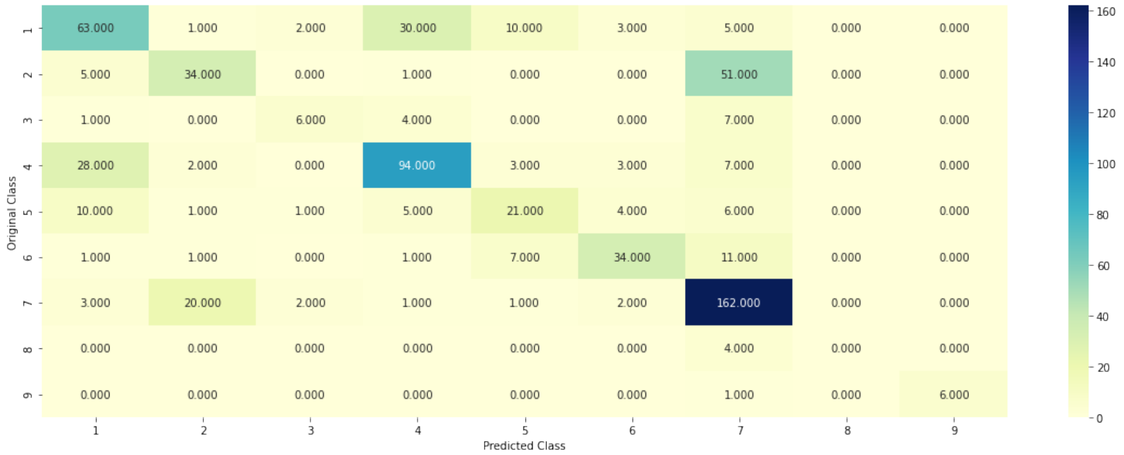


16 Maximum voting classifier

[196]:

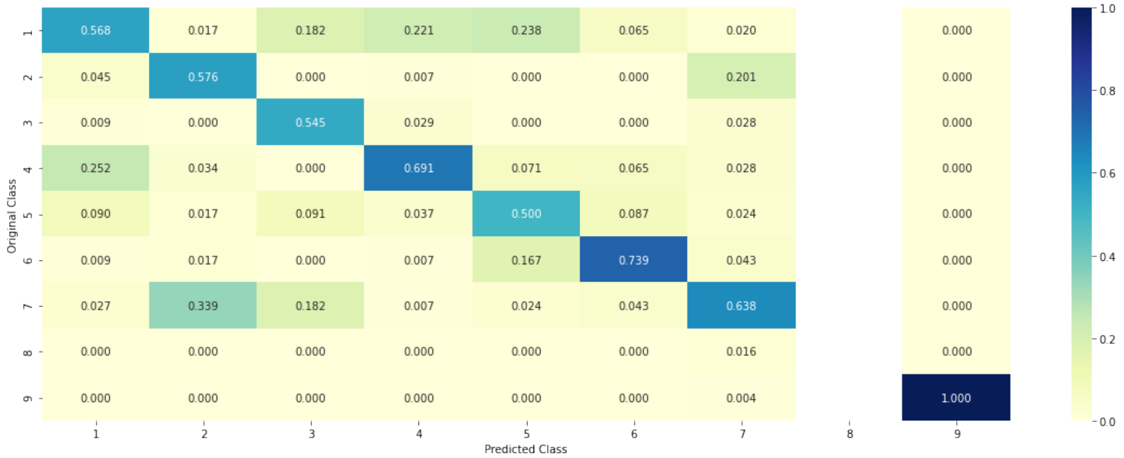
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf',␣
 ↪sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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 : 0.8636667527166619
Log loss (CV) on the VotingClassifier : 1.251699107781364
Log loss (test) on the VotingClassifier : 1.205411291874214
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



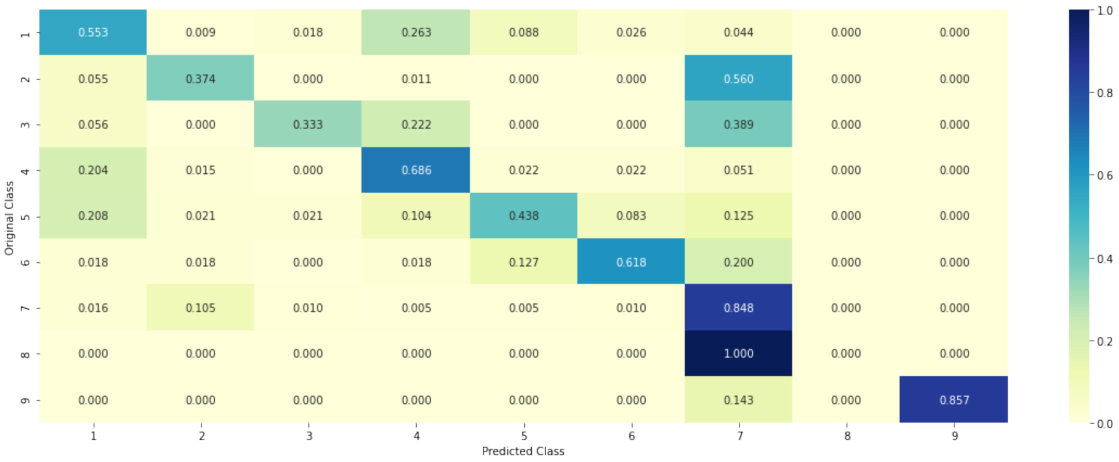
	1	2	3	4	5	6	7	8	9
1	63.000	1.000	2.000	30.000	10.000	3.000	5.000	0.000	0.000
2	5.000	34.000	0.000	1.000	0.000	0.000	51.000	0.000	0.000
3	1.000	0.000	6.000	4.000	0.000	0.000	7.000	0.000	0.000
4	28.000	2.000	0.000	94.000	3.000	3.000	7.000	0.000	0.000
5	10.000	1.000	1.000	5.000	21.000	4.000	6.000	0.000	0.000
6	1.000	1.000	0.000	1.000	7.000	34.000	11.000	0.000	0.000
7	3.000	20.000	2.000	1.000	1.000	2.000	162.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.568	0.017	0.182	0.221	0.238	0.065	0.020		0.000
2	0.045	0.576	0.000	0.007	0.000	0.000	0.201		0.000
3	0.009	0.000	0.545	0.029	0.000	0.000	0.028		0.000
4	0.252	0.034	0.000	0.691	0.071	0.065	0.028		0.000
5	0.090	0.017	0.091	0.037	0.500	0.087	0.024		0.000
6	0.009	0.017	0.000	0.007	0.167	0.739	0.043		0.000
7	0.027	0.339	0.182	0.007	0.024	0.043	0.638		0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.016		0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.004		1.000

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


Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
1: 0.553 0.009 0.018 0.263 0.088 0.026 0.044 0.000 0.000
2: 0.055 0.374 0.000 0.011 0.000 0.000 0.560 0.000 0.000
3: 0.056 0.000 0.333 0.222 0.000 0.000 0.389 0.000 0.000
4: 0.204 0.015 0.000 0.686 0.022 0.022 0.051 0.000 0.000
5: 0.208 0.021 0.021 0.104 0.438 0.083 0.125 0.000 0.000
6: 0.018 0.018 0.000 0.018 0.127 0.618 0.200 0.000 0.000
7: 0.016 0.105 0.010 0.005 0.005 0.010 0.848 0.000 0.000
8: 0.000 0.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000
9: 0.000 0.000 0.000 0.000 0.000 0.000 0.143 0.000 0.857
1.0
0.8
0.6
0.4
0.2
0.0

```
[ ]:
```