Prediction of Cancer using Machine learning

Problem statement:

To develop algorithms to classify genetic mutations based on clinical evidence (text).

Data overview:

We have considered the data from a Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/ (https://www.kaggle.com/c/msk-redefining-cancer-treatment/)

There are 2 files namely in our consideration:

- training_variants (ID , Gene, Variations, Class)
- training_text (ID, Text)

training_variants

```
ID,Gene,Variation,Class
0,FAM58A,Truncating Mutations,1
1,CBL,W802*,2
2,CBL,Q249E,2
...
```

training_text

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

So, as per our analysis, there are 9 different types(classes) of cancer namely from 1-9.

Exploratory data analysis

In [104]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import  re
import time
import warnings
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import names,words,stopwords
from tqdm import tqdm
from nltk.corpus.reader import WordListCorpusReader
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss,confusion_matrix
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from collections import defaultdict,Counter
from sklearn.preprocessing import normalize
from scipy.sparse import hstack
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.datasets import load_iris
from mlxtend.classifier import StackingClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

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

Reading Data

Reading Gene Data and Variation data

In [105]:

```
data = pd.read_csv('Datasets/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head(10)
```

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

Out[105]:

	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
5	5	CBL	V391I	4
6	6	CBL	V430M	5
7	7	CBL	Deletion	1
8	8	CBL	Y371H	4
9	9	CBL	C384R	4

Reading Text Data

```
In [106]: # note the seprator in this file
          data_text =pd.read_csv("Datasets/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],s
          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(10)
```

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

Out[106]:

	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...
5	5	Oncogenic mutations in the monomeric Casitas B...
6	6	Oncogenic mutations in the monomeric Casitas B...
7	7	CBL is a negative regulator of activated recep...
8	8	Abstract Juvenile myelomonocytic leukemia (JM...
9	9	Abstract Juvenile myelomonocytic leukemia (JM...

Preprocessing of text

In [107]:
```python
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [108]:
```
#text processing stage.
start_time = time.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.time() - start_time, "seconds")
```

```
/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2536347838.py:20:  SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/i
ndexing.html#returning-a-view-versus-a-copy  (https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy)
  data_text[column][index] = string

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 : 22.476922273635864 seconds
```

In [109]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[109]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

```
In [110]: result[result.isnull().any(axis=1)]
```

Out[110]:

	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

```
In [111]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [112]: result[result['ID']==1109]
```

Out[112]:

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

Test, Train and Cross Validation Split

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

In [113]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

```
/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2448509936.py:2:  FutureWarning
: The default value of regex will change from True to False in a future version.
  result.Gene      = result.Gene.str.replace('\s+', '_')
/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2448509936.py:3:  FutureWarning
: The default value of regex will change from True to False in a future version.
  result.Variation = result.Variation.str.replace('\s+', '_')
```

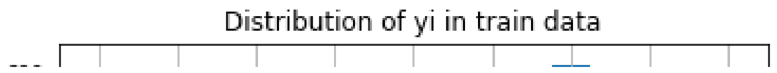
In [114]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

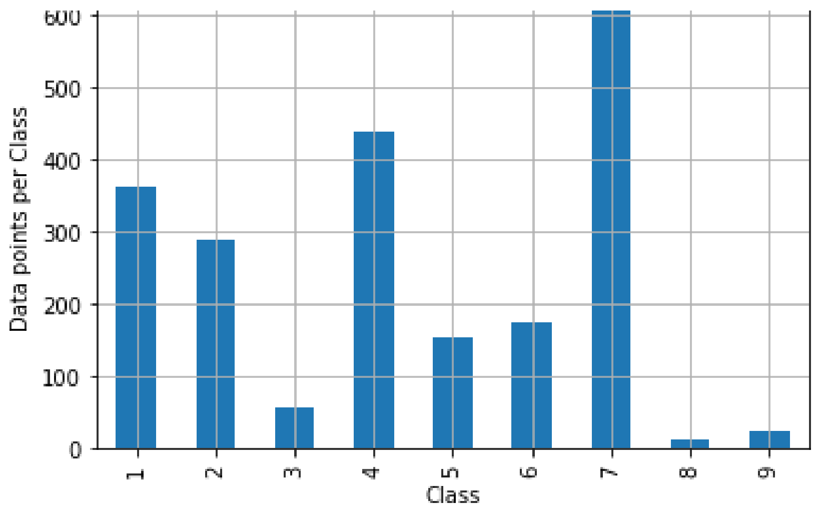
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

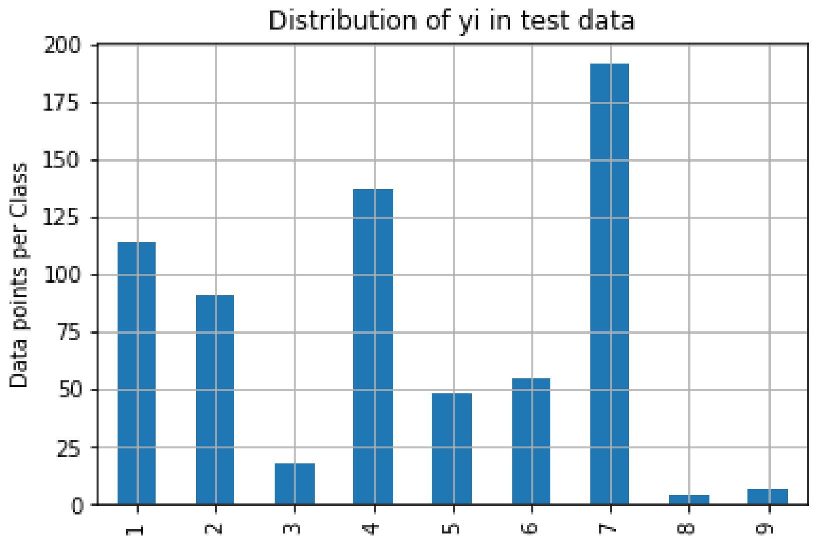
In [115]:
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution  =  train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution  =  cv_df['Class'].value_counts().sort_index()

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()
```

```python
# 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


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.

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.ro
```

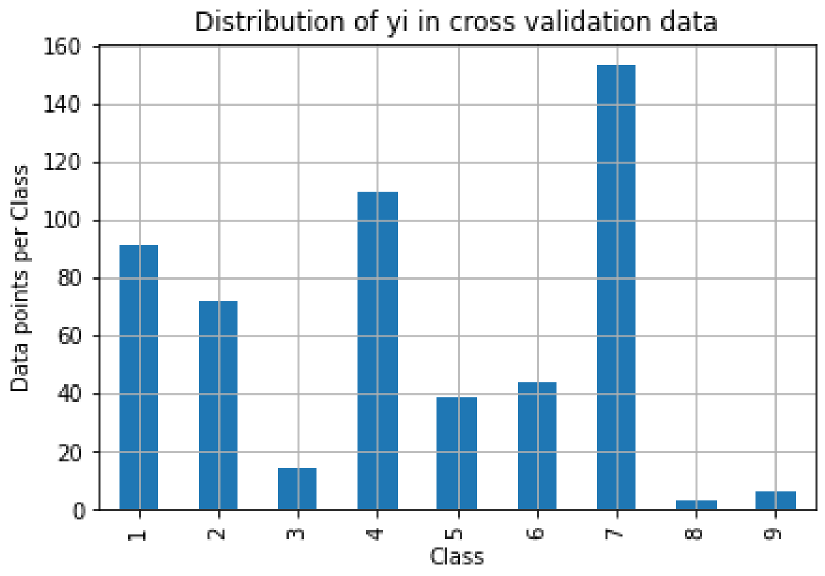




```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 %)
```

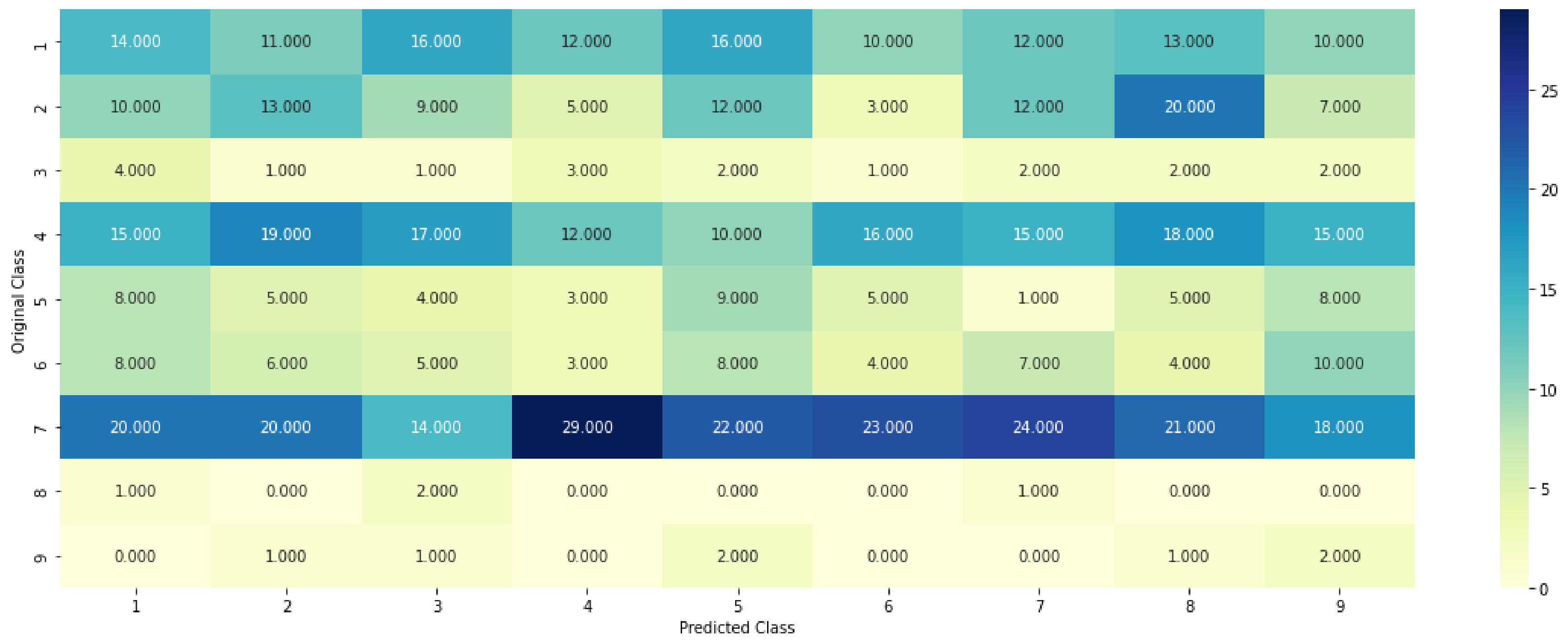
Prediction using a 'Random' Model

In [116]:
```
# 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

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

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

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

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

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

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

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

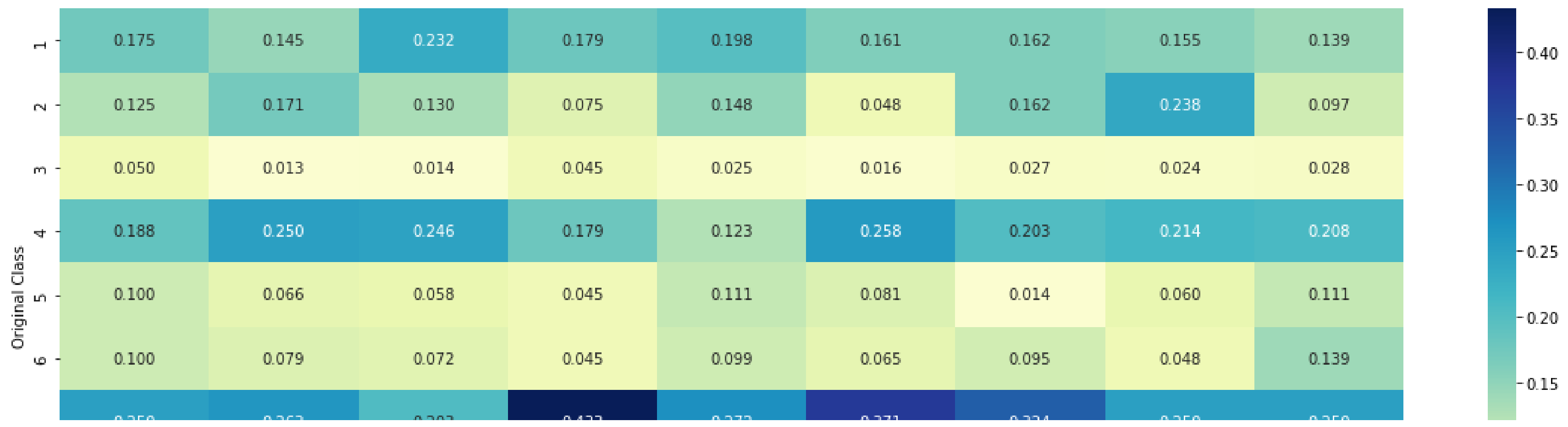
In [117]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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=1


# Test-Set error.
#we create a output array that has exactly same size 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)
```

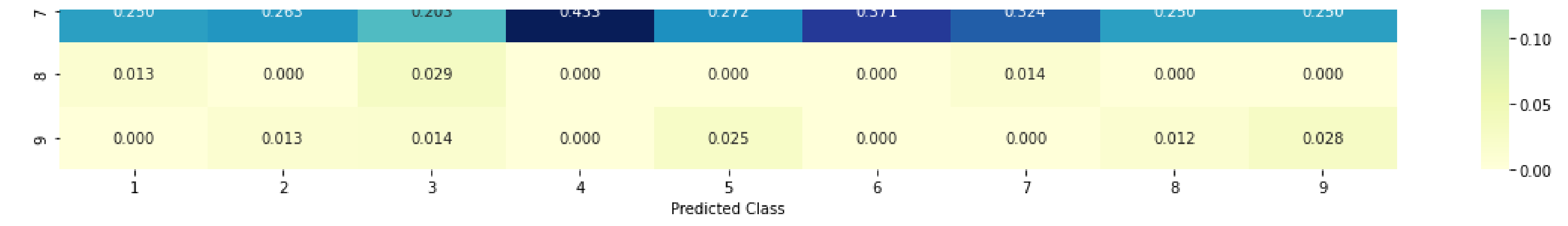
```
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.48811978981526
Log loss on Test Data using Random Model 2.5056858755410767
-------------------- Confusion matrix --------------------
```

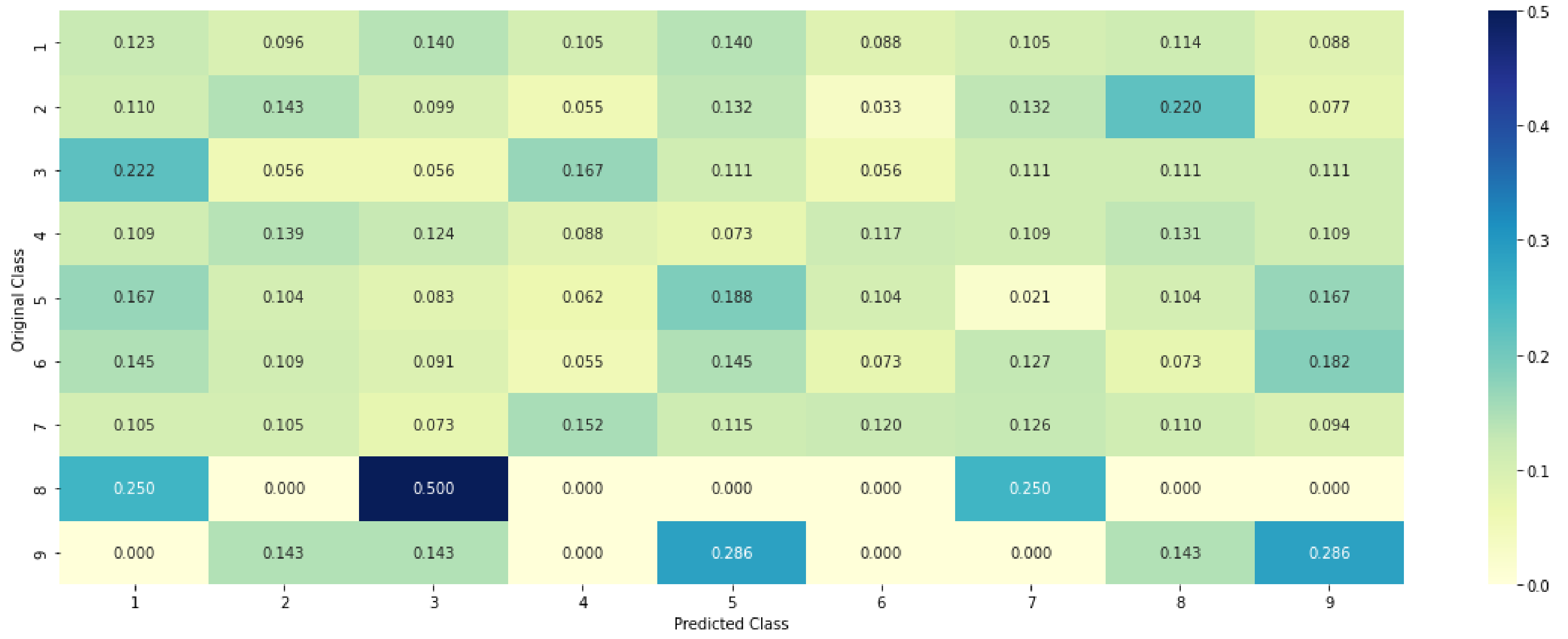


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Univariate Analysis

In [118]:

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

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    #  print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    #  print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()
    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variatio
    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 clas
        # 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 featur
            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': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.136363636
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.270408163
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.0
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.07878787
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.0728476821
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333
    #      ...
    #     }
    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
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

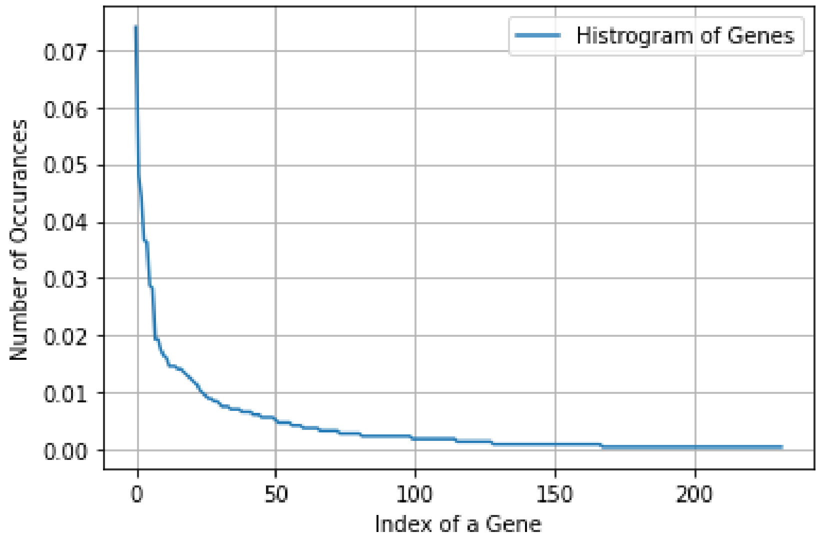
```
In [119]: unique_genes = train_df['Gene'].value_counts()
          print('Number of Unique Genes :', unique_genes.shape[0])
          # the top 10 genes that occured most
          print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1     157
TP53      102
EGFR       94
BRCA2      78
PTEN       77
BRAF       61
KIT        60
ERBB2      41
ALK        41
PIK3CA     37
Name: Gene, dtype: int64
```

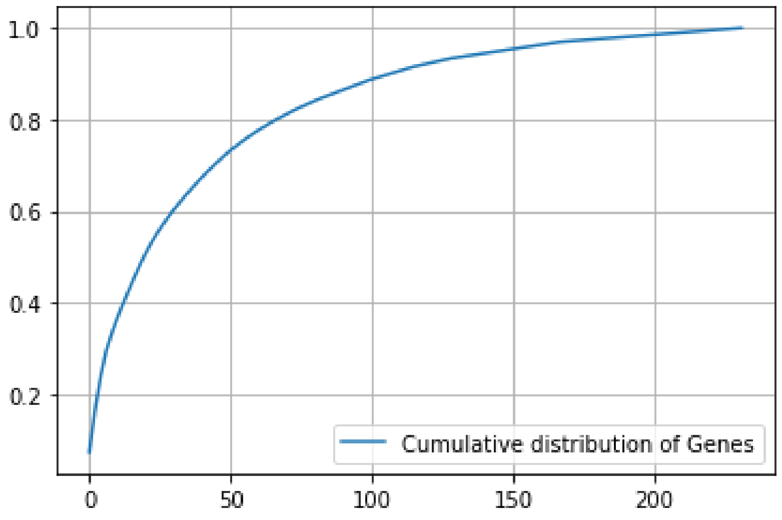
```
In [120]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data,
```

```
Ans: There are 232 different categories of genes in the train data, and they are distibuted as
follows
```

```
In [121]: 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()
```



In [122]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [123]:
```
#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))
```

In [124]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The s
```

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

In [125]:
```
# 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'])
```

In [126]:
```
train_df['Gene'].head()
```

Out[126]:
```
2323      JAK2
3204       RB1
3236     NTRK3
2107       B2M
2377    PTPN11
Name: Gene, dtype: object
```

In [127]:
```
gene_vectorizer.get_feature_names()
```

```
/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[127]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
```

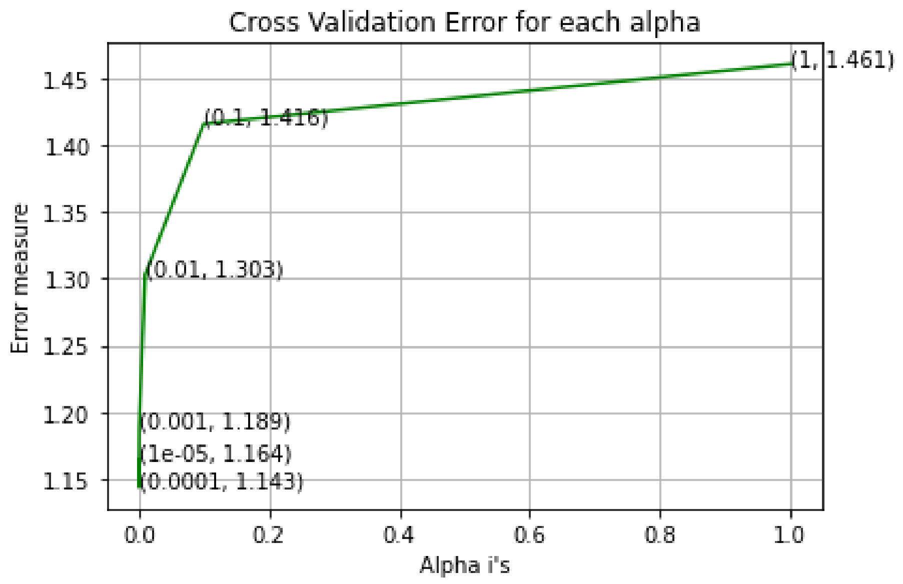
In [128]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The s
```

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

Q4. How good is this gene feature in predicting y_i?
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [129]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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


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

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_trai
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_
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,
```

```
For values of alpha = 1e-05 The log loss is: 1.1643878127901701
For values of alpha = 0.0001 The log loss is: 1.1434080312413728
For values of alpha = 0.001 The log loss is: 1.1891586887232262
For values of alpha =  0.01 The log loss is: 1.3029120258077458
For values of alpha =  0.1 The log loss is: 1.4161374096250532
For values of alpha = 1 The log loss is: 1.4609516366376099
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0101912269436284
For values of best alpha = 0.0001 The cross validation log loss is: 1.1434080312413728
For values of best alpha = 0.0001 The test log loss is: 1.1477052022797556
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

In [130]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape

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_d
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_d
```

```
Q6. How many data points in Test and CV datasets are covered by the 232  genes in train datase
t?
Ans
1. In test data 637 out of 665 : 95.78947368421052
2. In cross validation data 519 out of  532 : 97.55639097744361
```

Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

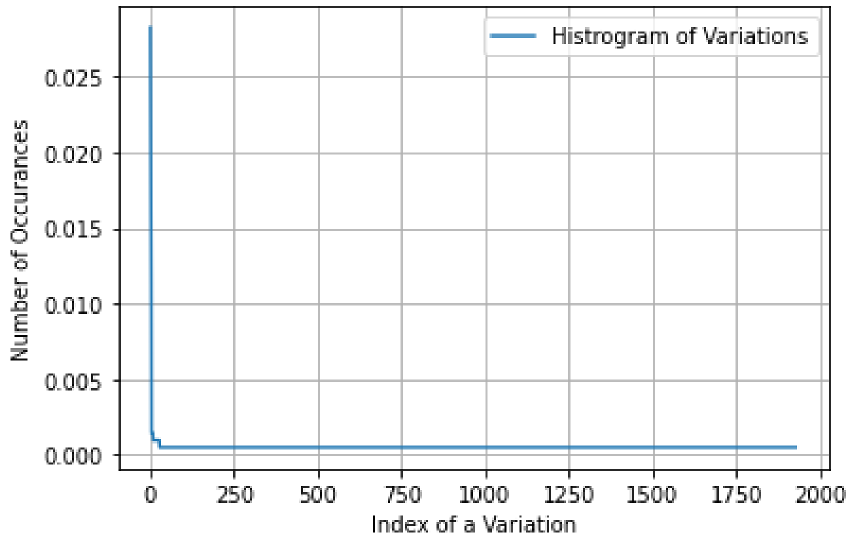
In [131]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1928
Truncating_Mutations     60
Deletion                 50
Amplification            37
Fusions                  24
Overexpression            3
Q61L                      3
E17K                      3
T58I                      3
G12V                      3
P34R                      2
Name: Variation, dtype: int64
```

In [132]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the t
```

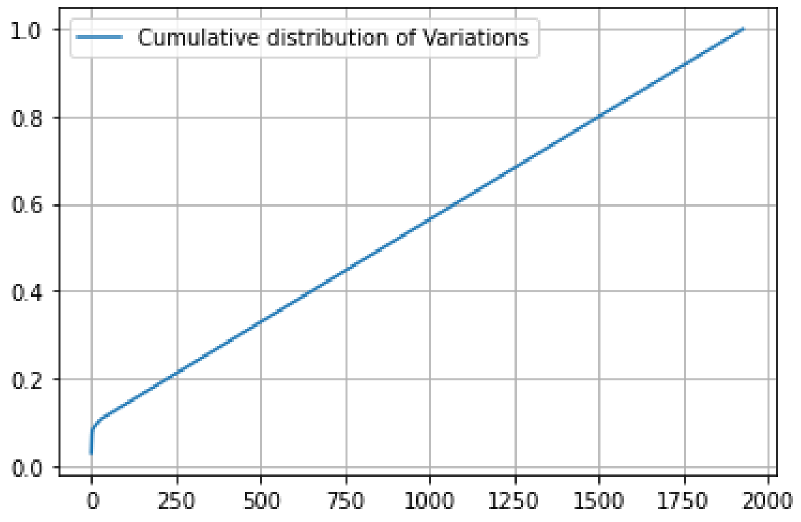
```
Ans: There are 1928 different categories of variations in the train data, and they are distibut
ed as follows
```

In [133]:
```
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()
```



```
In [134]: c = np.cumsum(h)
          print(c)
          plt.plot(c,label='Cumulative distribution of Variations')
          plt.grid()
          plt.legend()
          plt.show()
```

```
[0.02824859 0.05178908 0.06920904 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [135]:
```
# 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))
```

In [136]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding m
```

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

In [137]:
```
# 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'])
```

In [138]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding me
```

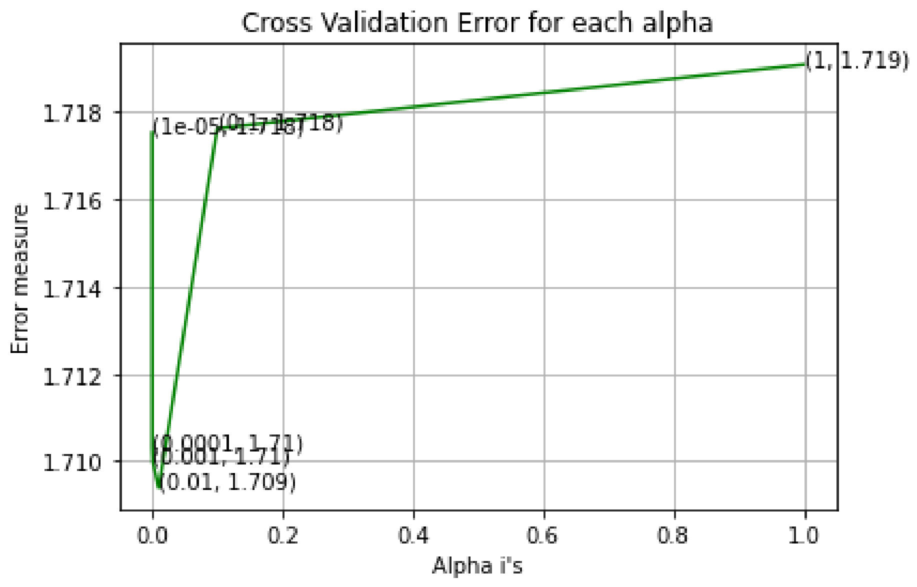
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
The shape of Variation feature: (2124, 1960)
```

Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [139]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.li
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', et
# class_weight=None, warm_start=False, average=False, n_iter=None)
# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)     Predict class labels for samples in X.

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding,    y_train)
    predict_y  =  sig_clf.predict_proba(cv_variation_feature_onehotCoding)

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

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_trai
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_
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,
```

```
For values of alpha = 1e-05 The log loss is: 1.7175297984144506
For values of alpha = 0.0001 The log loss is: 1.7103082563607537
For values of alpha = 0.001 The log loss is: 1.7099797451317027
For values of alpha =  0.01 The log loss is: 1.7093896202282575
For values of alpha = 0.1 The log loss is: 1.717626637506973
For values of alpha =  1 The log loss is: 1.7190882965879752
```



```
For values of best alpha =  0.01 The train log loss is: 1.4176536368773618
For values of best alpha = 0.01 The cross validation log loss is: 1.7093896202282575
For values of best alpha = 0.01 The test log loss is: 1.7301916010014
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Not sure! But lets be very sure using the below analysis.

In [140]:
```python
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in
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_d
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_d
```

```
Q12. How many data points are covered by total 1928 genes in test and cross validation data s
ets?
Ans
1. In test data 79 out of 665 : 11.879699248120302
2. In cross validation data 47 out of  532 : 8.834586466165414
```

Univariate Analysis on Text Feature

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

In [141]:
```python
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [142]:
```
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
```

In [143]:
```
# 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 fea
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 : 53540

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

In [144]:
```
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)
```

In [145]:
```
#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)
```

In [146]:
```
#  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_resp
test_text_feature_responseCoding  =  (test_text_feature_responseCoding.T/test_text_feature_respons
cv_text_feature_responseCoding  =  (cv_text_feature_responseCoding.T/cv_text_feature_responseCodin
```

In [147]:
```
# 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)
```

In [148]:
```
#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()))
```

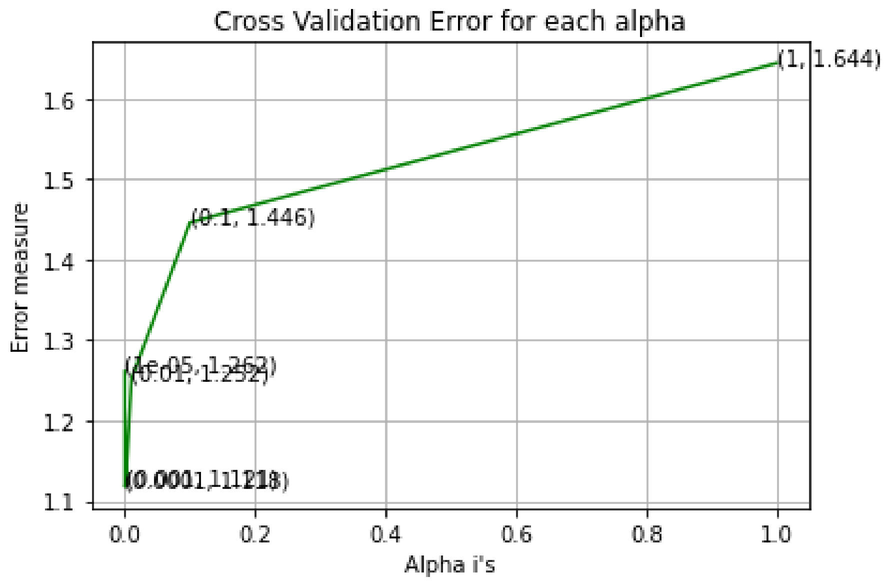
In [149]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 5173, 4: 3688, 6: 2963, 5: 2789, 8: 2260, 7: 2025, 9: 1848, 12: 1321, 11: 1307, 10:
1288, 13: 1238, 14: 875, 16: 776, 15: 762, 18: 743, 17: 692, 20: 581, 24: 507, 21: 482, 19: 478
, 22: 460, 23: 424, 25: 421, 33: 403, 26: 398, 30: 350, 27: 346, 36: 324, 44: 315, 28: 315, 32:
304, 29: 275, 34: 271, 31: 260, 37: 237, 42: 227, 35: 225, 40: 219, 45: 215, 38: 208, 50: 192,
49: 188, 39: 188, 46: 185, 48: 180, 43: 172, 41: 164, 54: 154, 47: 154, 53: 146, 52: 139, 57: 1
34, 56: 123, 51: 123, 62: 117, 61: 117, 64: 114, 55: 114, 60: 113, 59: 113, 72: 111, 65: 111, 6
8: 109, 63: 106, 58: 106, 74: 103, 70: 97, 69: 97, 76: 96, 66: 94, 80: 93, 67: 92, 73: 91, 81:
88, 84: 85, 78: 85, 83: 84, 77: 84, 71: 83, 85: 77, 79: 75, 88: 74, 91: 73, 86: 73, 75: 66, 111
: 65, 93: 64, 92: 64, 89: 64, 82: 63, 96: 60, 103: 59, 87: 59, 115: 58, 95: 58, 90: 58, 100: 55
, 120: 54, 98: 54, 99: 53, 106: 52, 108: 51, 119: 50, 97: 50, 117: 49, 105: 48, 144: 46, 122: 4
5, 104: 45, 94: 45, 114: 44, 112: 44, 102: 44, 147: 42, 123: 42, 113: 42, 110: 42, 109: 41, 101
: 41, 124: 40, 116: 40, 128: 39, 133: 38, 132: 38, 126: 38, 138: 37, 139: 36, 134: 36, 129: 36,
107: 36, 141: 35, 140: 35, 118: 35, 160: 34, 175: 33, 145: 33, 143: 33, 135: 33, 131: 33, 156:
32, 148: 32, 125: 32, 153: 31, 152: 31, 142: 31, 154: 30, 127: 30, 121: 30, 182: 29, 150: 29, 1
36: 29, 169: 28, 167: 28, 164: 28, 149: 28, 130: 28, 174: 27, 168: 27, 166: 27, 165: 27, 157: 2
7, 155: 27, 183: 26, 158: 26, 146: 26, 213: 25, 210: 25, 203: 25, 187: 25, 172: 25, 159: 25, 15
1: 25, 272: 24, 230: 24, 178: 24, 177: 24, 264: 23, 224: 23, 209: 23, 200: 23, 198: 23, 181: 23
, 170: 23, 162: 23, 161: 23, 137: 23, 192: 22, 190: 22, 185: 22, 176: 22, 171: 22, 163: 22, 262
: 21, 226: 21, 216: 21, 201: 21, 188: 21, 184: 21, 302: 20, 285: 20, 254: 20, 211: 20, 208: 20,
```

In [150]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

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

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

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

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_trai
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_
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,
```

```
For values of alpha = 1e-05 The log loss is: 1.2615228528143771
For values of alpha = 0.0001 The log loss is: 1.1177556566400166
For values of alpha = 0.001 The log loss is: 1.120800570242757
For values of alpha = 0.01 The log loss is: 1.2517234789517966
For values of alpha = 0.1 The log loss is: 1.4459742660680286
For values of alpha = 1 The log loss is: 1.6438571044531183
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6594726151209621
For values of best alpha = 0.0001 The cross validation log loss is: 1.1177556566400166
For values of best alpha = 0.0001 The test log loss is: 1.1446395511686538
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it seems like!

```
In [151]: 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
```

```
In [152]: 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")
```

```
/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg,  category=FutureWarning)

96.599 % of word of test data appeared in train data
97.392 % of word of Cross Validation appeared in train data
```

Machine Learning Models

In [153]:
```
#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)
```

In [154]:
```
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)
```

```
In [155]: # 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,ye
                  else:
                      word  = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
                      yes_no = True if word in text.split() else False
                      if yes_no:
                          word_present += 1
                          print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no)

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

Stacking the three types of features

In [156]:
```
# 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_on
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_oneho
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCodin

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).to
train_y = np.array(list(train_df['Class']))

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_fea
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_featur
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_resp

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_responseCo
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCodin
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
```

```
In [157]: print("One hot encoding features :")
          print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shap
          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_onehotCodi
```

```
One hot encoding features :
(number of data points * number of features) in train data = (2124, 55732)
(number of data points * number of features) in test data = (665, 55732)
(number of data points * number of features) in cross validation data = (532, 55732)
```

```
In [158]: print(" Response encoding features :")
          print("(number of data points * number of features) in train data = ", train_x_responseCoding.sh
          print("(number of data points * number of features) in test data = ", test_x_responseCoding.shap
          print("(number of data points * number of features) in cross validation data =", cv_x_responseCo
```

```
 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)
```

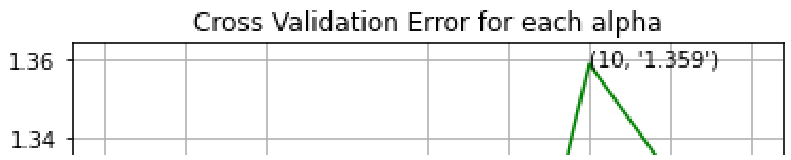
Machine Learning Models

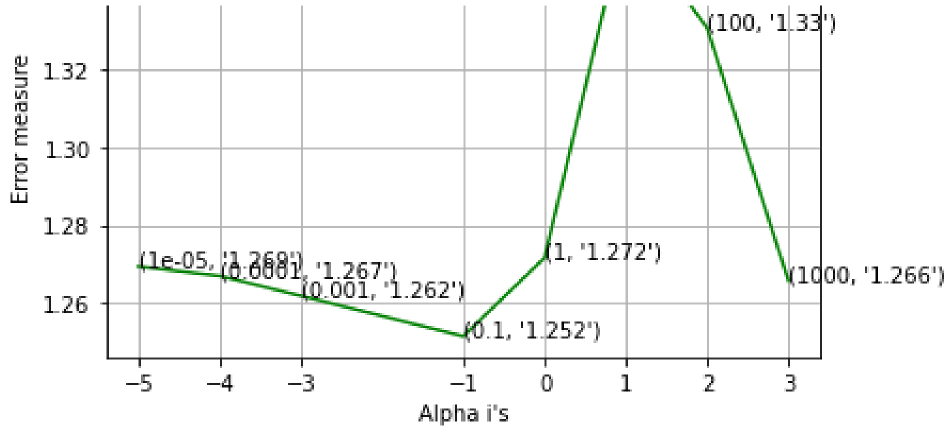
Naive Bayes

Hyper parameter tuning

```
In [159]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/ge
          # -------------------------
          # default paramters
          #  sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

          # some of methods of MultinomialNB()
          # fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
          # predict(X)    Perform classification on an array of test vectors X.
          # ------------------------
          #  video  link:  https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-baye
          # ------------------------


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


          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_trai
          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_
          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,
```

```
for alpha = 1e-05
Log Loss : 1.269474368549463
for alpha = 0.0001
Log Loss : 1.2670385240536137
for alpha = 0.001
Log Loss : 1.2619340912047825
for alpha = 0.1
Log Loss : 1.251587709087935
for alpha = 1
Log Loss : 1.2716731921288464
for alpha = 10
Log Loss : 1.3588783344079964
for alpha = 100
Log Loss : 1.3304358310222428
for alpha = 1000
Log Loss : 1.2658643173299036
```



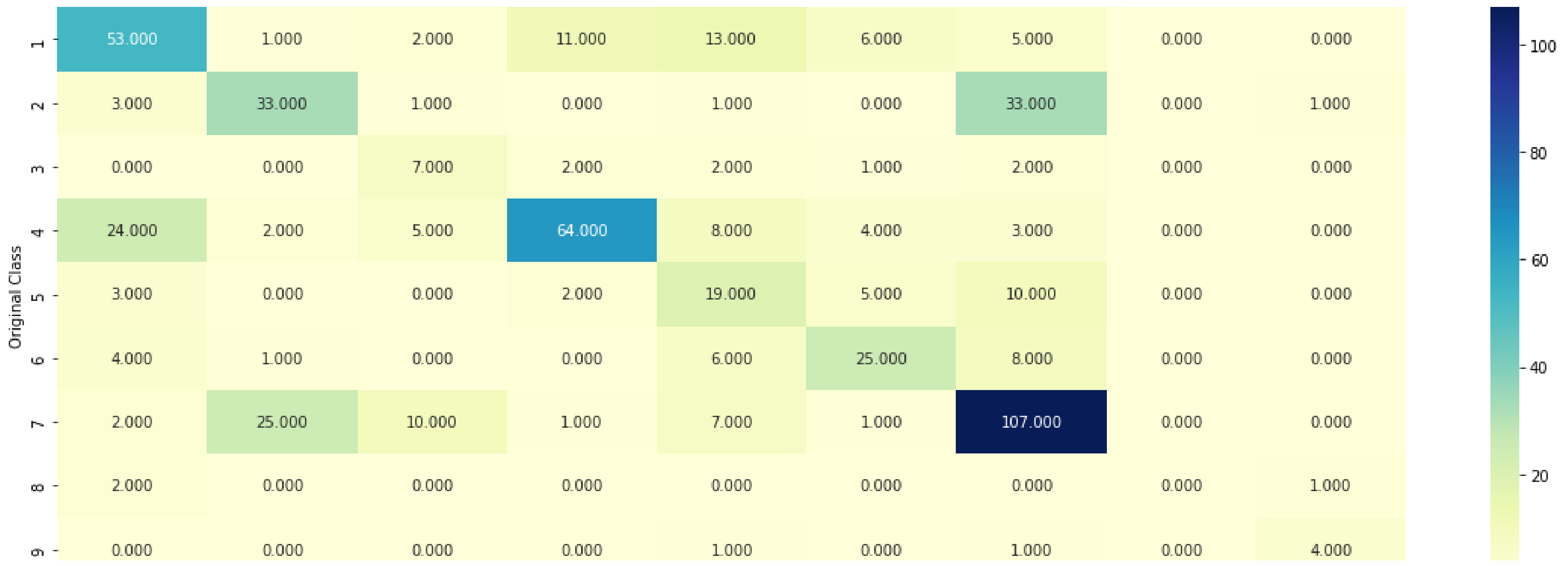


```
For values of best alpha =  0.1 The train log loss is: 0.8498373664517876
For values of best alpha = 0.1 The cross validation log loss is: 1.251587709087935
For values of best alpha = 0.1 The test log loss is: 1.2387104813224996
```

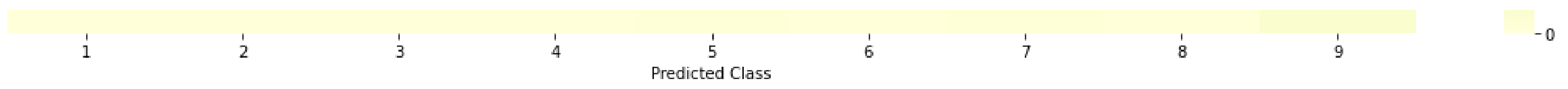
Testing the model with best hyper paramters

In [160]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/ge
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

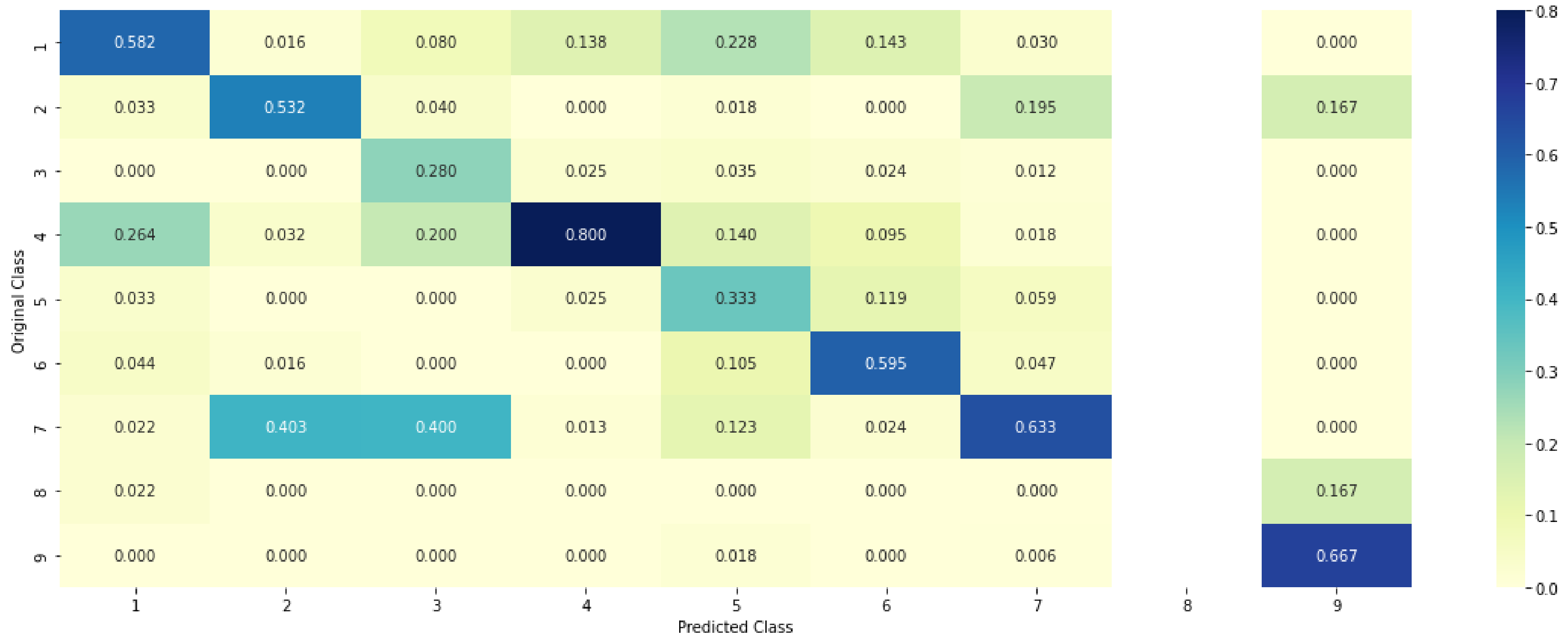
```
Log Loss : 1.251587709087935
Number of missclassified point : 0.41353383458646614

/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2633006593.py:22: RuntimeWarni
ng: invalid value encountered in true_divide
  B =(C/C.sum(axis=0))

-------------------- Confusion matrix --------------------
```



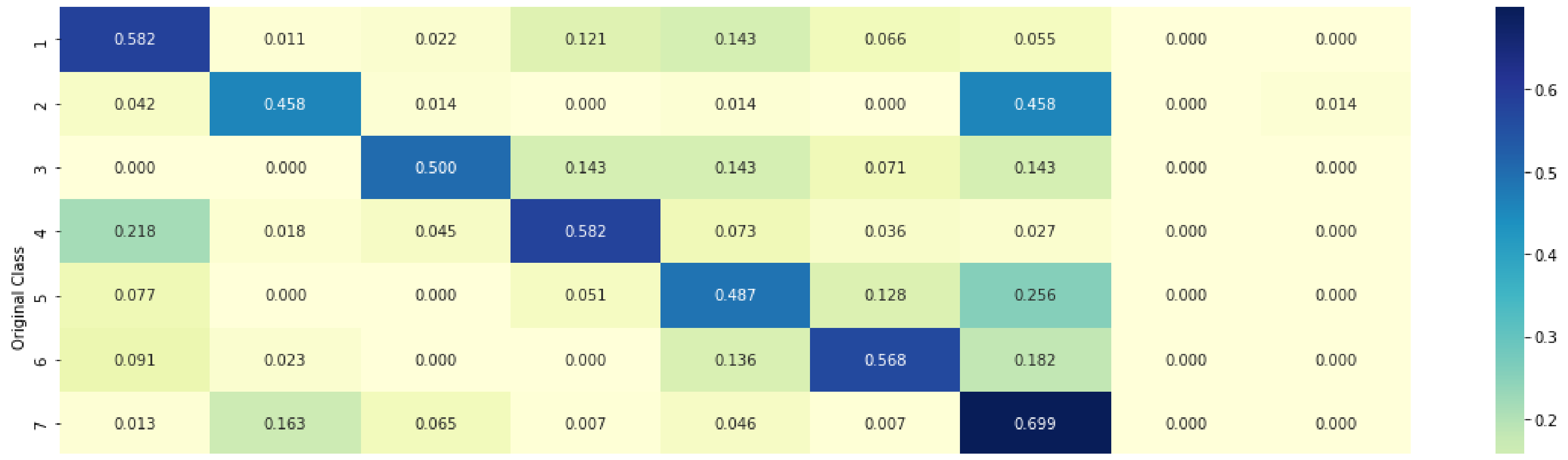
1	53.000	1.000	2.000	11.000	13.000	6.000	5.000	0.000	0.000
2	3.000	33.000	1.000	0.000	1.000	0.000	33.000	0.000	1.000
3	0.000	0.000	7.000	2.000	2.000	1.000	2.000	0.000	0.000
4	24.000	2.000	5.000	64.000	8.000	4.000	3.000	0.000	0.000
5	3.000	0.000	0.000	2.000	19.000	5.000	10.000	0.000	0.000
6	4.000	1.000	0.000	0.000	6.000	25.000	8.000	0.000	0.000
7	2.000	25.000	10.000	1.000	7.000	1.000	107.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	4.000

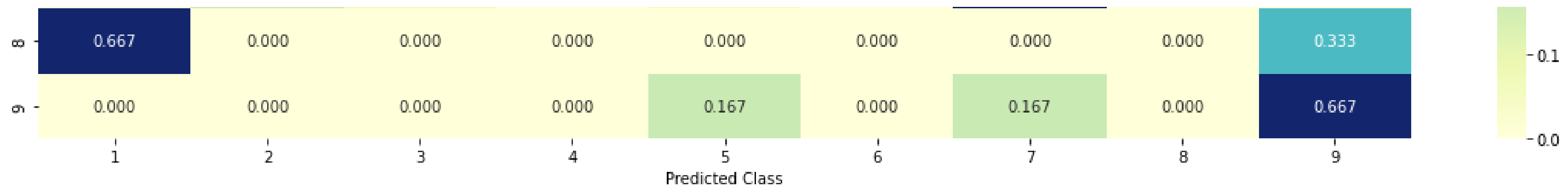


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





Feature Importance, Correctly classified point

In [161]:
```
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_
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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[tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0717 0.07   0.0129 0.1014 0.035  0.0325 0.6677 0.0051 0.0038
]]
Actual Class : 7
--------------------------------------------------

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:103: FutureWarning: Attribute `coef_` was deprecated in version 0.24 and will b
e removed in 1.1 (renaming of 0.26).
  warnings.warn(msg, category=FutureWarning)
/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

15 Text feature [presence] present in test data point [True]
16 Text feature [kinase] present in test data point [True]
17 Text feature [activating] present in test data point [True]
18 Text feature [downstream] present in test data point [True]
20 Text feature [well] present in test data point [True]
21 Text feature [independent] present in test data point [True]
```

```
22 Text feature [activation] present in test data point [True]
23 Text feature [also] present in test data point [True]
24 Text feature [contrast] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [recently] present in test data point [True]
27 Text feature [inhibitor] present in test data point [True]
28 Text feature [previously] present in test data point [True]
29 Text feature [shown] present in test data point [True]
30 Text feature [showed] present in test data point [True]
31 Text feature [cell] present in test data point [True]
33 Text feature [growth] present in test data point [True]
34 Text feature [similar] present in test data point [True]
35 Text feature [however] present in test data point [True]
36 Text feature [higher] present in test data point [True]
37 Text feature [compared] present in test data point [True]
38 Text feature [cells] present in test data point [True]
39 Text feature [factor] present in test data point [True]
40 Text feature [potential] present in test data point [True]
41 Text feature [suggest] present in test data point [True]
42 Text feature [may] present in test data point [True]
43 Text feature [addition] present in test data point [True]
44 Text feature [10] present in test data point [True]
45 Text feature [mutations] present in test data point [True]
46 Text feature [found] present in test data point [True]
47 Text feature [treated] present in test data point [True]
48 Text feature [studies] present in test data point [True]
49 Text feature [1a] present in test data point [True]
50 Text feature [described] present in test data point [True]
51 Text feature [observed] present in test data point [True]
52 Text feature [activated] present in test data point [True]
53 Text feature [without] present in test data point [True]
55 Text feature [3b] present in test data point [True]
56 Text feature [total] present in test data point [True]
57 Text feature [approximately] present in test data point [True]
61 Text feature [respectively] present in test data point [True]
62 Text feature [using] present in test data point [True]
63 Text feature [inhibition] present in test data point [True]
64 Text feature [inhibited] present in test data point [True]
65 Text feature [interestingly] present in test data point [True]
66 Text feature [12] present in test data point [True]
```

```
67 Text feature [including] present in test data point [True]
68 Text feature [3a] present in test data point [True]
69 Text feature [reported] present in test data point [True]
70 Text feature [confirmed] present in test data point [True]
71 Text feature [performed] present in test data point [True]
72 Text feature [signaling] present in test data point [True]
73 Text feature [whereas] present in test data point [True]
74 Text feature [various] present in test data point [True]
75 Text feature [fig] present in test data point [True]
76 Text feature [1b] present in test data point [True]
77 Text feature [confirm] present in test data point [True]
78 Text feature [identified] present in test data point [True]
79 Text feature [consistent] present in test data point [True]
80 Text feature [mutation] present in test data point [True]
81 Text feature [figure] present in test data point [True]
82 Text feature [different] present in test data point [True]
83 Text feature [although] present in test data point [True]
84 Text feature [constitutively] present in test data point [True]
86 Text feature [followed] present in test data point [True]
87 Text feature [report] present in test data point [True]
88 Text feature [proliferation] present in test data point [True]
89 Text feature [show] present in test data point [True]
90 Text feature [small] present in test data point [True]
91 Text feature [enhanced] present in test data point [True]
92 Text feature [two] present in test data point [True]
93 Text feature [suggests] present in test data point [True]
94 Text feature [results] present in test data point [True]
95 Text feature [three] present in test data point [True]
96 Text feature [inhibitors] present in test data point [True]
97 Text feature [discussion] present in test data point [True]
98 Text feature [15] present in test data point [True]
99 Text feature [demonstrated] present in test data point [True]
Out of the top  100  features  78 are present in query point
```

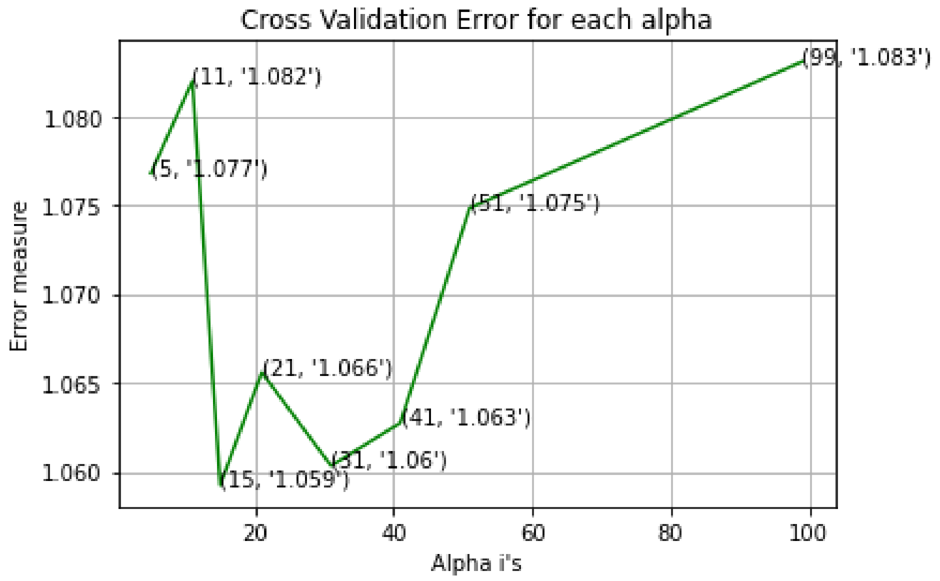
K Nearest Neighbour Classification

Hyper parameter tuning

In [162]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/s
# ----------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding,   train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 1.076789173707284
for alpha = 11
Log Loss : 1.0819935055130754
for alpha = 15
Log Loss : 1.0592279132071798
for alpha = 21
Log Loss : 1.065585295479777
for alpha = 31
```

```
Log Loss : 1.0603268946809172
for alpha = 41
Log Loss : 1.0627420839766668
for alpha = 51
Log Loss : 1.074838254434183
for alpha = 99
Log Loss : 1.0830821054719668
```

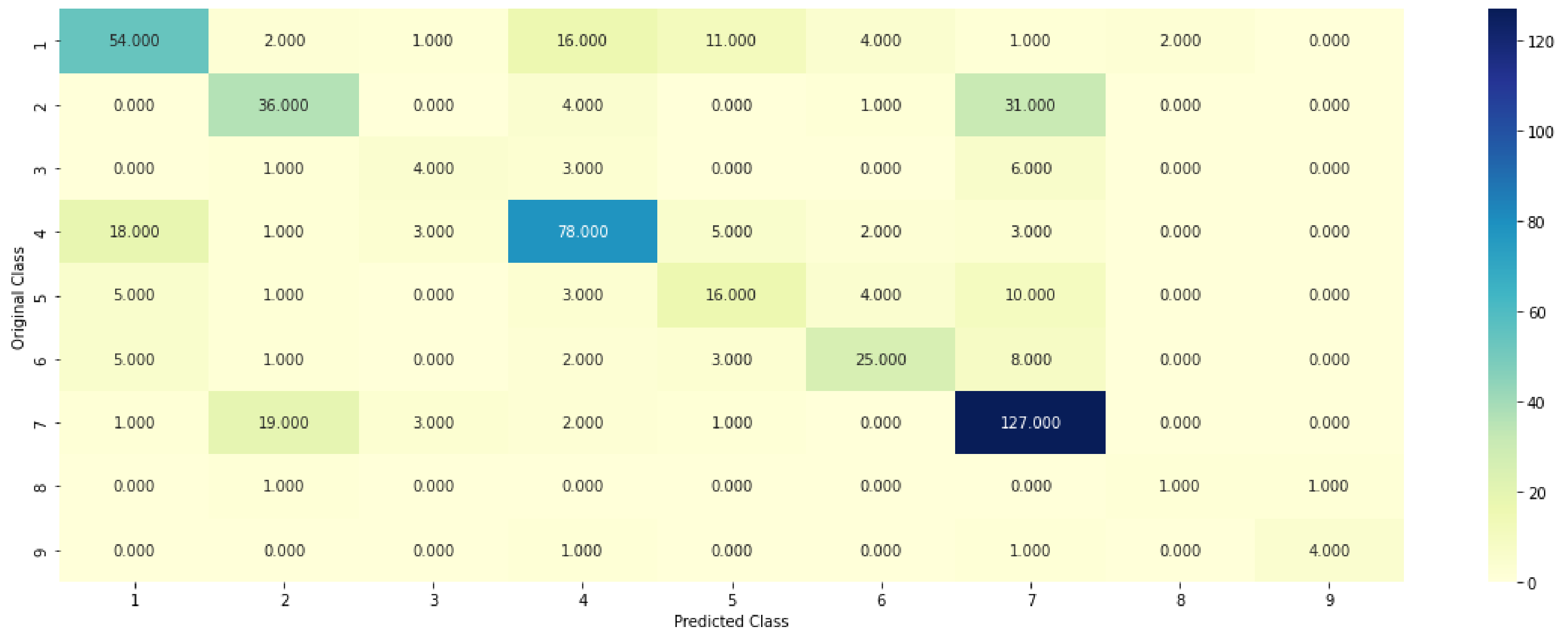


```
For values of best alpha =  15 The train log loss is: 0.6666402164254099
For values of best alpha = 15 The cross validation log loss is: 1.0592279132071798
For values of best alpha = 15 The test log loss is: 1.0278035715528273
```

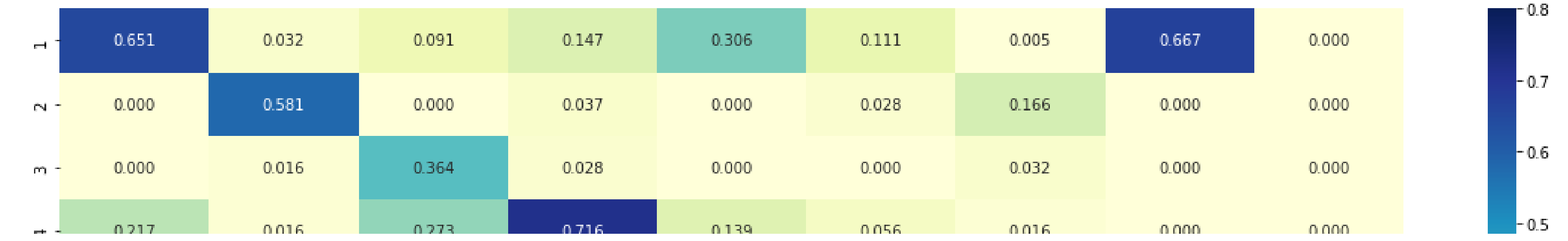
Testing the model with best hyper paramters

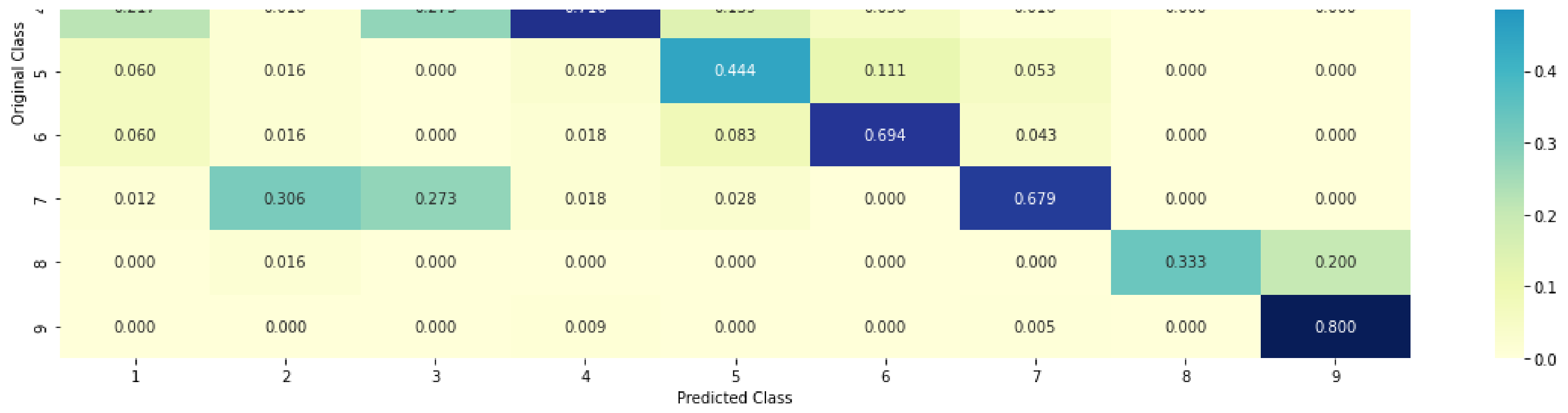
In [163]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/s
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
#  video  link:  https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, cl
```

```
Log loss : 1.0592279132071798
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```

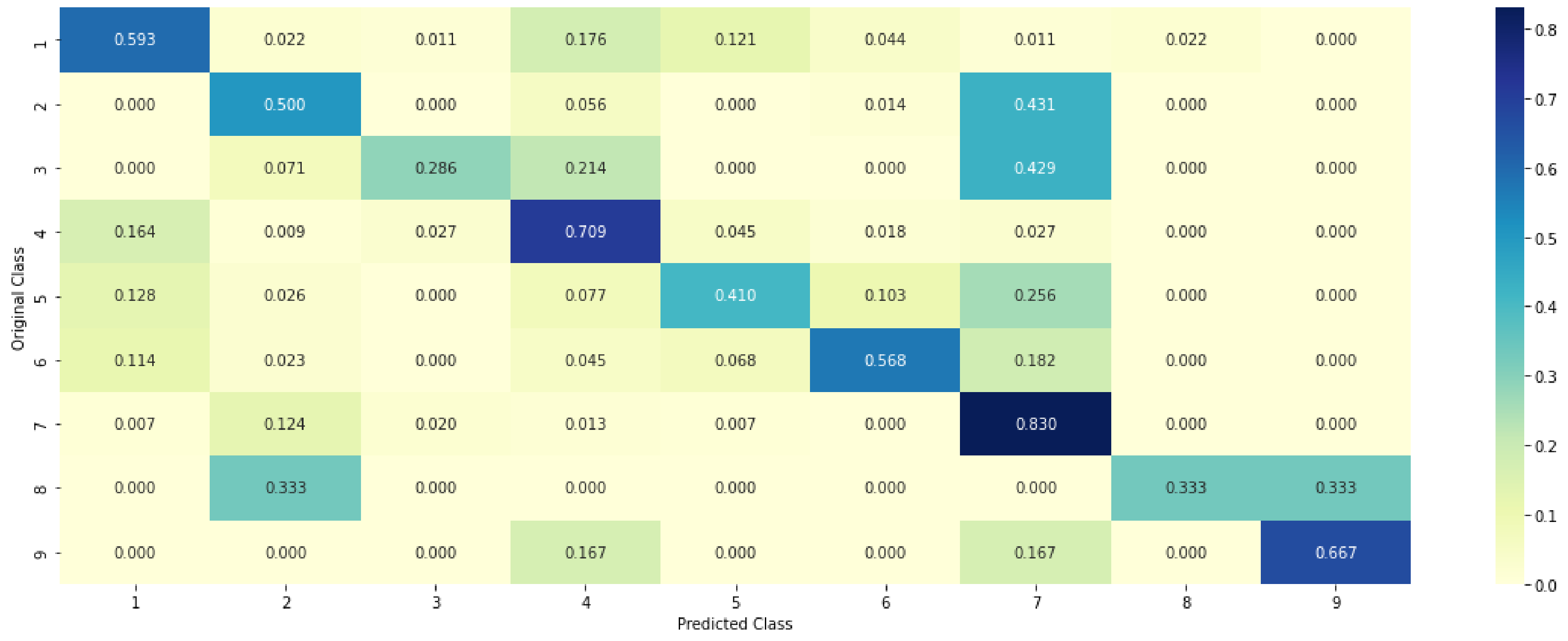


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Sample Query point -1

```
In [164]: 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_al
          print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train
          print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [5 3 3 7 7 7 7 7 7 7 7 7 3 7
7]
Fequency of nearest points : Counter({7: 11, 3: 3, 5: 1})
```

Sample Query point -2

In [165]:
```
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_al
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test points
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 2
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [2 7
2 2 7 7 2 7 2 2 2 2 2 2 6]
Fequency of nearest points : Counter({2: 10, 7: 4, 6: 1})
```

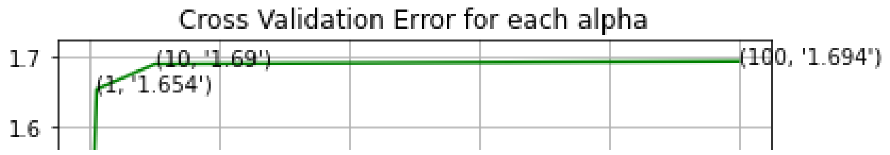
Logistic Regression

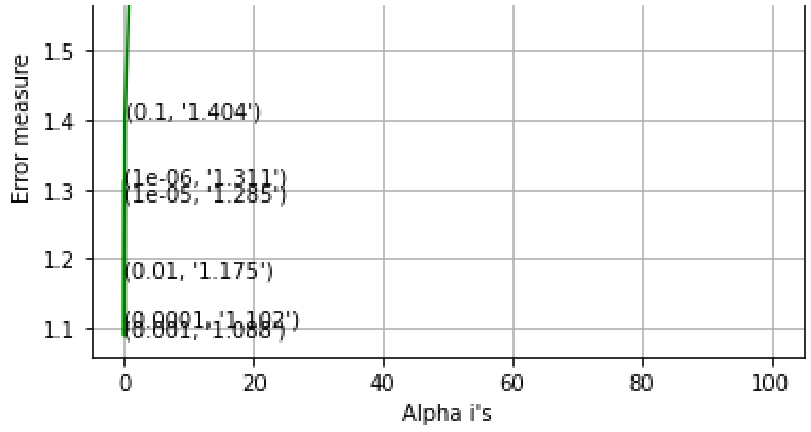
With Class Balancing

Hyper parameter tuning

In [166]:
```

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

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

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


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state
    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',
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_trai
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_
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,
```

```
for alpha = 1e-06
Log Loss : 1.3107377907899243
for alpha = 1e-05
Log Loss : 1.284502448857142
for alpha = 0.0001
Log Loss : 1.1024619822711859
for alpha = 0.001
Log Loss : 1.087543773076006
for alpha = 0.01
Log Loss : 1.1746456801387495
for alpha = 0.1
Log Loss : 1.4038980421857028
for alpha = 1
Log Loss : 1.65367971262763
for alpha = 10
Log Loss : 1.690057091633653
for alpha = 100
Log Loss : 1.6940595181771185
```



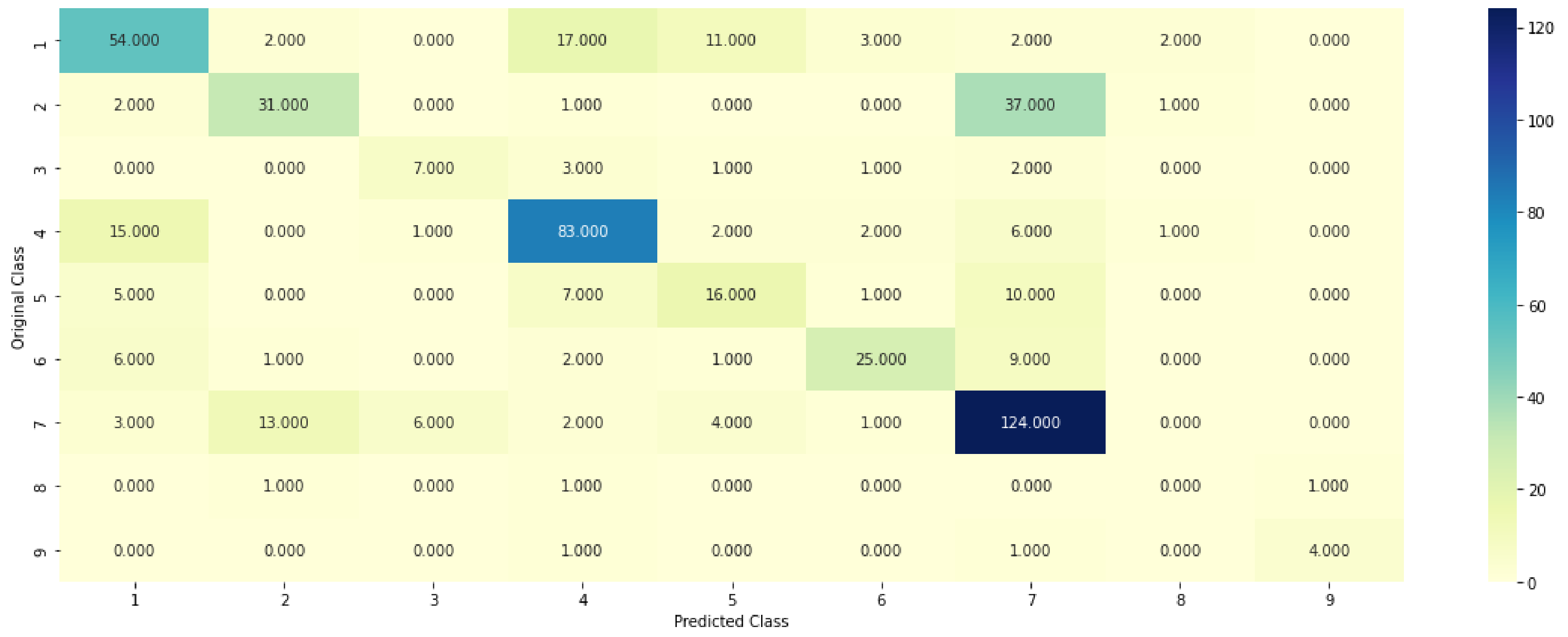


```
For values of best alpha =  0.001 The train log loss is: 0.5284856878959037
For values of best alpha = 0.001 The cross validation log loss is: 1.087543773076006
For values of best alpha = 0.001 The test log loss is: 1.0660614644085222
```

Testing the model with best Hyper parameters

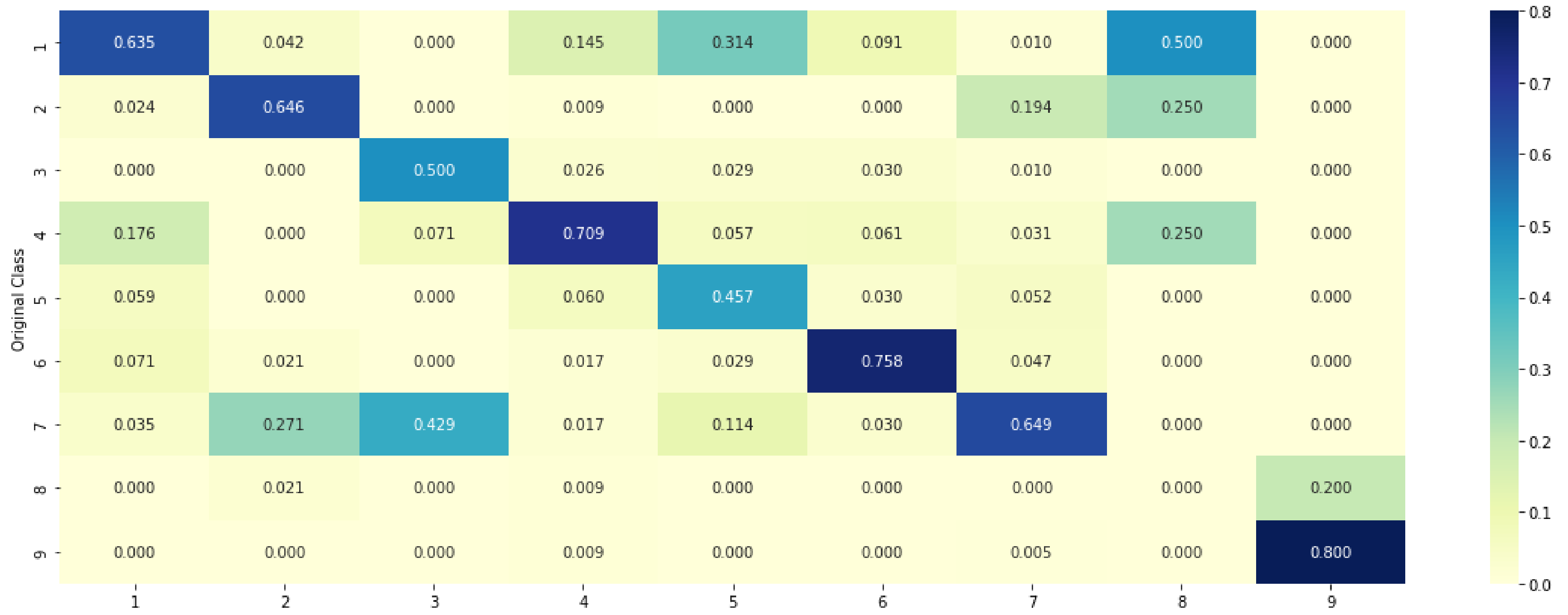
In [167]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.li
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', et
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.087543773076006
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



	1	2	3	4	5	6	7	8	9
1	54.000	2.000	0.000	17.000	11.000	3.000	2.000	2.000	0.000
2	2.000	31.000	0.000	1.000	0.000	0.000	37.000	1.000	0.000
3	0.000	0.000	7.000	3.000	1.000	1.000	2.000	0.000	0.000
4	15.000	0.000	1.000	83.000	2.000	2.000	6.000	1.000	0.000
5	5.000	0.000	0.000	7.000	16.000	1.000	10.000	0.000	0.000
6	6.000	1.000	0.000	2.000	1.000	25.000	9.000	0.000	0.000
7	3.000	13.000	6.000	2.000	4.000	1.000	124.000	0.000	0.000
8	0.000	1.000	0.000	1.000	0.000	0.000	0.000	0.000	1.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

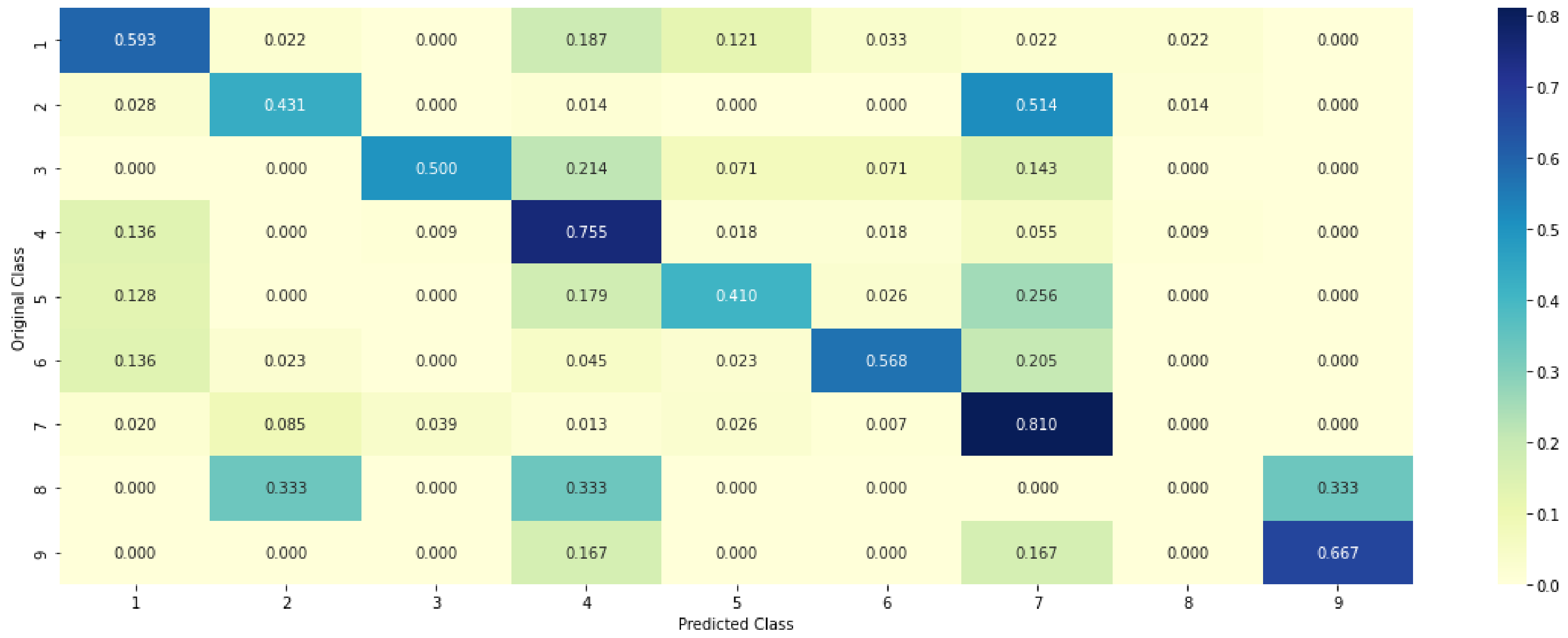
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.635	0.042	0.000	0.145	0.314	0.091	0.010	0.500	0.000
2	0.024	0.646	0.000	0.009	0.000	0.000	0.194	0.250	0.000
3	0.000	0.000	0.500	0.026	0.029	0.030	0.010	0.000	0.000
4	0.176	0.000	0.071	0.709	0.057	0.061	0.031	0.250	0.000
5	0.059	0.000	0.000	0.060	0.457	0.030	0.052	0.000	0.000
6	0.071	0.021	0.000	0.017	0.029	0.758	0.047	0.000	0.000
7	0.035	0.271	0.429	0.017	0.114	0.030	0.649	0.000	0.000
8	0.000	0.021	0.000	0.009	0.000	0.000	0.000	0.000	0.200
9	0.000	0.000	0.000	0.009	0.000	0.000	0.005	0.000	0.800

Predicted Class

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



Feature Importance

In [168]:
```
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']))
```

Correctly Classified Point

In [169]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log',
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_
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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[tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0255 0.0408 0.0086 0.0263 0.0135 0.0063 0.8679 0.0085 0.0027
]]
Actual Class : 7
--------------------------------------------------

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

37 Text feature [constitutive] present in test data point [True]
38 Text feature [activated] present in test data point [True]
40 Text feature [constitutively] present in test data point [True]
76 Text feature [murine] present in test data point [True]
88 Text feature [activating] present in test data point [True]
108 Text feature [transforms] present in test data point [True]
117 Text feature [nude] present in test data point [True]
119 Text feature [transforming] present in test data point [True]
143 Text feature [infect] present in test data point [True]
151 Text feature [activation] present in test data point [True]
173 Text feature [ligand] present in test data point [True]
182 Text feature [tk] present in test data point [True]
185 Text feature [oncogene] present in test data point [True]
203 Text feature [transform] present in test data point [True]
221 Text feature [receptors] present in test data point [True]
223 Text feature [agar] present in test data point [True]
233 Text feature [grew] present in test data point [True]
253 Text feature [extracellular] present in test data point [True]
259 Text feature [oncogenes] present in test data point [True]
300 Text feature [phospho] present in test data point [True]
324 Text feature [downstream] present in test data point [True]
373 Text feature [tyrosine] present in test data point [True]
408 Text feature [inhibited] present in test data point [True]
413 Text feature [soft] present in test data point [True]
426 Text feature [expressing] present in test data point [True]
451 Text feature [autophosphorylation] present in test data point [True]
Out of the top  500  features  26 are present in query point
```

Incorrectly Classified Point

In [170]:
```
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_
print("Actual Class :", test_y[test_point_index])
indices  =  np.argsort(-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[tes
```

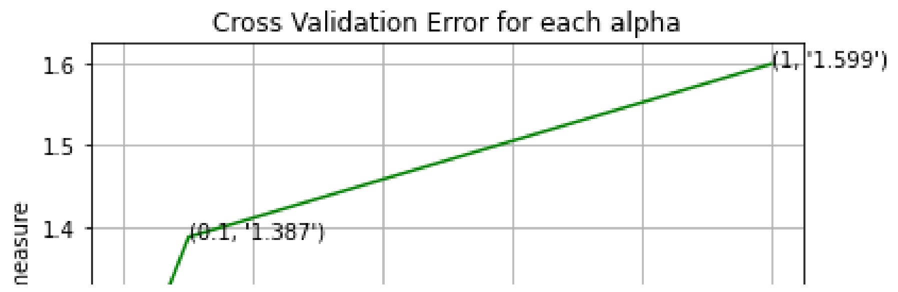
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0668 0.1643 0.0108 0.0846 0.0346 0.0353 0.5893 0.0078 0.0064
]]
Actual Class : 2
--------------------------------------------------
88 Text feature [activating] present in test data point [True]
120 Text feature [technology] present in test data point [True]
151 Text feature [activation] present in test data point [True]
227 Text feature [rearranges] present in test data point [True]
304 Text feature [cysteine] present in test data point [True]
324 Text feature [downstream] present in test data point [True]
373 Text feature [tyrosine] present in test data point [True]
470 Text feature [concentrations] present in test data point [True]
Out of the top 500 features 8 are present in query point
```

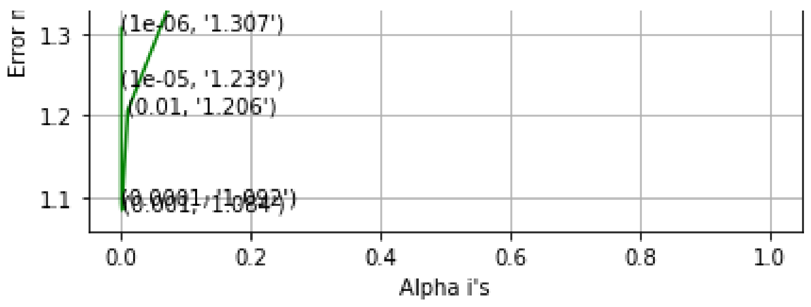
Without Class balancing

Hyper paramter tuning

In [171]:
```
#  read  more  about  SGDClassifier()  at  http://scikit-learn.org/stable/modules/generated/sklearn.li
# ------------------------------
#  default  parameters
#  SGDClassifier(loss='hinge',  penalty='l2',  alpha=0.0001,  l1_ratio=0.15,  fit_intercept=True,  max
#  shuffle=True,  verbose=0,  epsilon=0.1,  n_jobs=1,  random_state=None,  learning_rate='optimal',  et
#  class_weight=None,  warm_start=False,  average=False,  n_iter=None)

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3072565479336107
for alpha = 1e-05
Log Loss : 1.2393529135806205
for alpha = 0.0001
Log Loss : 1.091876970590176
for alpha = 0.001
Log Loss : 1.0838403845656113
for alpha = 0.01
Log Loss : 1.2056153470141253
for alpha = 0.1
Log Loss : 1.3871692274467302
for alpha = 1
Log Loss : 1.5994303451551597
```



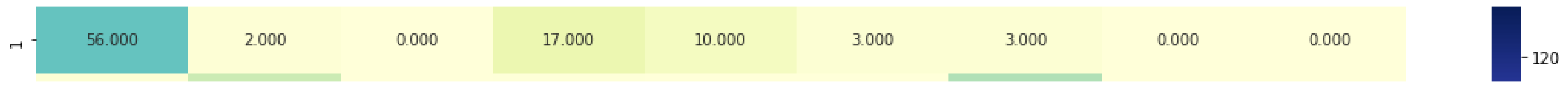


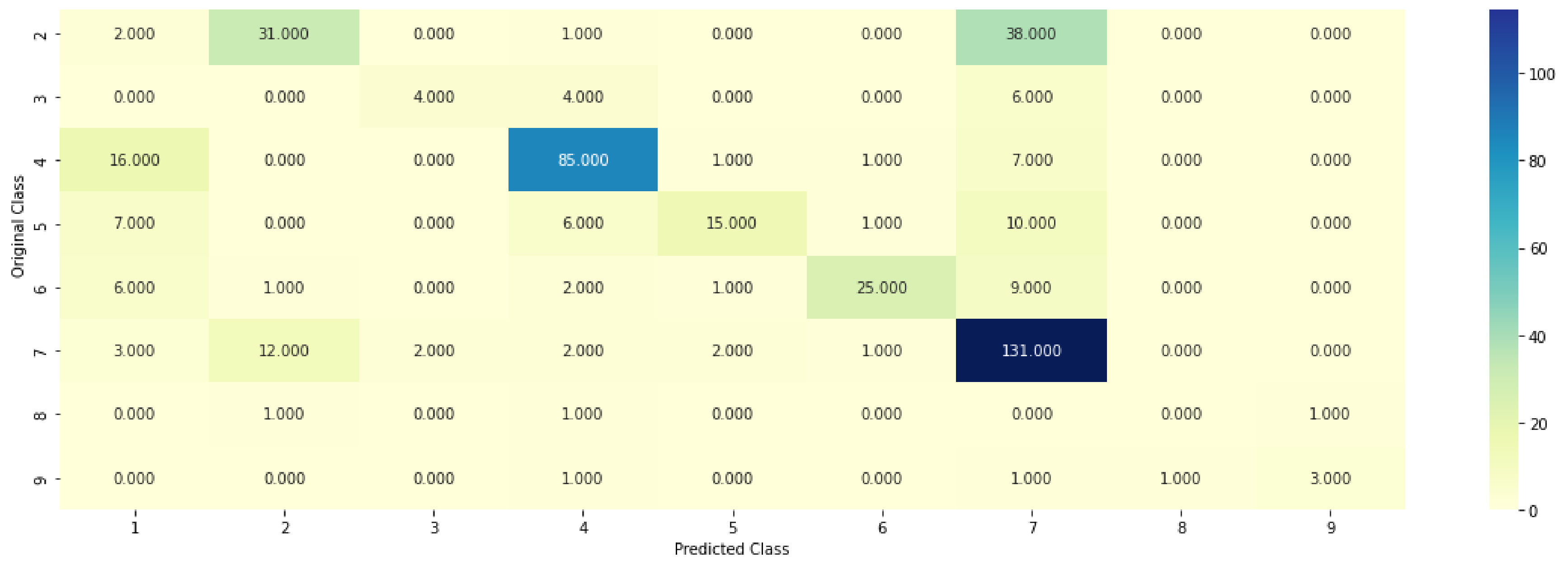
```
For values of best alpha =  0.001 The train log loss is: 0.5192740524355157
For values of best alpha = 0.001 The cross validation log loss is: 1.0838403845656113
For values of best alpha = 0.001 The test log loss is: 1.0636772347874175
```

Testing model with best hyper parameters

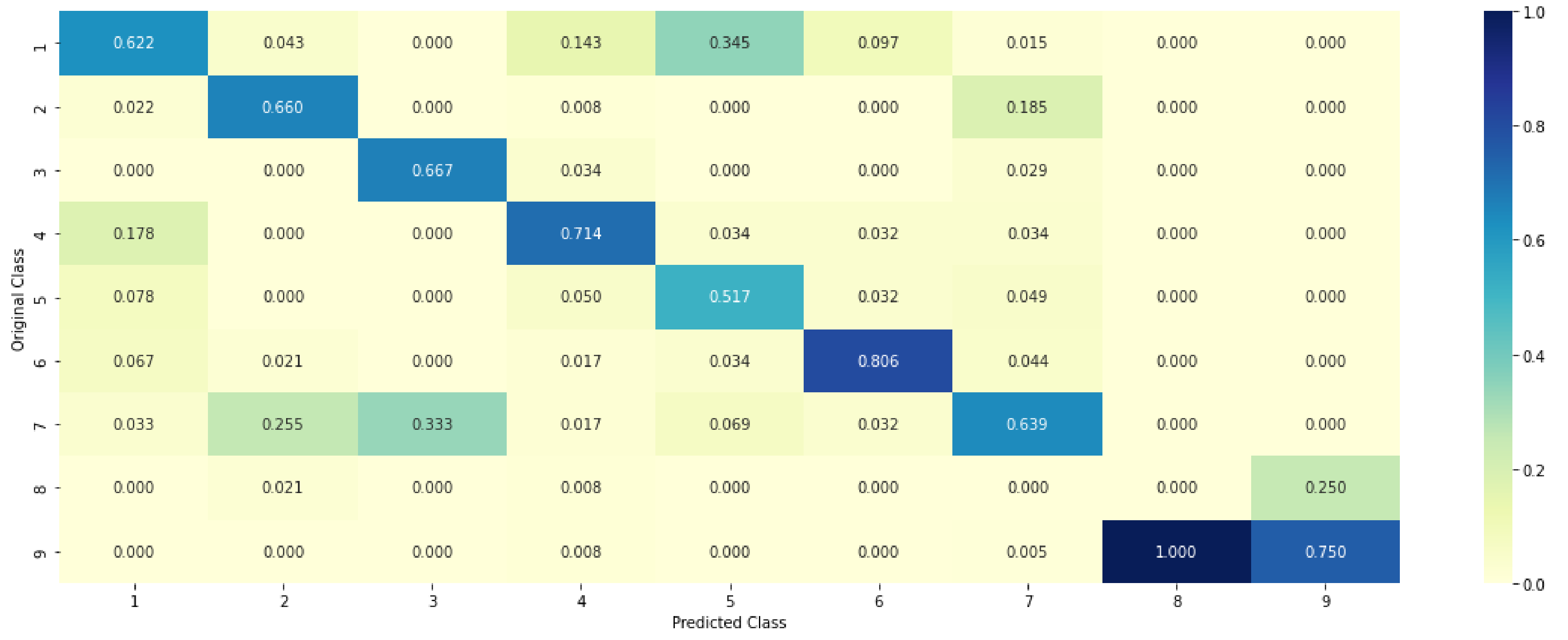
In [172]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.li
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', et
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.0838403845656113
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```

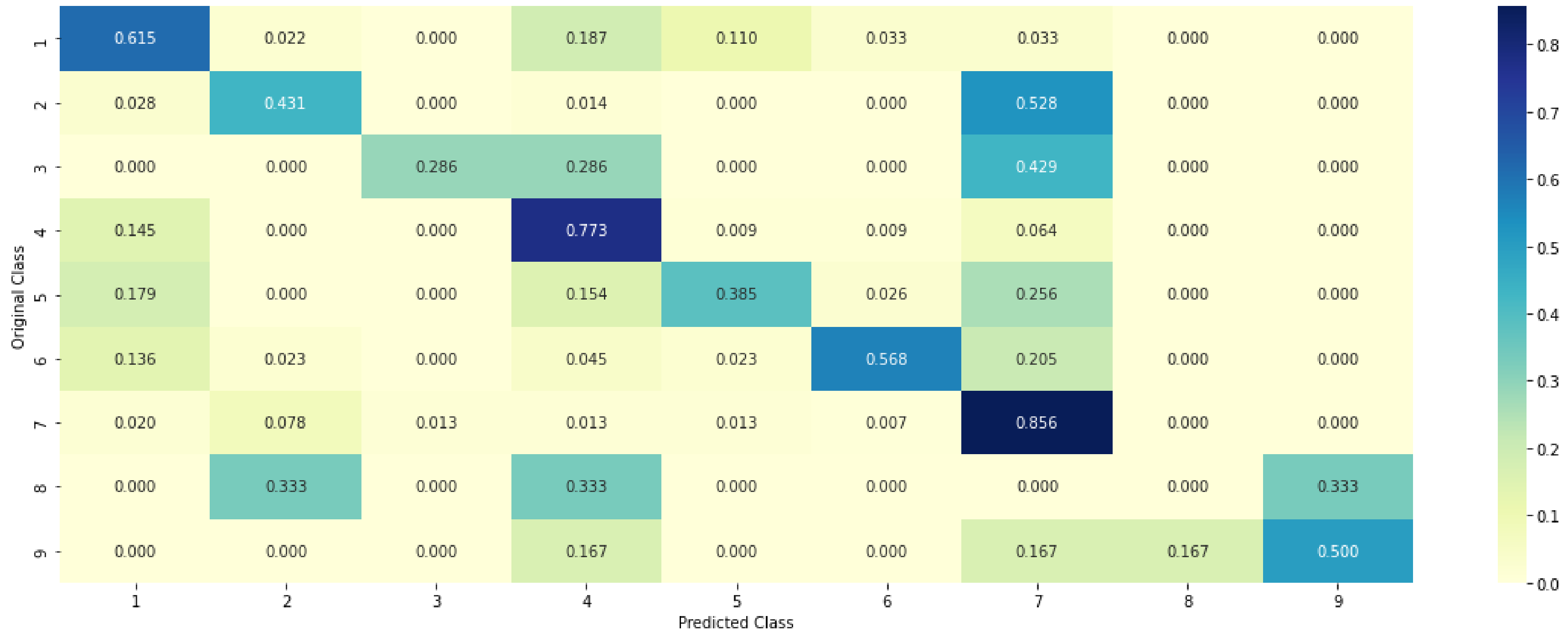




-------------------- Precision matrix (Columm Sum=1) --------------------



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



Feature Importance, Correctly Classified point

In [173]:
```
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_
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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[tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.590e-02 3.910e-02 7.300e-03 2.710e-02 1.220e-02 5.700e-03 8.
739e-01
  7.900e-03 8.000e-04]]
```

```
Actual Class : 7
--------------------------------------------------

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

120 Text feature [constitutive] present in test data point [True]
125 Text feature [constitutively] present in test data point [True]
128 Text feature [activated] present in test data point [True]
137 Text feature [activating] present in test data point [True]
156 Text feature [transforming] present in test data point [True]
199 Text feature [murine] present in test data point [True]
225 Text feature [nude] present in test data point [True]
271 Text feature [agar] present in test data point [True]
283 Text feature [infect] present in test data point [True]
301 Text feature [activation] present in test data point [True]
308 Text feature [extracellular] present in test data point [True]
322 Text feature [transforms] present in test data point [True]
336 Text feature [transform] present in test data point [True]
367 Text feature [tk] present in test data point [True]
385 Text feature [oncogene] present in test data point [True]
393 Text feature [grew] present in test data point [True]
401 Text feature [soft] present in test data point [True]
426 Text feature [phospho] present in test data point [True]
444 Text feature [ligand] present in test data point [True]
496 Text feature [oncogenes] present in test data point [True]
Out of the top  500  features  20 are present in query point
```

Feature Importance, Inorrectly Classified point

```
In [174]: 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_
          print("Actual Class :", test_y[test_point_index])
          indices  =  np.argsort(-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[tes
```

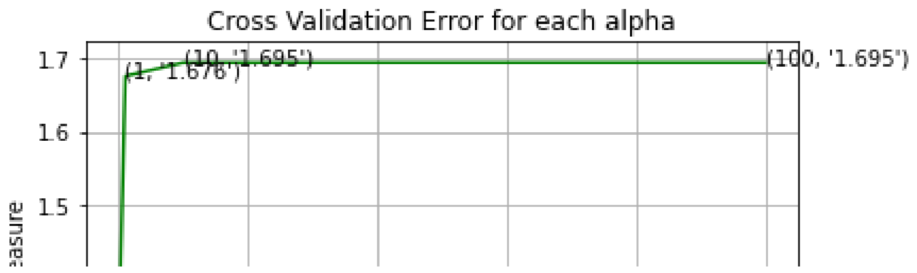
```
Predicted Class : 7
Predicted Class Probabilities: [[0.0694 0.1741 0.0115 0.0857 0.0376 0.038  0.5711 0.0083 0.0044
]]
Actual Class : 2
--------------------------------------------------
137 Text feature [activating] present in test data point [True]
198 Text feature [technology] present in test data point [True]
301 Text feature [activation] present in test data point [True]
306 Text feature [cysteine] present in test data point [True]
349 Text feature [rearranges] present in test data point [True]
Out of the top 500 features 5 are present in query point
```

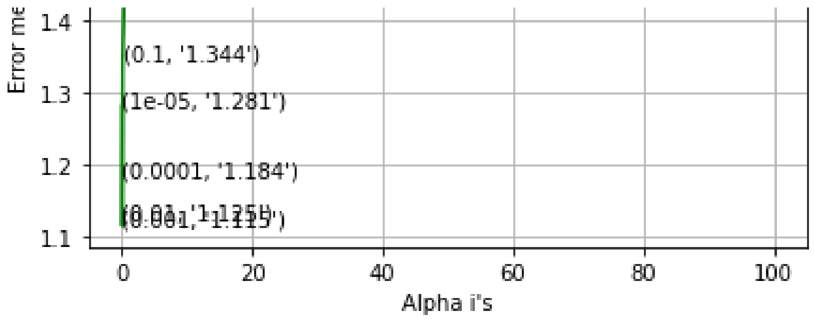
Linear Support Vector Machines

Hyper paramter tuning

```
In [175]: # read more about support vector machines with linear kernals here http://scikit-learn.org/stabl

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

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



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

          alpha = [10 ** x for x in range(-5, 3)]
          cv_log_error_array = []
          for i in alpha:
              print("for C =", i)
          #     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
              clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_st
              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'
          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_trai
          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_
          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,
```

```
for C = 1e-05
Log Loss : 1.280965399703902
for C = 0.0001
Log Loss : 1.1837091097927626
for C = 0.001
Log Loss : 1.1152053959572574
for C = 0.01
Log Loss : 1.12467494493669
for C = 0.1
Log Loss : 1.3444560186800179
for C = 1
Log Loss : 1.6764658665611232
for C = 10
Log Loss : 1.6947916838061143
for C = 100
Log Loss : 1.6947917707178537
```





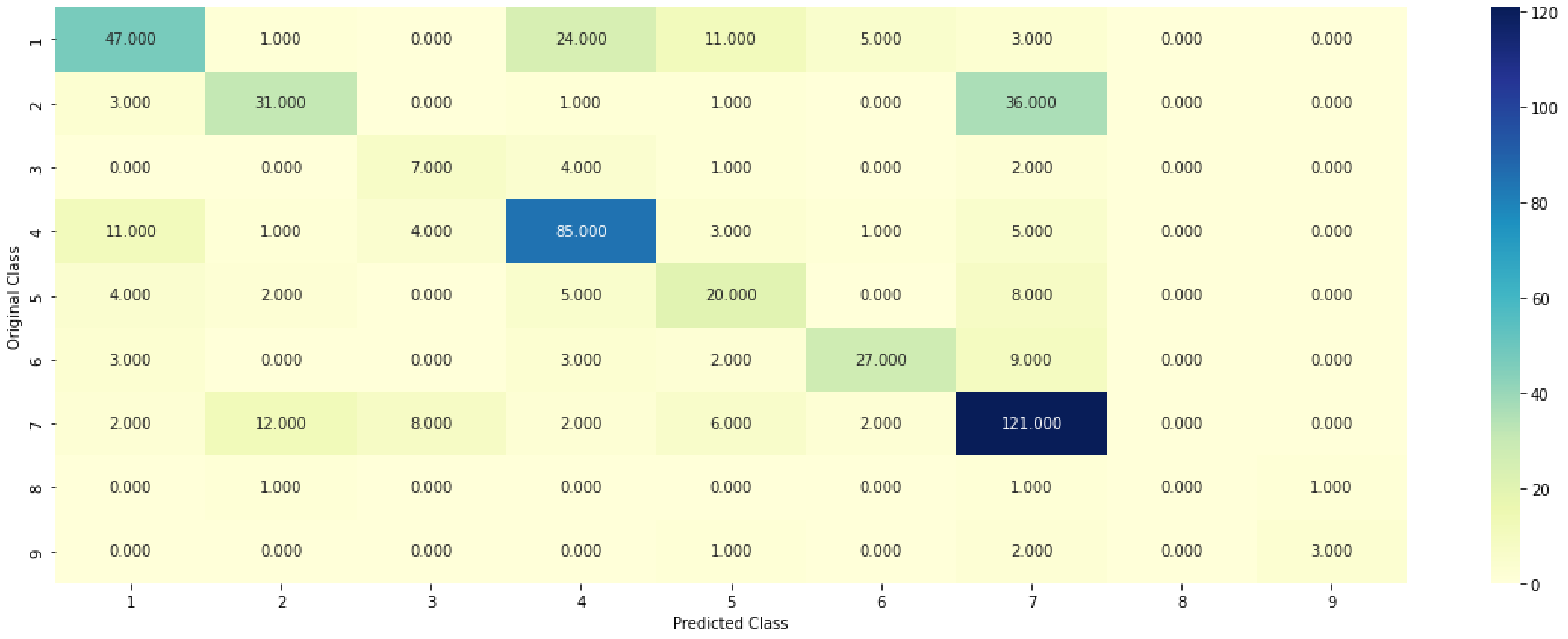
```
For values of best alpha =  0.001 The train log loss is: 0.5383562835418495
For values of best alpha = 0.001 The cross validation log loss is: 1.1152053959572574
For values of best alpha = 0.001 The test log loss is: 1.1330562883278201
```

Testing model with best hyper parameters

In [176]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stabl

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

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


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

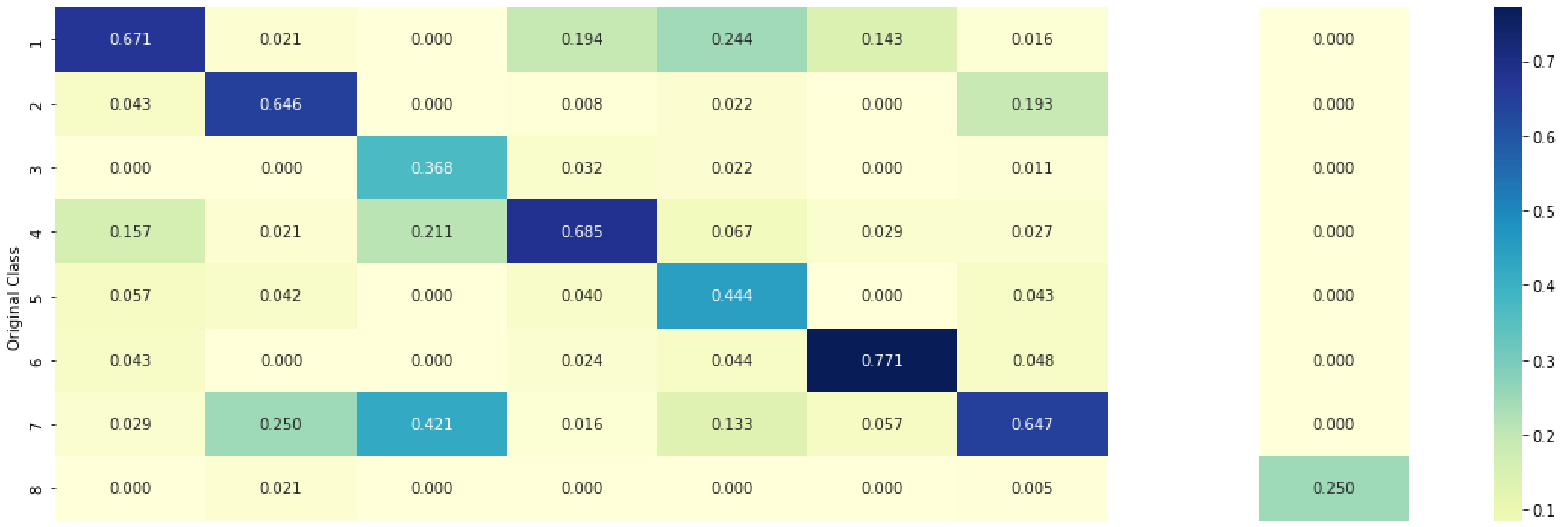
```
Log loss : 1.1152053959572574
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------

/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2633006593.py:22:  RuntimeWarni
```

```
ng: invalid value encountered in true_divide
  B =(C/C.sum(axis=0))
```

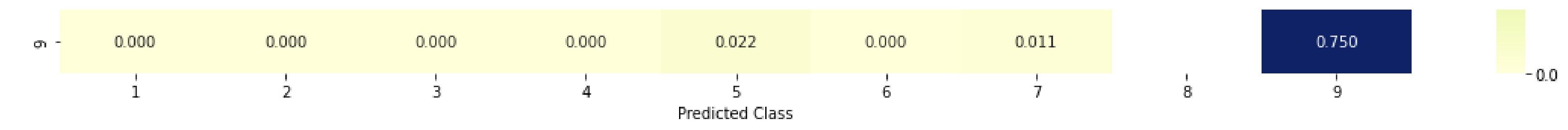


	1	2	3	4	5	6	7	8	9
1	47.000	1.000	0.000	24.000	11.000	5.000	3.000	0.000	0.000
2	3.000	31.000	0.000	1.000	1.000	0.000	36.000	0.000	0.000
3	0.000	0.000	7.000	4.000	1.000	0.000	2.000	0.000	0.000
4	11.000	1.000	4.000	85.000	3.000	1.000	5.000	0.000	0.000
5	4.000	2.000	0.000	5.000	20.000	0.000	8.000	0.000	0.000
6	3.000	0.000	0.000	3.000	2.000	27.000	9.000	0.000	0.000
7	2.000	12.000	8.000	2.000	6.000	2.000	121.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000
9	0.000	0.000	0.000	0.000	1.000	0.000	2.000	0.000	3.000

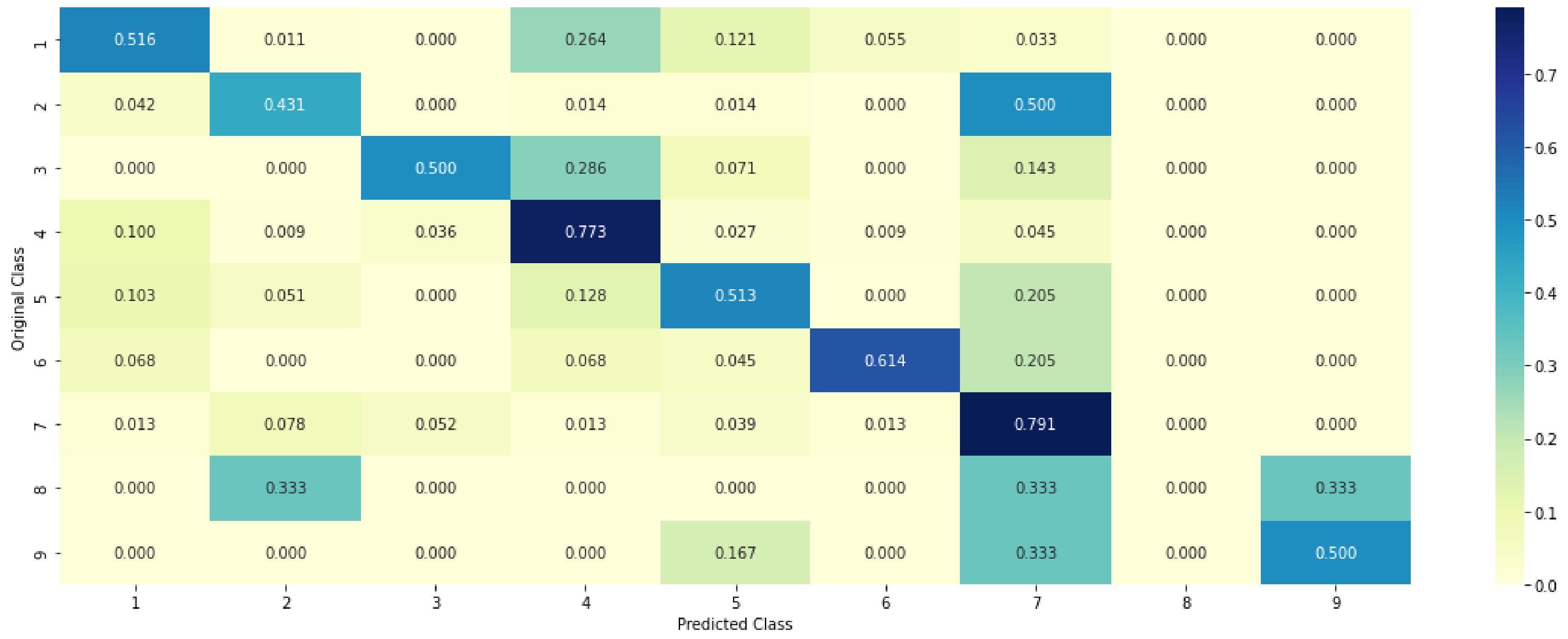
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



1	0.671	0.021	0.000	0.194	0.244	0.143	0.016		0.000
2	0.043	0.646	0.000	0.008	0.022	0.000	0.193		0.000
3	0.000	0.000	0.368	0.032	0.022	0.000	0.011		0.000
4	0.157	0.021	0.211	0.685	0.067	0.029	0.027		0.000
5	0.057	0.042	0.000	0.040	0.444	0.000	0.043		0.000
6	0.043	0.000	0.000	0.024	0.044	0.771	0.048		0.000
7	0.029	0.250	0.421	0.016	0.133	0.057	0.647		0.000
8	0.000	0.021	0.000	0.000	0.000	0.000	0.005		0.250



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



	1	2	3	4	5	6	7	8	9
1	0.516	0.011	0.000	0.264	0.121	0.055	0.033	0.000	0.000
2	0.042	0.431	0.000	0.014	0.014	0.000	0.500	0.000	0.000
3	0.000	0.000	0.500	0.286	0.071	0.000	0.143	0.000	0.000
4	0.100	0.009	0.036	0.773	0.027	0.009	0.045	0.000	0.000
5	0.103	0.051	0.000	0.128	0.513	0.000	0.205	0.000	0.000
6	0.068	0.000	0.000	0.068	0.045	0.614	0.205	0.000	0.000
7	0.013	0.078	0.052	0.013	0.039	0.013	0.791	0.000	0.000
8	0.000	0.333	0.000	0.000	0.000	0.000	0.333	0.000	0.333
9	0.000	0.000	0.000	0.000	0.167	0.000	0.333	0.000	0.500

Feature Importance

For Correctly classified point

```
In [177]: 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_
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-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[tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0698 0.0457 0.0109 0.0875 0.0331 0.0126 0.7274 0.006  0.0069
]]
Actual Class : 7
--------------------------------------------------

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

Out of the top  500  features  0 are present in query point
```

For Incorrectly classified point

In [178]:
```
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_
print("Actual Class :", test_y[test_point_index])
indices  =  np.argsort(-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[tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0646 0.2415 0.0141 0.1006 0.042  0.0352 0.4852 0.0074 0.0093
]]
Actual Class : 2
--------------------------------------------------
242 Text feature [rearranges] present in test data point [True]
456 Text feature [technology] present in test data point [True]
Out of the top 500 features 2 are present in query point
```

Random Forest Classifier

Hyper paramter tuning (With One hot Encoding)

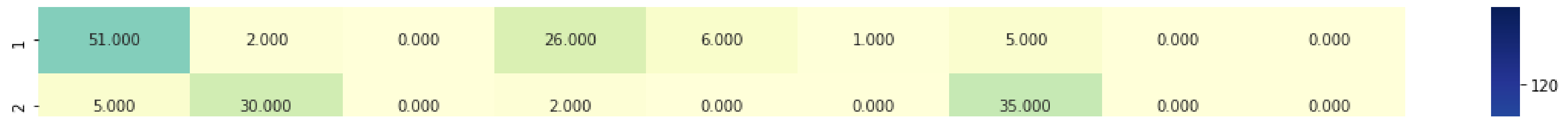
In [179]:
```
# ----------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, mi
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose
# class_weight=None)

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

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

# ----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-for
# ----------------------------------


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state
        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[
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=
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log los
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_l
```

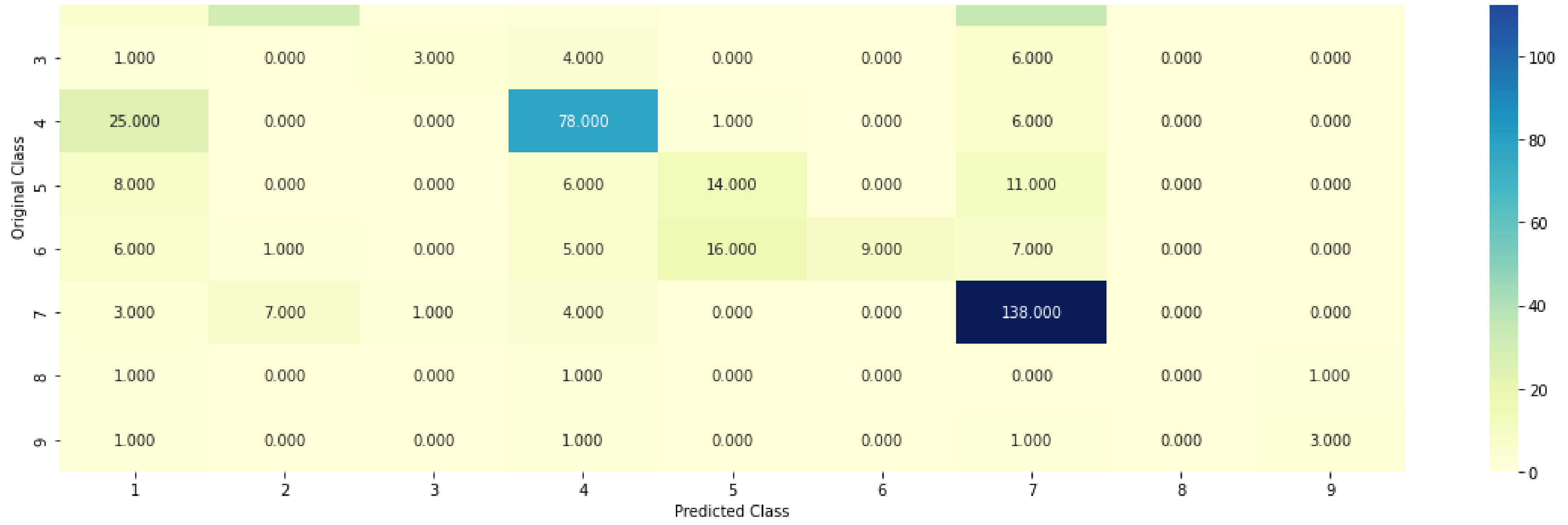
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1654734153936381
for n_estimators = 100 and max depth =  10
Log Loss : 1.1282610750340203
for n_estimators = 200 and max depth =  5
Log Loss : 1.1508243499342474
for n_estimators = 200 and max depth =  10
Log Loss : 1.1191206708490327
for n_estimators = 500 and max depth =  5
Log Loss : 1.1430742921379784
for n_estimators = 500 and max depth =  10
Log Loss : 1.114432378028928
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1386191565816257
for n_estimators = 1000 and max depth =  10
Log Loss : 1.114939266830843
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1379598549698737
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1136839186957035
For values of best estimator =  2000 The train log loss is: 0.6612054781710692
For values of best estimator = 2000 The cross validation log loss is: 1.1136839186957037
For values of best estimator = 2000 The test log loss is: 1.1377563145264038
```

Testing model with best hyper parameters (One Hot Encoding)

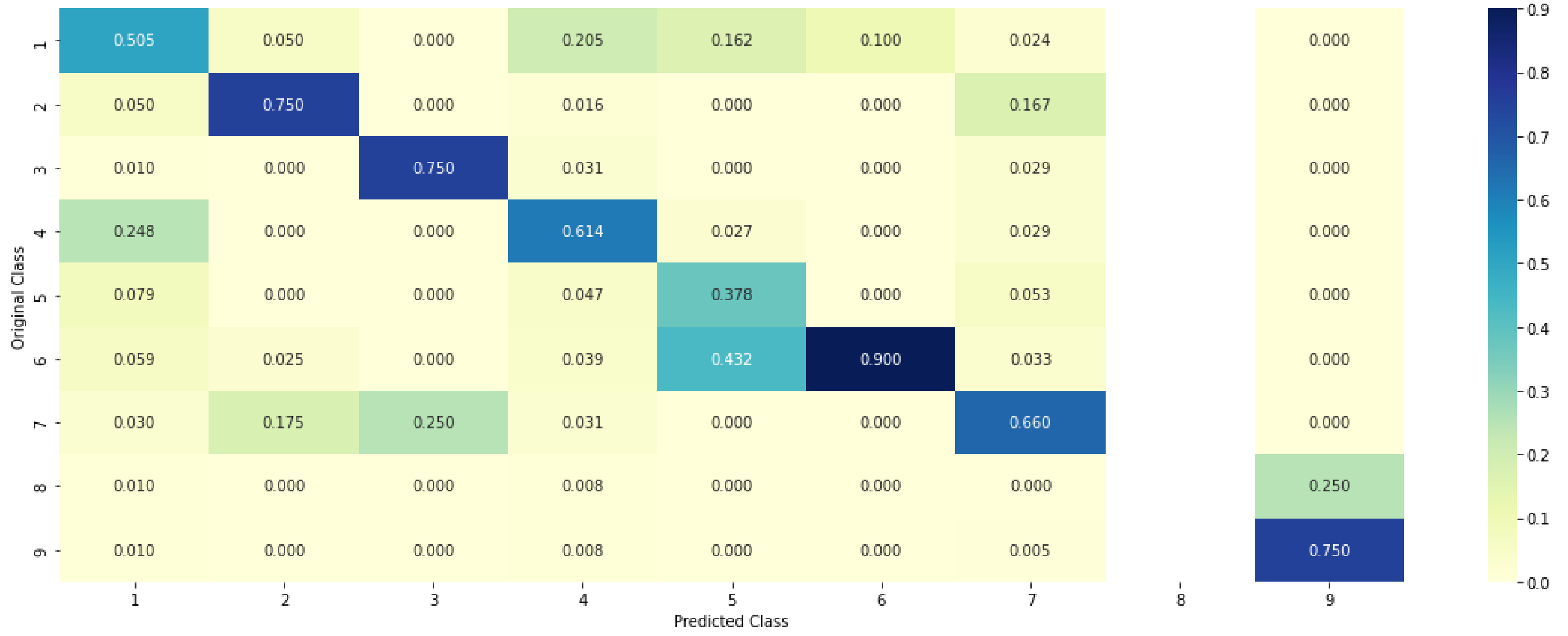
In [180]:
```python
# ----------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, mi
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose
# class_weight=None)

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

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

# ----------------------------
#  video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-for
# ----------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1136839186957035
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------

/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2633006593.py:22: RuntimeWarni
ng: invalid value encountered in true_divide
  B =(C/C.sum(axis=0))
```

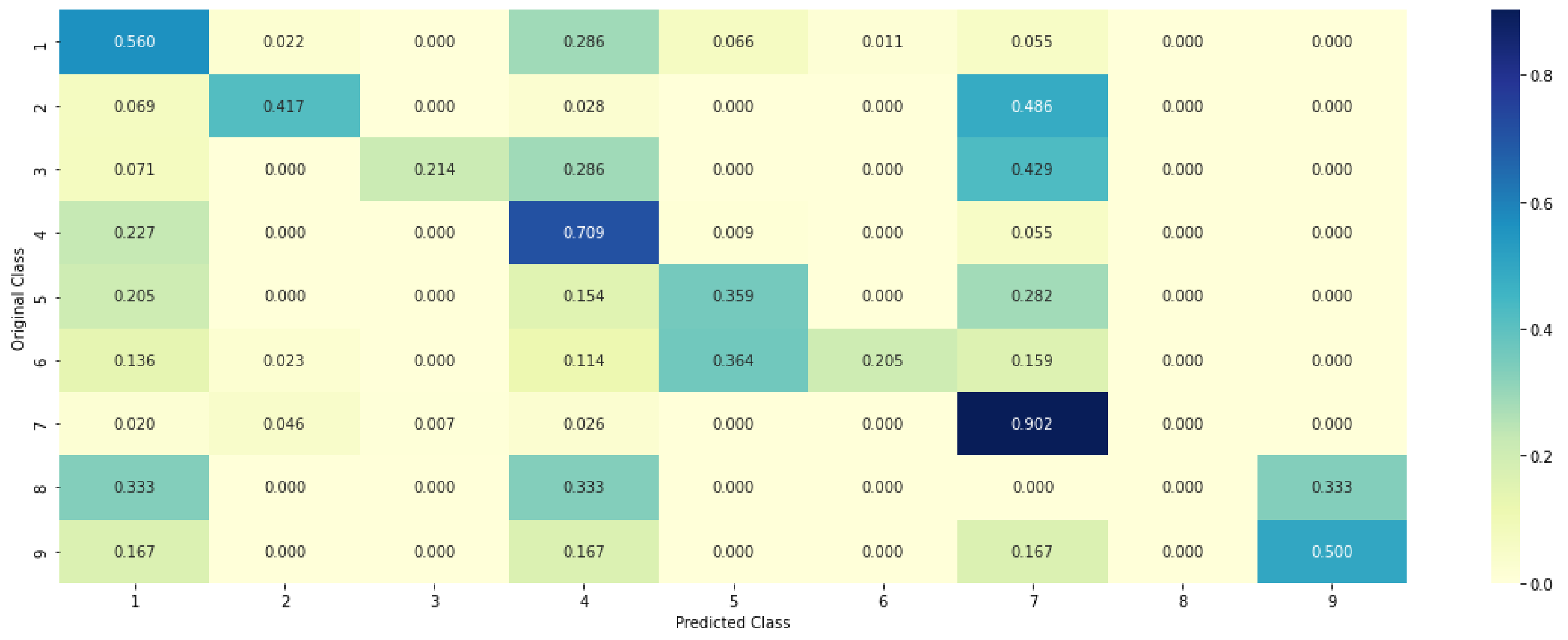




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



	1	2	3	4	5	6	7	8	9
1	0.560	0.022	0.000	0.286	0.066	0.011	0.055	0.000	0.000
2	0.069	0.417	0.000	0.028	0.000	0.000	0.486	0.000	0.000
3	0.071	0.000	0.214	0.286	0.000	0.000	0.429	0.000	0.000
4	0.227	0.000	0.000	0.709	0.009	0.000	0.055	0.000	0.000
5	0.205	0.000	0.000	0.154	0.359	0.000	0.282	0.000	0.000
6	0.136	0.023	0.000	0.114	0.364	0.205	0.159	0.000	0.000
7	0.020	0.046	0.007	0.026	0.000	0.000	0.902	0.000	0.000
8	0.333	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.333
9	0.167	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.500

Feature Importance

Correctly Classified point

In [181]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=
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_
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'
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0438 0.1257 0.0182 0.0409 0.0402 0.0322 0.6866 0.007  0.0054
]]
Actual Class : 7
--------------------------------------------------

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/utils
/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names
is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

0 Text feature [kinase] present in test data point [True]
1 Text feature [tyrosine] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [constitutive] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [activated] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
11 Text feature [function] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [erk] present in test data point [True]
15 Text feature [cells] present in test data point [True]

16 Text feature [akt] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
18 Text feature [loss] present in test data point [True]
19 Text feature [treatment] present in test data point [True]
20 Text feature [trials] present in test data point [True]
21 Text feature [variants] present in test data point [True]
27 Text feature [growth] present in test data point [True]
28 Text feature [extracellular] present in test data point [True]
29 Text feature [constitutively] present in test data point [True]
31 Text feature [receptor] present in test data point [True]
32 Text feature [activate] present in test data point [True]
33 Text feature [egfr] present in test data point [True]
34 Text feature [functional] present in test data point [True]
35 Text feature [therapy] present in test data point [True]
```

```
39 Text feature [downstream] present in test data point [True]
40 Text feature [patients] present in test data point [True]
43 Text feature [treated] present in test data point [True]
44 Text feature [phospho] present in test data point [True]
45 Text feature [expression] present in test data point [True]
46 Text feature [therapeutic] present in test data point [True]
47 Text feature [autophosphorylation] present in test data point [True]
48 Text feature [expressing] present in test data point [True]
51 Text feature [variant] present in test data point [True]
53 Text feature [mek] present in test data point [True]
54 Text feature [mapk] present in test data point [True]
59 Text feature [transforming] present in test data point [True]
61 Text feature [proliferation] present in test data point [True]
62 Text feature [clinical] present in test data point [True]
65 Text feature [protein] present in test data point [True]
66 Text feature [cell] present in test data point [True]
69 Text feature [kinases] present in test data point [True]
71 Text feature [ras] present in test data point [True]
73 Text feature [stimulation] present in test data point [True]
74 Text feature [phosphorylated] present in test data point [True]
79 Text feature [inhibition] present in test data point [True]
81 Text feature [oncogene] present in test data point [True]
84 Text feature [kit] present in test data point [True]
85 Text feature [imatinib] present in test data point [True]
86 Text feature [ligand] present in test data point [True]
91 Text feature [factor] present in test data point [True]

93 Text feature [serum] present in test data point [True]
95 Text feature [mutant] present in test data point [True]
97 Text feature [sensitive] present in test data point [True]
99 Text feature [defect] present in test data point [True]
Out of the top  100  features  57 are present in query point
```

Incorrectly Classified point

In [182]:
```
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_
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'
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2611 0.1283 0.0282 0.2514 0.0699 0.0689 0.1718 0.0088 0.0118
]]
Actuall Class : 2
--------------------------------------------------
1 Text feature [tyrosine] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
11 Text feature [function] present in test data point [True]
15 Text feature [cells] present in test data point [True]
17 Text feature [signaling] present in test data point [True]
32 Text feature [activate] present in test data point [True]
34 Text feature [functional] present in test data point [True]
39 Text feature [downstream] present in test data point [True]
46 Text feature [therapeutic] present in test data point [True]
61 Text feature [proliferation] present in test data point [True]
65 Text feature [protein] present in test data point [True]
66 Text feature [cell] present in test data point [True]
71 Text feature [ras] present in test data point [True]
91 Text feature [factor] present in test data point [True]
95 Text feature [mutant] present in test data point [True]
97 Text feature [sensitive] present in test data point [True]
Out of the top 100 features 17 are present in query point
```

Hyper paramter tuning (With Response Coding)

In [183]:
```
# ----------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, mi
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-for
# --------------------------------


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state
        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[
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=
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
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
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(
```

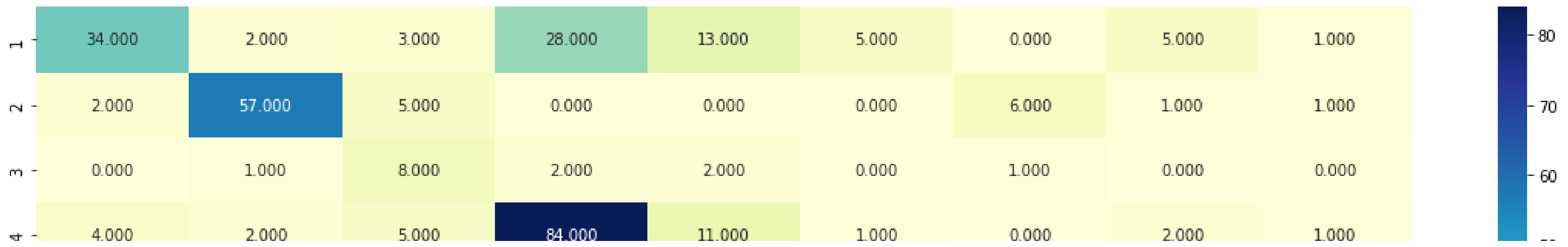
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.083855297699004
for n_estimators = 10 and max depth =  3
Log Loss : 1.630694972689507
for n_estimators = 10 and max depth =  5
Log Loss : 1.4251154142765263
for n_estimators = 10 and max depth =  10
Log Loss : 1.9405418744716412
for n_estimators = 50 and max depth =  2
```

```
Log Loss : 1.6580707929397633
for n_estimators = 50 and max depth = 3
Log Loss : 1.3948367892610267
for n_estimators = 50 and max depth = 5
Log Loss : 1.284595062107084
for n_estimators = 50 and max depth = 10
Log Loss : 1.7311107003409116
for n_estimators = 100 and max depth = 2
Log Loss : 1.5871679703167414
for n_estimators = 100 and max depth = 3
Log Loss : 1.4202749608860727
for n_estimators = 100 and max depth = 5
Log Loss : 1.2913096687736774
for n_estimators = 100 and max depth = 10
Log Loss : 1.7434381609186589
for n_estimators = 200 and max depth = 2
Log Loss : 1.6749510125241613
for n_estimators = 200 and max depth = 3
Log Loss : 1.4972186169788575
for n_estimators = 200 and max depth = 5
Log Loss : 1.3127139552952898
for n_estimators = 200 and max depth = 10
Log Loss : 1.6663364731120516
for n_estimators = 500 and max depth = 2
Log Loss : 1.6887576272582439
for n_estimators = 500 and max depth = 3
Log Loss : 1.5163730243438853
for n_estimators = 500 and max depth = 5
Log Loss : 1.3492850528833853
for n_estimators = 500 and max depth = 10
Log Loss : 1.6383614220040035
for n_estimators = 1000 and max depth = 2
Log Loss : 1.6768756619123433
for n_estimators = 1000 and max depth = 3
Log Loss : 1.5294921520413383
for n_estimators = 1000 and max depth = 5
Log Loss : 1.3246583533786478
for n_estimators = 1000 and max depth = 10
Log Loss : 1.6270407425832567
For values of best alpha =  50 The train log loss is: 0.06796827513008001
For values of best alpha = 50 The cross validation log loss is: 1.2845950524479293
For values of best alpha = 50 The test log loss is: 1.3143842965170753
```

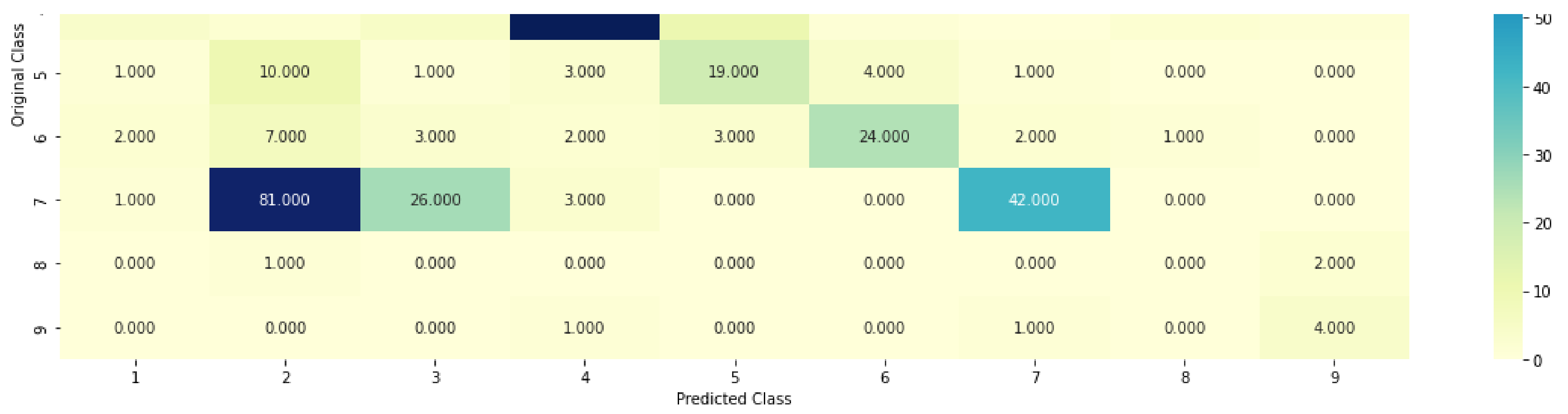
Testing model with best hyper parameters (Response Coding)

In [184]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, mi
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose
# class_weight=None)

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

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

# --------------------------------
#  video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-for
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

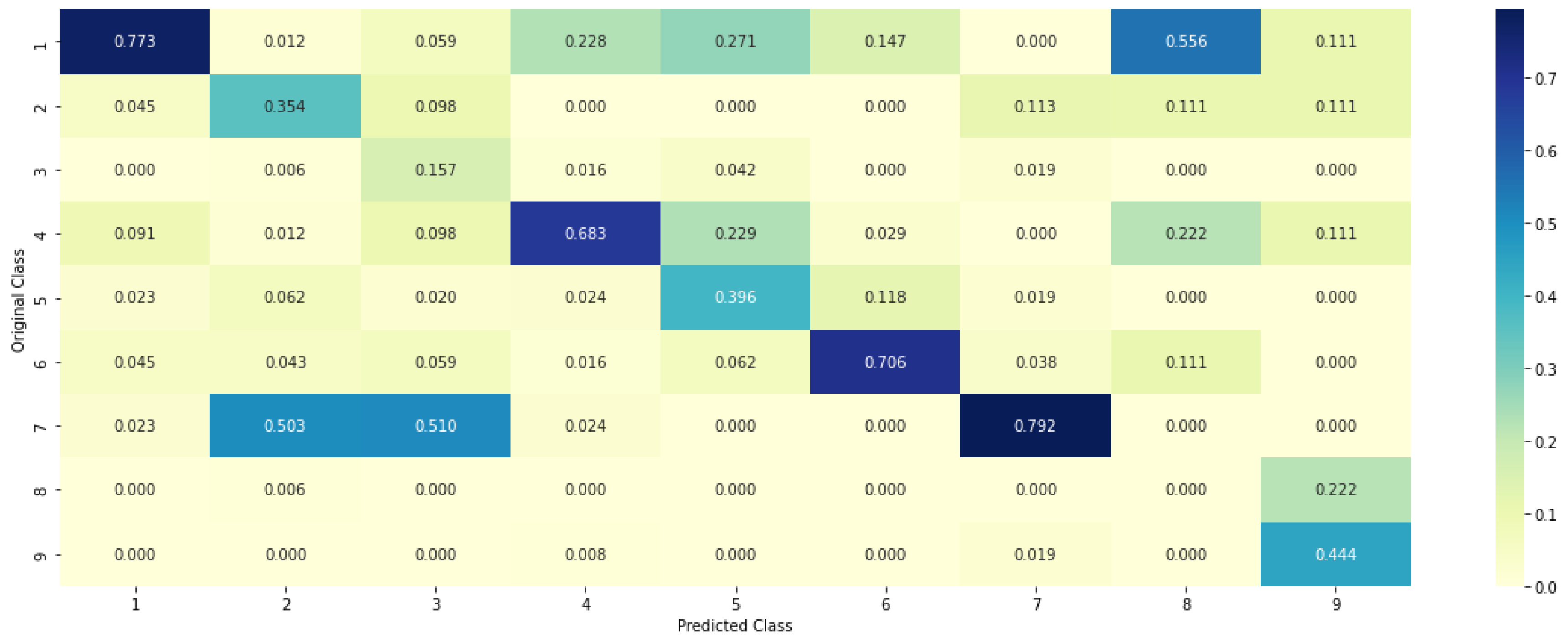
```
Log loss : 1.2845950524479293
Number of mis-classified points : 0.48872180451127817
-------------------- Confusion matrix --------------------
```



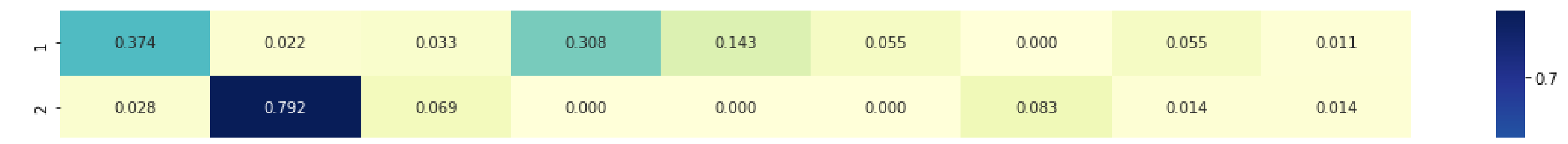
1	34.000	2.000	3.000	28.000	13.000	5.000	0.000	5.000	1.000
2	2.000	57.000	5.000	0.000	0.000	0.000	6.000	1.000	1.000
3	0.000	1.000	8.000	2.000	2.000	0.000	1.000	0.000	0.000
4	4.000	2.000	5.000	84.000	11.000	1.000	0.000	2.000	1.000

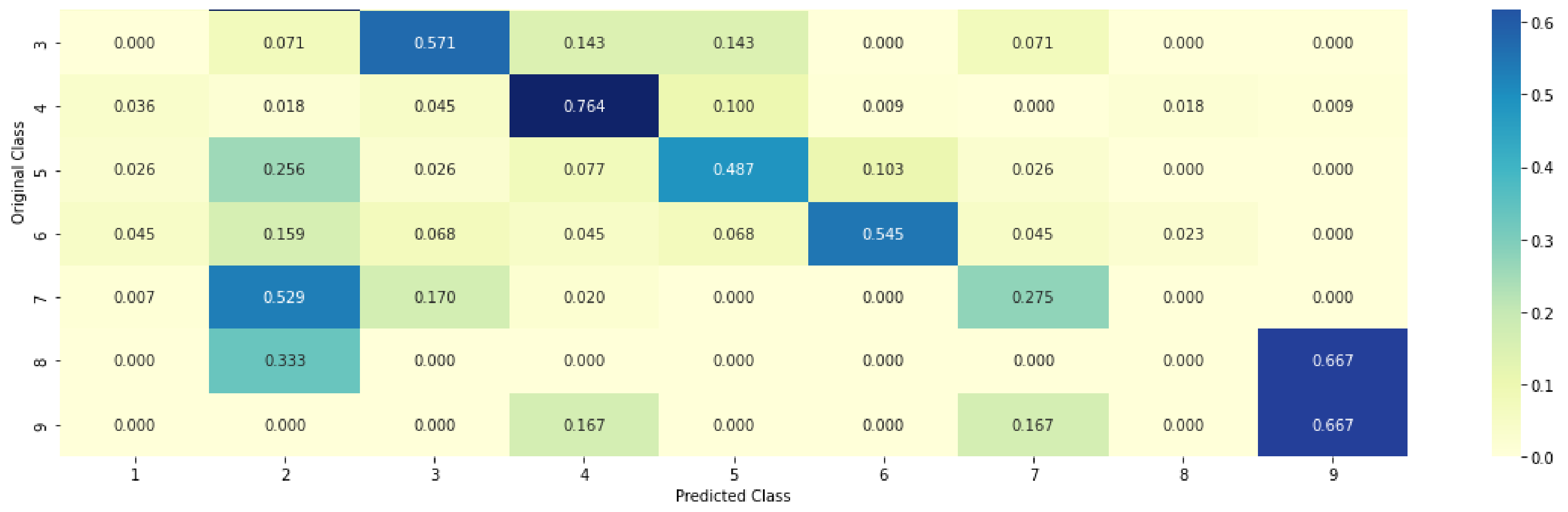


------------------- Precision matrix (Columm Sum=1) --------------------



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





	1	2	3	4	5	6	7	8	9
3	0.000	0.071	0.571	0.143	0.143	0.000	0.071	0.000	0.000
4	0.036	0.018	0.045	0.764	0.100	0.009	0.000	0.018	0.009
5	0.026	0.256	0.026	0.077	0.487	0.103	0.026	0.000	0.000
6	0.045	0.159	0.068	0.045	0.068	0.545	0.045	0.023	0.000
7	0.007	0.529	0.170	0.020	0.000	0.000	0.275	0.000	0.000
8	0.000	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.667
9	0.000	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.667

Feature Importance

Correctly Classified point

In [185]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=
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[tes
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 : 3
Predicted Class Probabilities: [[0.0131 0.1625 0.4701 0.0117 0.0187 0.0161 0.2786 0.0178 0.0115
]]
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
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text 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
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

Incorrectly Classified point

```
In [186]: 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[tes
          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: [[0.0346 0.3898 0.1527 0.0532 0.051  0.0356 0.1271 0.0891 0.0669
]]
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
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text 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
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

Stack the models

testing with hyper parameter tuning

In [187]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.li
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', et
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stabl
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr',

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stabl
# ----------------------------------
# default parameters
#  sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, mi
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose
# class_weight=None)

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

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

# ----------------------------------
#  video  link:  https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-for
# ----------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_stat
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=
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)
```

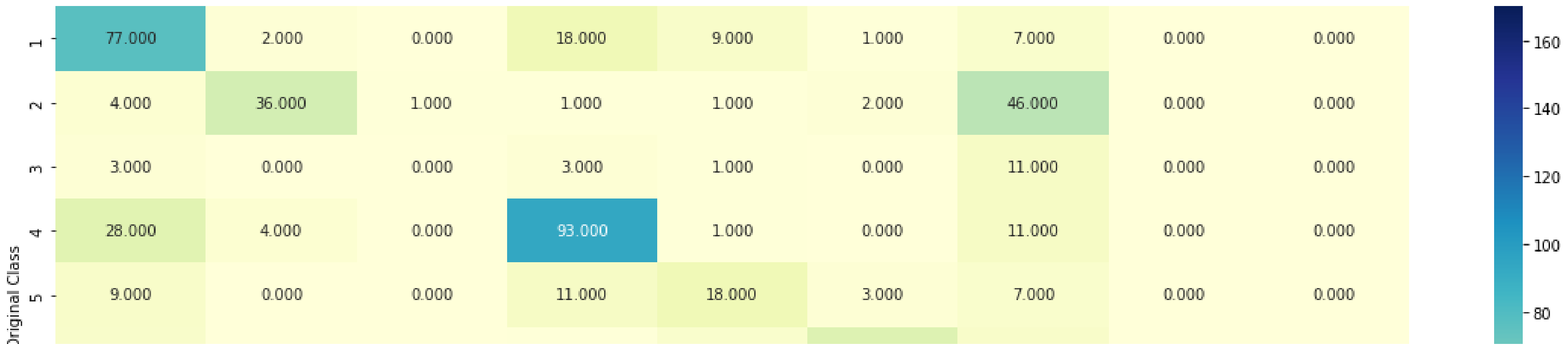
```
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_one
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : 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, us
    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,
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

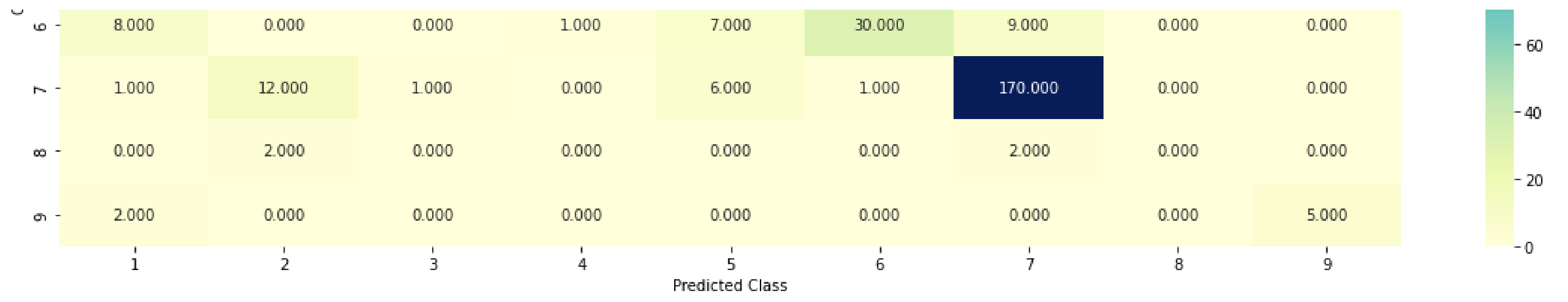
```
Logistic Regression : Log Loss: 1.08
Support vector machines : Log Loss: 1.68
Naive Bayes : Log Loss: 1.26
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.818
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.722
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.322
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.148
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.400
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.717

/Users/Madhavi Pagare/opt/anaconda3/envs/venv-ml-pa1/lib/python3.8/site-packages/sklearn/linea
r_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html  (https://scikit-learn.org/stable
/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
(https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

testing the model with the best hyper parameters

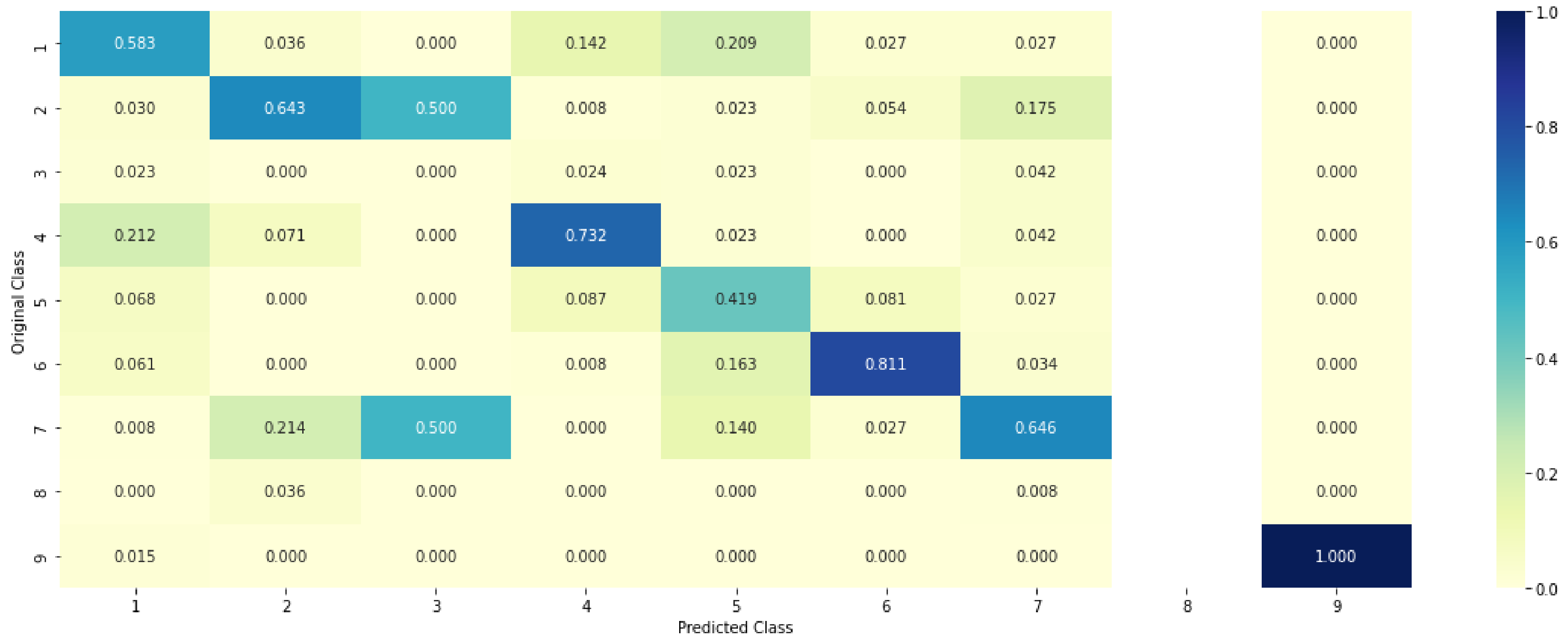
```
In [188]: lr = LogisticRegression(C=0.1)
          sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_pr
          sclf.fit(train_x_onehotCoding, train_y)

          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)- t
          plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the stacking classifier : 0.4962422009183231
Log loss (CV) on the stacking classifier : 1.1483985171083995
Log loss (test) on the stacking classifier : 1.1691890974628807
Number of missclassified point : 0.3548872180451128

/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2633006593.py:22: RuntimeWarni
ng: invalid value encountered in true_divide
  B =(C/C.sum(axis=0))

-------------------- Confusion matrix --------------------
```

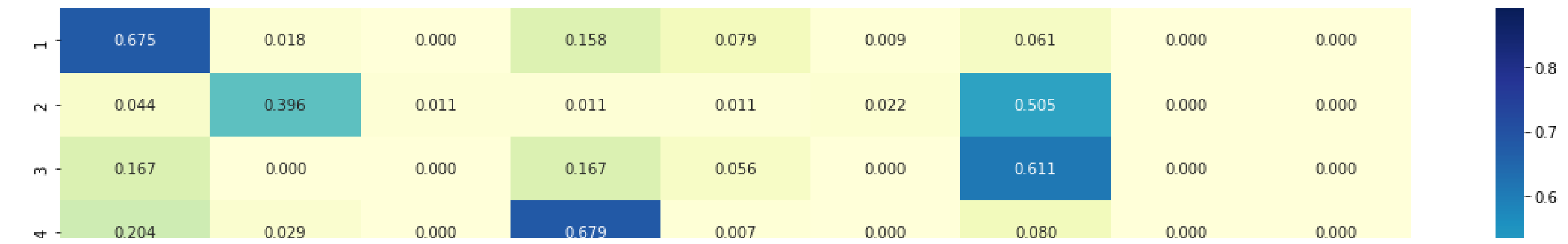


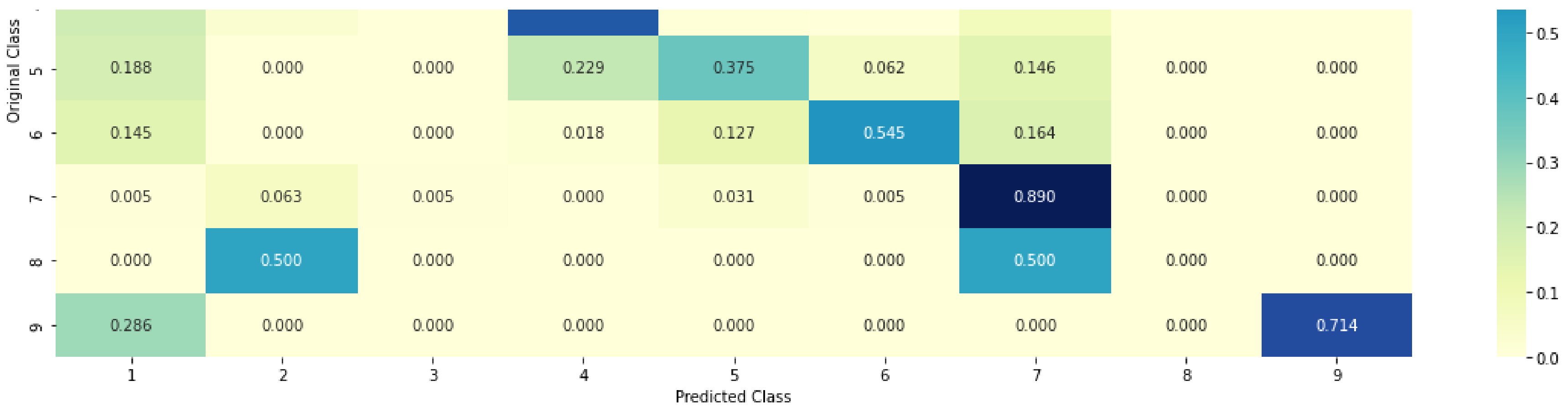


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

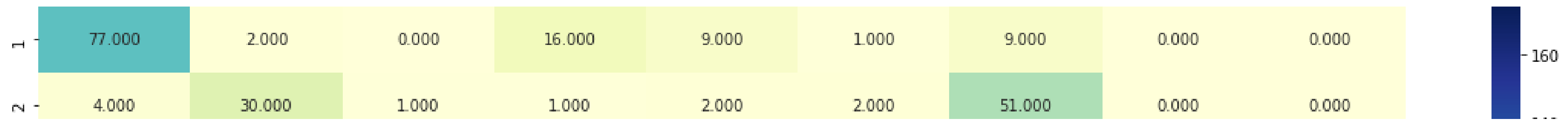


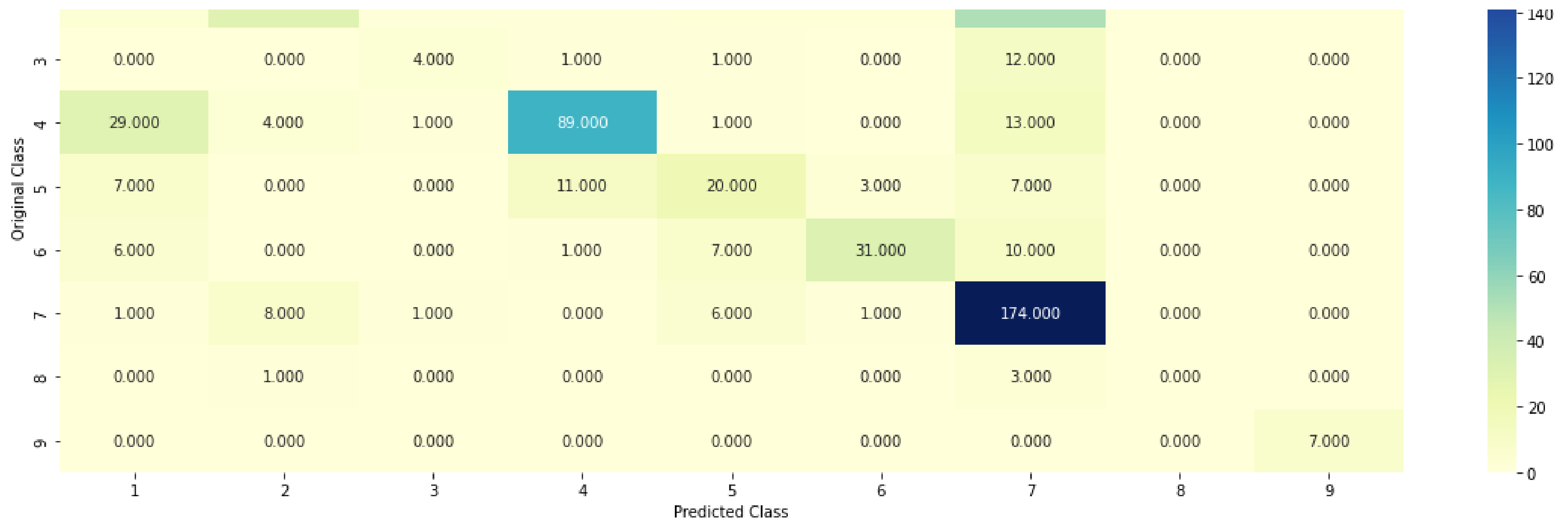


Maximum Voting classifier

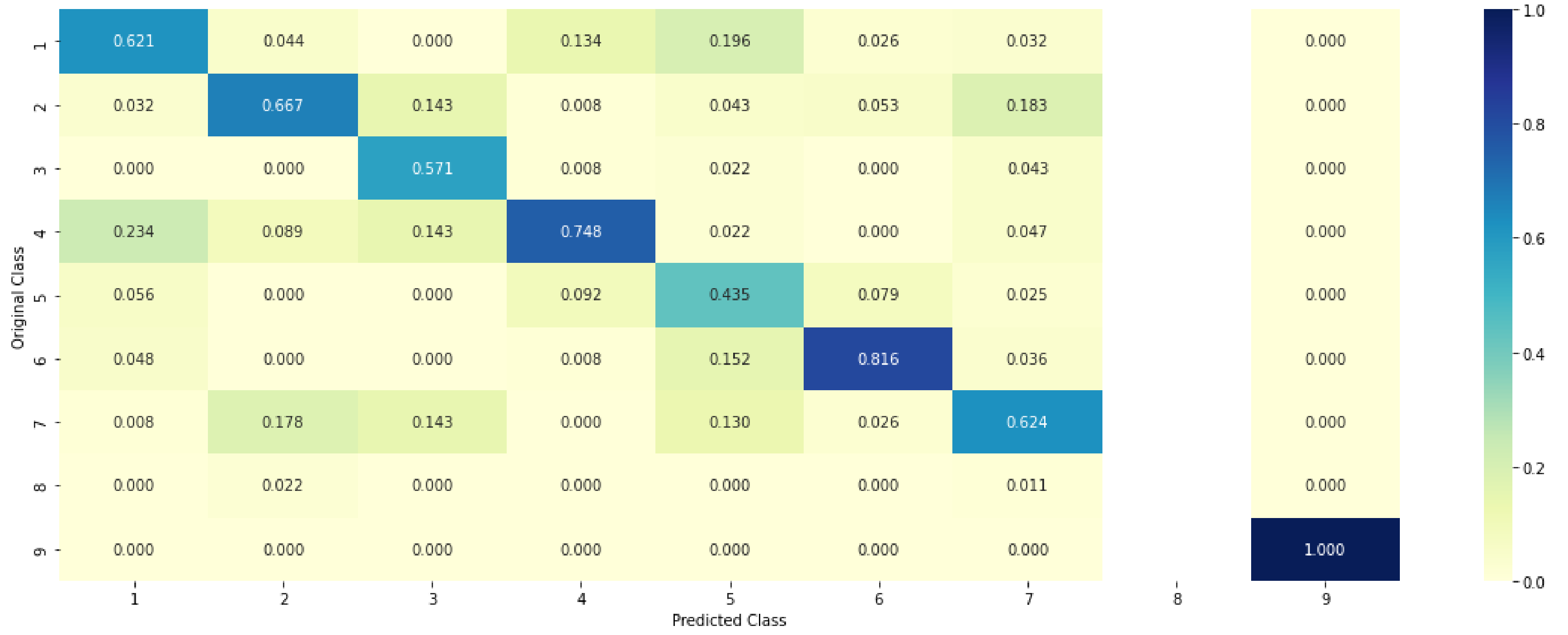
```
In [189]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voti
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCo
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_on
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- t
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.8685002540131741
Log loss (CV) on the VotingClassifier : 1.1769636090161018
Log loss (test) on the VotingClassifier : 1.1951834530521057
Number of missclassified point : 0.35037593984962406

/var/folders/9v/pzmqqtc55hs27lnwt7mk8wz40000gn/T/ipykernel_26637/2633006593.py:22: RuntimeWarni
ng: invalid value encountered in true_divide
  B =(C/C.sum(axis=0))

-------------------- Confusion matrix --------------------
```



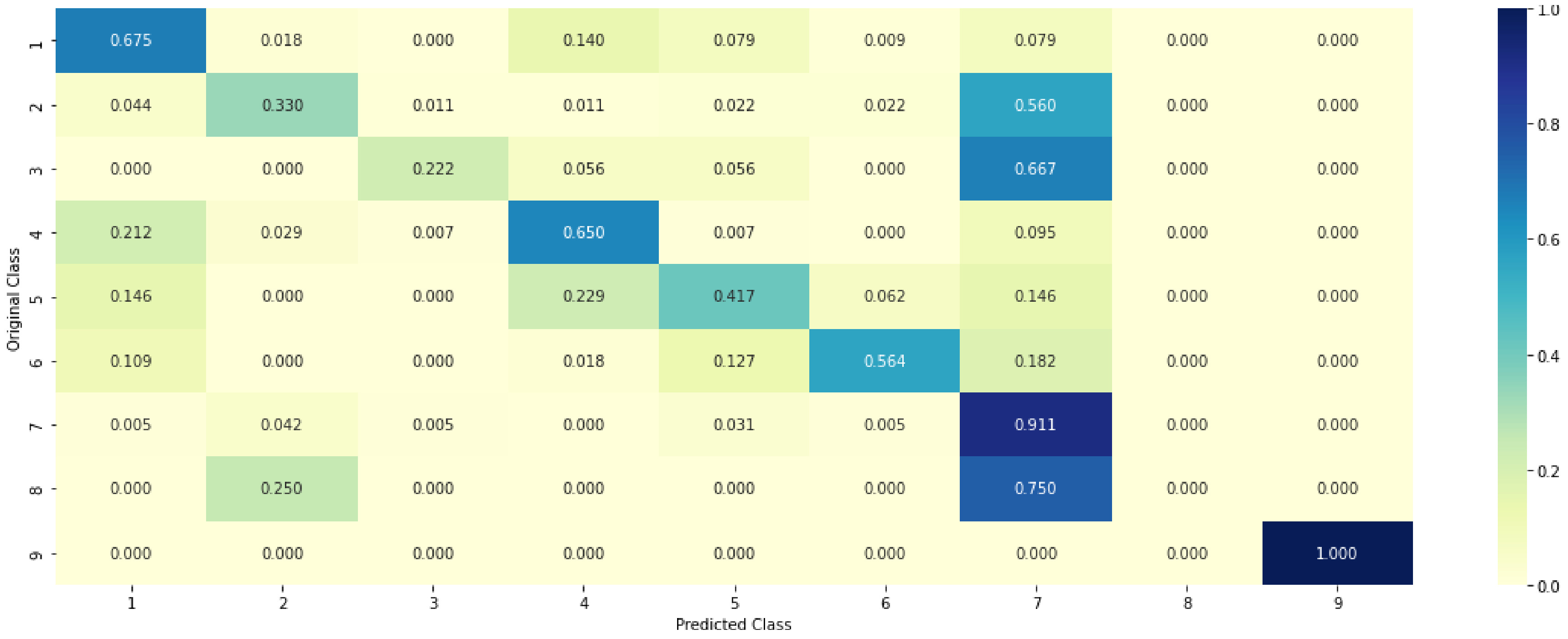


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

```
- -
```



	1	2	3	4	5	6	7	8	9
1	0.675	0.018	0.000	0.140	0.079	0.009	0.079	0.000	0.000
2	0.044	0.330	0.011	0.011	0.022	0.022	0.560	0.000	0.000
3	0.000	0.000	0.222	0.056	0.056	0.000	0.667	0.000	0.000
4	0.212	0.029	0.007	0.650	0.007	0.000	0.095	0.000	0.000
5	0.146	0.000	0.000	0.229	0.417	0.062	0.146	0.000	0.000
6	0.109	0.000	0.000	0.018	0.127	0.564	0.182	0.000	0.000
7	0.005	0.042	0.005	0.000	0.031	0.005	0.911	0.000	0.000
8	0.000	0.250	0.000	0.000	0.000	0.000	0.750	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```