In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import  train_test_split
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/deprecation.py:143: FutureWarning: The sklearn.metrics.c
lassification module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding cl
asses / functions should instead be imported from sklearn.metrics. Anything that cannot be imported from skle
arn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
```

In [2]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input dir
ectory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you
 create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/msk-redefining-cancer-treatment/stage2_test_variants.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/stage2_sample_submission.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/stage_2_private_solution.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/training_text.zip
/kaggle/input/msk-redefining-cancer-treatment/stage1_solution_filtered.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/test_text.zip
/kaggle/input/msk-redefining-cancer-treatment/stage2_test_text.csv.7z
/kaggle/input/msk-redefining-cancer-treatment/training_variants.zip
/kaggle/input/msk-redefining-cancer-treatment/test_variants.zip
```

In [3]:

```
import pandas as pd
data = pd.read_csv('../input/msk-redefining-cancer-treatment/training_variants.zip')
data_text =pd.read_csv("../input/msk-redefining-cancer-treatment/training_text.zip",sep="\|\|",engine="pytho
n",names=["ID","TEXT"],skiprows=1)
```

In [4]:
```
data.head()
```

Out[4]:

	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

In [5]:
```
data_text.head()
```

Out[5]:

	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...

In [6]:
```
print(data.info())
print(data_text.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ID         3321 non-null   int64 
 1   Gene       3321 non-null   object
 2   Variation  3321 non-null   object
 3   Class      3321 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 103.9+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   ID      3321 non-null   int64 
 1   TEXT    3316 non-null   object
dtypes: int64(1), object(1)
memory usage: 52.0+ KB
None
```

preprocessing the text data

In [7]:
```
# 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 [8]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 59.15178399999999 seconds
```

In [9]:
```
result = pd.merge(data,data_text,on='ID',how='left')
result.head()
```

Out[9]:

	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...

filling null values in text data with gene and variation

In [10]:
```
k=result[result.isnull().any(axis=1)]
for row  in k.values:
    result['TEXT'][row[0]] = result['Gene'][row[0]]+ ''+ result['Variation'][row[0]]
```

EDA

In [ ]:
```

```

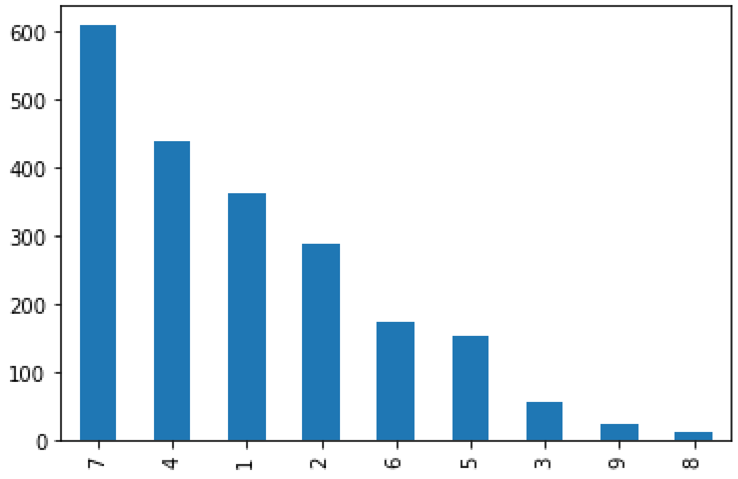
In [11]:
```
# splitting the data into train ,cv,test
y_true = result['Class']
#preprocessing Gene and variation ie taking unnecessary spaces and replacing it with just one underscore
result.Gene = result.Gene.str.replace('\s+',"_")
result.Variation = result.Variation.str.replace('\s+',"_")
# splitting dataset into train and test
X_train,test_df,y_train,y_test=train_test_split(result,y_true,stratify = y_true,test_size=0.2)
#splitting train data into train and cv
train_df,cv_df,y_train,y_cv = train_test_split(X_train,y_train,stratify=y_train,test_size=0.2)
```

In [12]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
```

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

In [13]:
```
train_df['Class'].value_counts().plot.bar()
```

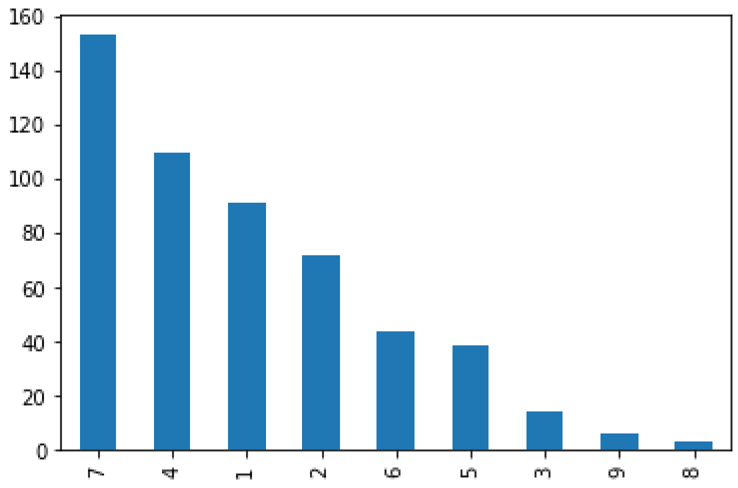
Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f52672613d0>
```



Here there are more 7th class datapoints

In [14]:
```
cv_df['Class'].value_counts().plot.bar()
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5266fe8a50>
```



so if u look at above graphs we come to notice two things

1. data is imbalanced data.
2. since we have splitted using startify on class labels the splitting done is same or similar ie, distribution of classes in train,test,cv are almost similar

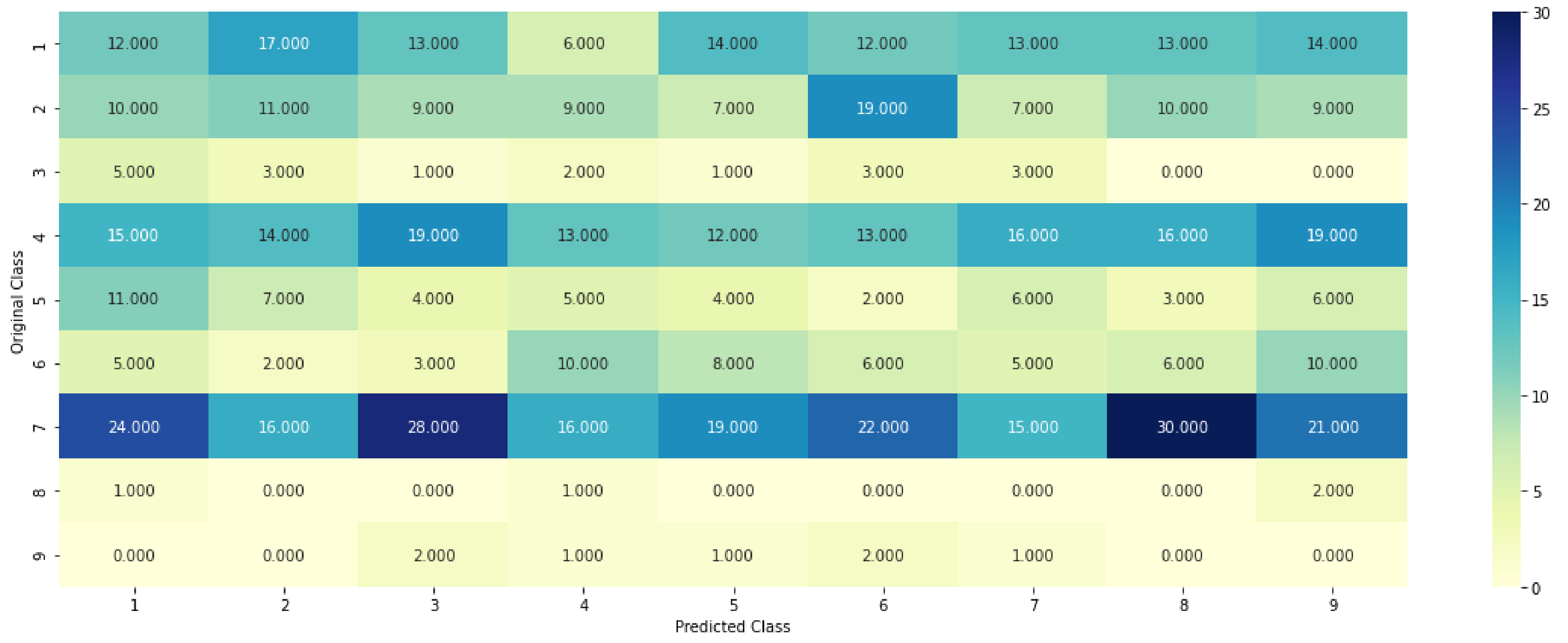
prediction using a random model

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

In [15]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

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

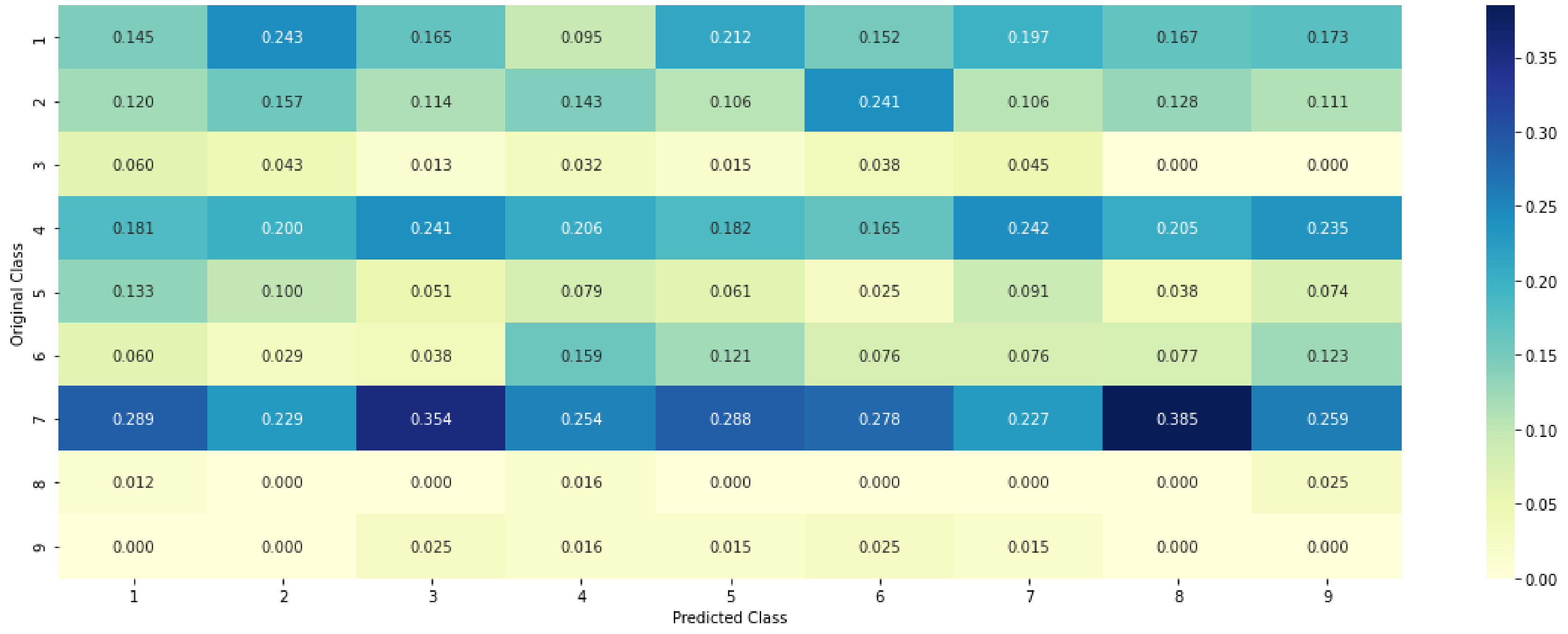
In [16]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.4405157306827943
Log loss on Test Data using Random Model 2.458828077899186
-------------------- Confusion matrix --------------------
```



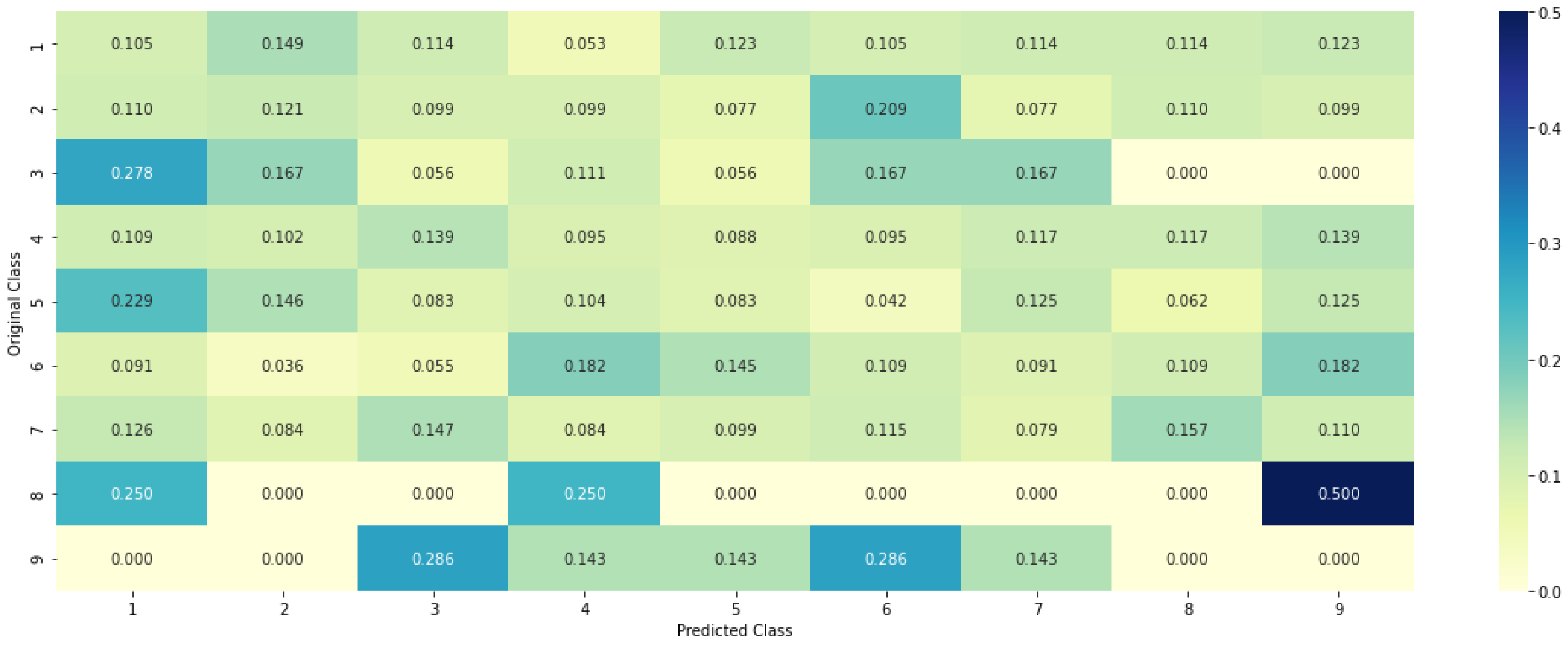
Original \ Predicted	1	2	3	4	5	6	7	8	9
1	12.000	17.000	13.000	6.000	14.000	12.000	13.000	13.000	14.000
2	10.000	11.000	9.000	9.000	7.000	19.000	7.000	10.000	9.000
3	5.000	3.000	1.000	2.000	1.000	3.000	3.000	0.000	0.000
4	15.000	14.000	19.000	13.000	12.000	13.000	16.000	16.000	19.000
5	11.000	7.000	4.000	5.000	4.000	2.000	6.000	3.000	6.000
6	5.000	2.000	3.000	10.000	8.000	6.000	5.000	6.000	10.000
7	24.000	16.000	28.000	16.000	19.000	22.000	15.000	30.000	21.000
8	1.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	2.000
9	0.000	0.000	2.000	1.000	1.000	2.000	1.000	0.000	0.000

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



	1	2	3	4	5	6	7	8	9
1	0.145	0.243	0.165	0.095	0.212	0.152	0.197	0.167	0.173
2	0.120	0.157	0.114	0.143	0.106	0.241	0.106	0.128	0.111
3	0.060	0.043	0.013	0.032	0.015	0.038	0.045	0.000	0.000
4	0.181	0.200	0.241	0.206	0.182	0.165	0.242	0.205	0.235
5	0.133	0.100	0.051	0.079	0.061	0.025	0.091	0.038	0.074
6	0.060	0.029	0.038	0.159	0.121	0.076	0.076	0.077	0.123
7	0.289	0.229	0.354	0.254	0.288	0.278	0.227	0.385	0.259
8	0.012	0.000	0.000	0.016	0.000	0.000	0.000	0.000	0.025
9	0.000	0.000	0.025	0.016	0.015	0.025	0.015	0.000	0.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.105	0.149	0.114	0.053	0.123	0.105	0.114	0.114	0.123
2	0.110	0.121	0.099	0.099	0.077	0.209	0.077	0.110	0.099
3	0.278	0.167	0.056	0.111	0.056	0.167	0.167	0.000	0.000
4	0.109	0.102	0.139	0.095	0.088	0.095	0.117	0.117	0.139
5	0.229	0.146	0.083	0.104	0.083	0.042	0.125	0.062	0.125
6	0.091	0.036	0.055	0.182	0.145	0.109	0.091	0.109	0.182
7	0.126	0.084	0.147	0.084	0.099	0.115	0.079	0.157	0.110
8	0.250	0.000	0.000	0.250	0.000	0.000	0.000	0.000	0.500
9	0.000	0.000	0.286	0.143	0.143	0.286	0.143	0.000	0.000

univariate analysis

In [17]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / number of tim
e it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #         {BRCA1      174
    #          TP53       106
    #          EGFR        86
    #          BRCA2       75
    #          PTEN        69
    #          KIT         61
    #          BRAF        60
    #          ERBB2       47
    #          PDGFRA      46
    #          ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                          3
    # S222D                                         2
    # P130S                                         2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.2
5, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.0
61224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818181
8177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782,
 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.
075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.0
66225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.
093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

univariate analysis of gene feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

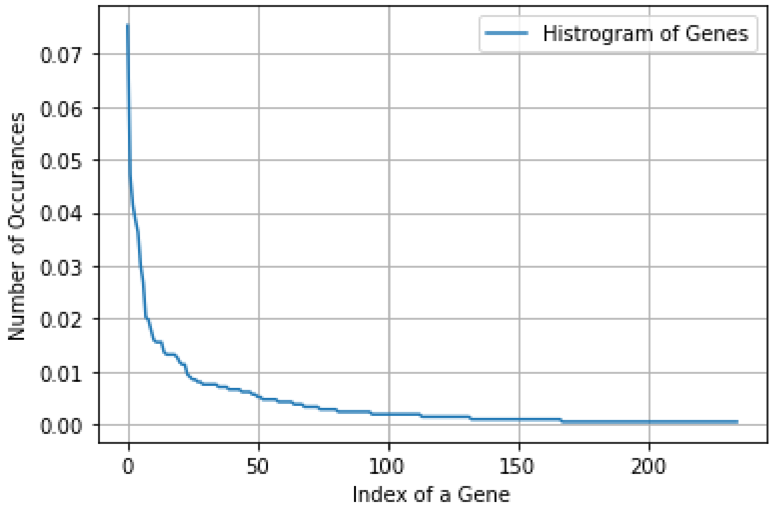
> Q2. How many categories are there and How they are distributed?

In [18]:
```
unique_genes = train_df['Gene'].value_counts()
unique_genes.head(10)
```

Out[18]:
```
BRCA1     160
TP53      100
EGFR       88
PTEN       82
BRCA2      77
KIT        64
BRAF       57
ALK        43
ERBB2      42
CDKN2A     38
Name: Gene, dtype: int64
```

In [19]:
```python
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, and they are distibuted as follows",)
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()
```

```
Ans: There are 235 different categories of genes in the train data, and they are distibuted as follows
```



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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

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

In [20]:
```
k = time.clock()
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
print(time.clock()-k)
```

```
13.441859000000008
```

In [21]:
```
train_gene_feature_responseCoding.shape
```

Out[21]:
```
(2124, 9)
```

In [22]:
```
# 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 [23]:
```
train_gene_feature_onehotCoding.shape
```

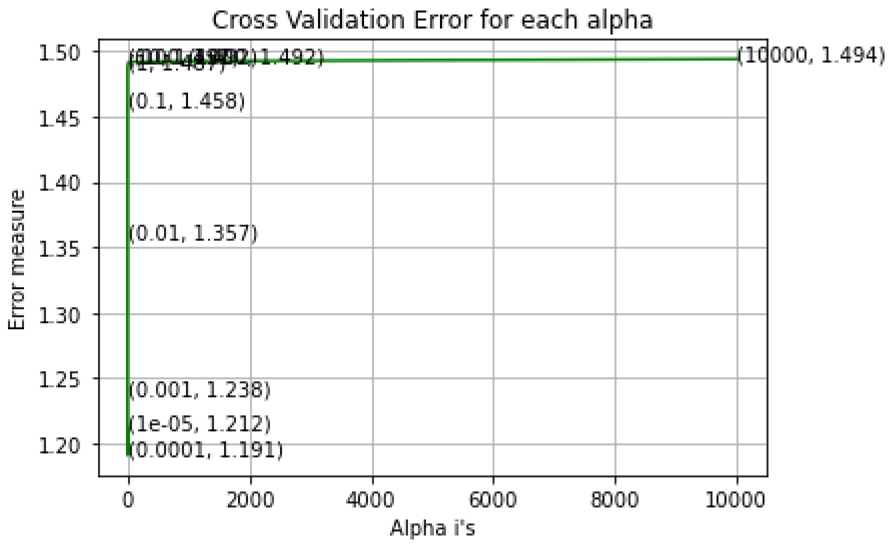
Out[23]:
```
(2124, 234)
```

how good is gene feature in predicting yi

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 [24]:
```
alphavalues = (10 ** x for x in range(-5,5))
cv_log_error_array=[]
alphavalueslist = []
for i in alphavalues:
    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))
    alphavalueslist.append(i)
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=
1e-15))

fig, ax = plt.subplots()
ax.plot(alphavalueslist, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alphavalueslist[i],np.round(txt,3)), (alphavalueslist[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)
k = alphavalueslist[best_alpha]
clf = SGDClassifier(alpha=k, 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 = ', k, "The train log loss is:",log_loss(y_train, predict_y, labels=clf.clas
ses_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', k, "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=
clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', k, "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classe
s_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2117049684254682
For values of alpha =  0.0001 The log loss is: 1.1908227397055235
For values of alpha =  0.001 The log loss is: 1.2378401467277482
For values of alpha =  0.01 The log loss is: 1.3572013524730129
For values of alpha =  0.1 The log loss is: 1.457714046478715
For values of alpha =  1 The log loss is: 1.4874775684765487
For values of alpha =  10 The log loss is: 1.4912783033981398
For values of alpha =  100 The log loss is: 1.491775787982522
For values of alpha =  1000 The log loss is: 1.4923078404934584
For values of alpha =  10000 The log loss is: 1.493932150403798
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9985029871072895
For values of best alpha =  0.0001 The cross validation log loss is: 1.1908227397055235
For values of best alpha =  0.0001 The test log loss is: 1.181845666610073
```

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 [25]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes
 in train dataset?")

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*1
00)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*1
00)
```

```
Q6. How many data points in Test and CV datasets are covered by the  235  genes in train dataset?
Ans
1. In test data 646 out of 665 : 97.14285714285714
2. In cross validation data 517 out of  532 : 97.18045112781954
```

here from above information what we get is.

1. out of 656 gene points in test data 646 of them we have alredy seen them in train data , similarly for test data also

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

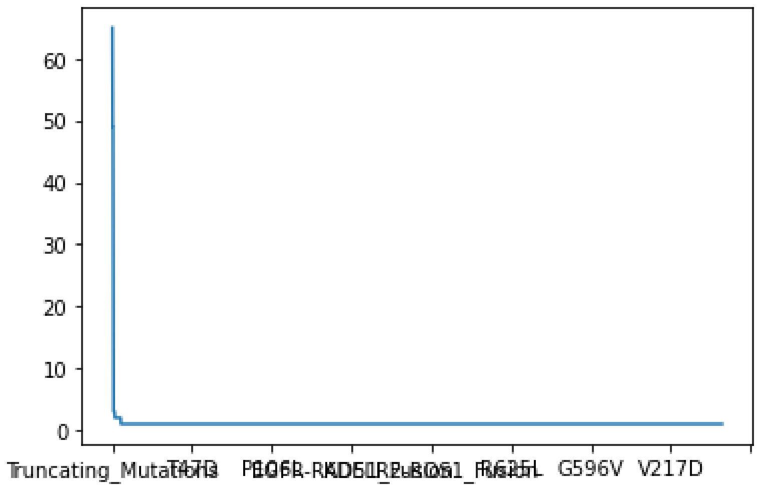
Ans. Variation is a categorical variable

Q8. How many categories are there?

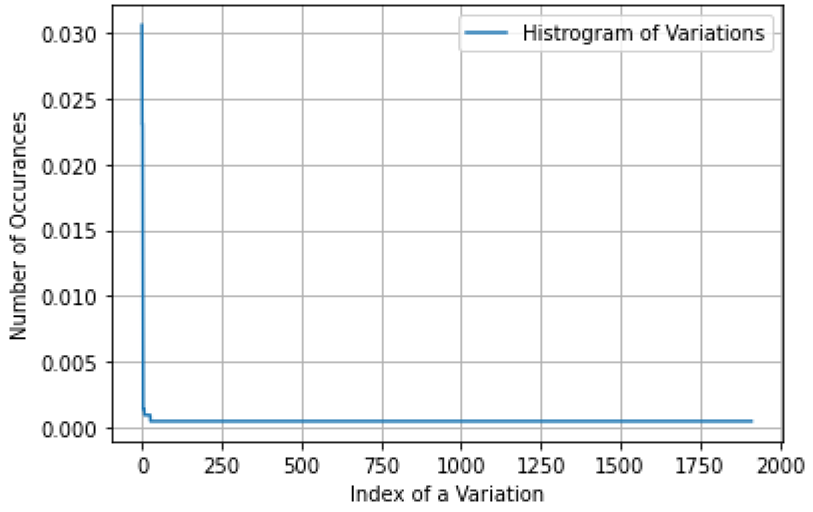
In [26]:
```
unique_variations=train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
unique_variations.head(10)
train_df['Variation'].value_counts().plot()
```

```
Number of Unique Variations : 1912
```

Out[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5266a70950>
```



In [27]:
```
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()
```



1. 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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

One hot Encoding Response coding We will be using both these methods to featurize the Variation Feature

In [28]:
```
# 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 [29]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The sh
ape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [30]:
```
# 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 [31]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The sha
pe of Variation feature:", train_variation_feature_onehotCoding.shape)
```

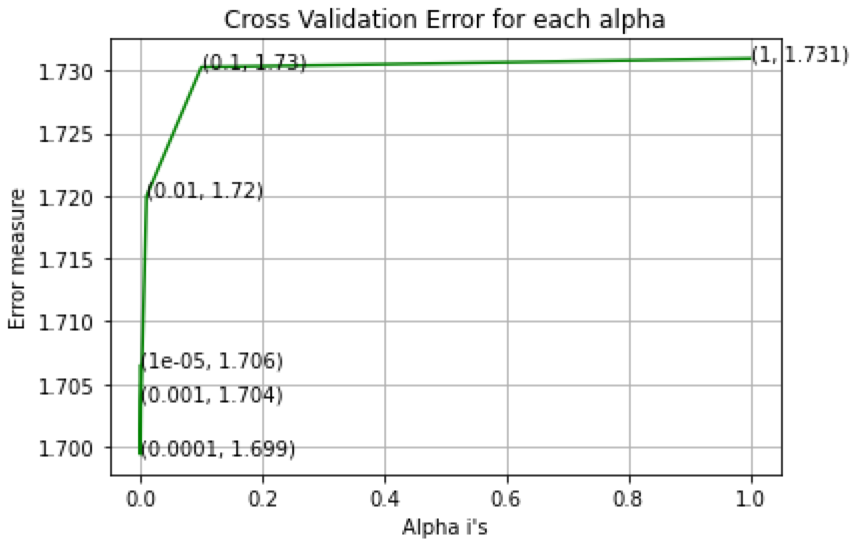
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of V
ariation feature: (2124, 1940)
```

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

Let's build a model just like the earlier!

In [32]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7064488754244915
For values of alpha =  0.0001 The log loss is: 1.6993642244269924
For values of alpha =  0.001 The log loss is: 1.7036445792913124
For values of alpha =  0.01 The log loss is: 1.7199632929804176
For values of alpha =  0.1 The log loss is: 1.7302954233640118
For values of alpha =  1 The log loss is: 1.7309817384700747
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6781213333363376
For values of best alpha =  0.0001 The cross validation log loss is: 1.6993642244269924
For values of best alpha =  0.0001 The test log loss is: 1.7271031183523977
```

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 [33]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cros
s validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*1
00)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*1
00)
```

```
Q12. How many data points are covered by total  1912  genes in test and cross validation data sets?
Ans
1. In test data 60 out of 665 : 9.022556390977442
2. In cross validation data 49 out of  532 : 9.210526315789473
```

Note: variation feature here is not stable across test and train and cv

3.2.3 Univariate Analysis on Text Feature

How many unique words are present in train data?

How are word frequencies distributed?

How to featurize text field?

Is the text feature useful in predicitng y_i?

Is the text feature stable across train, test and CV datasets?

In [34]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 52990
```

In [35]:
```
#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)
```

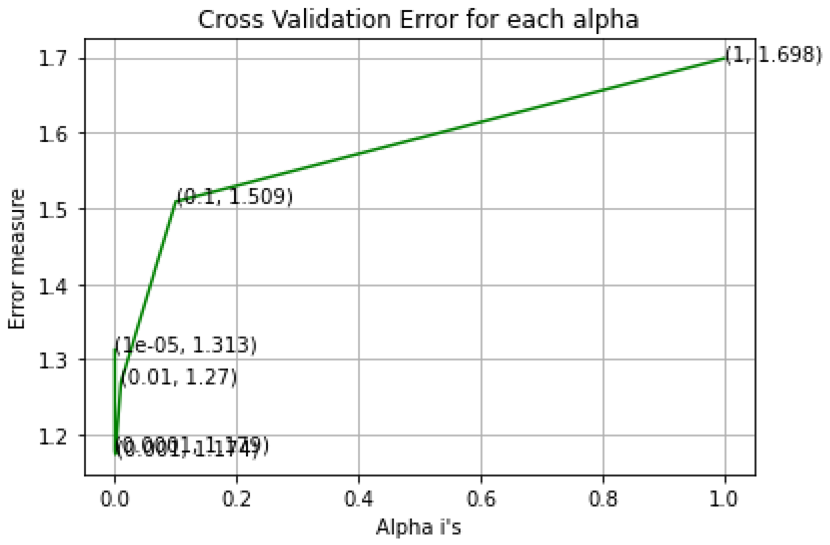
In [36]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCoding.su
m(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(a
xis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1
)).T
```

In [37]:
```
# 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 [38]:
```
#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 [39]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3129536147222927
For values of alpha =  0.0001 The log loss is: 1.179386439263951
For values of alpha =  0.001 The log loss is: 1.1742713036156371
For values of alpha =  0.01 The log loss is: 1.269535101075292
For values of alpha =  0.1 The log loss is: 1.5086360076715486
For values of alpha =  1 The log loss is: 1.6983242671241472
```



```
For values of best alpha =  0.001 The train log loss is: 0.6537559315471534
For values of best alpha =  0.001 The cross validation log loss is: 1.1742713036156371
For values of best alpha =  0.001 The test log loss is: 1.1055922767654471
```

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

Ans. Yes, it seems like!

In [40]:
```
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 [41]:
```
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")
```

```
96.536 % of word of test data appeared in train data
98.729 % of word of Cross Validation appeared in train data
```

ML models

In [42]:
```
#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 [43]:
```
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 [44]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

Stacking the three types of features

In [45]:
```
# merging gene, variance and text features

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

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

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_response
Coding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCod
ing))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

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

In [46]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55164)
(number of data points * number of features) in test data =  (665, 55164)
(number of data points * number of features) in cross validation data = (532, 55164)
```

In [47]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

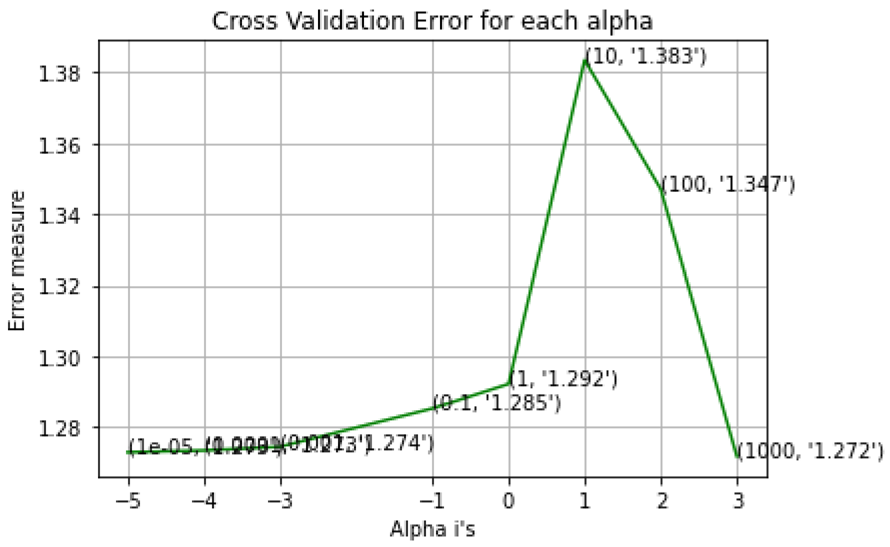
Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [48]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklea
rn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 1.272847835912897
for alpha = 0.0001
Log Loss : 1.2733588050178313
for alpha = 0.001
Log Loss : 1.2743957990020447
for alpha = 0.1
Log Loss : 1.285193204379748
for alpha = 1
Log Loss : 1.2921354813077794
for alpha = 10
Log Loss : 1.383495729753779
for alpha = 100
Log Loss : 1.3470380026402216
for alpha = 1000
Log Loss : 1.271721858086986
```



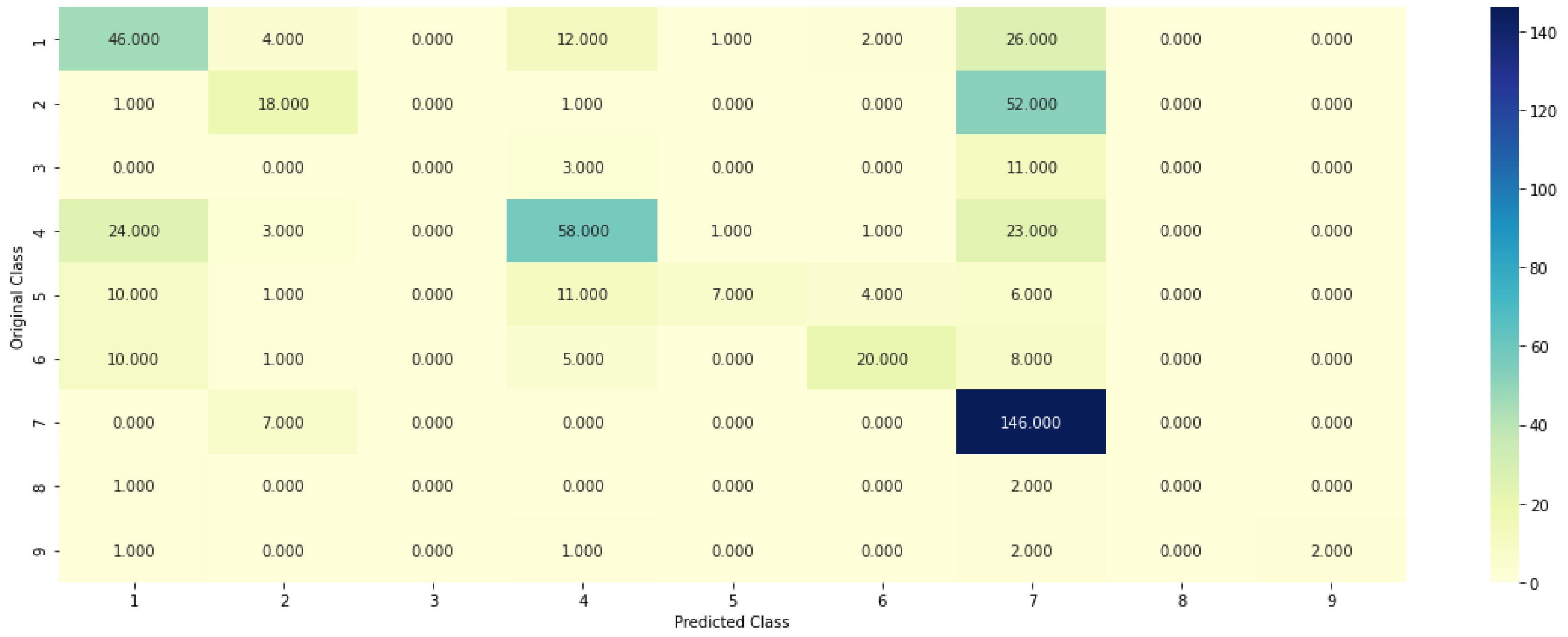
```
For values of best alpha =  1000 The train log loss is: 1.044492707286355
For values of best alpha =  1000 The cross validation log loss is: 1.271721858086986
For values of best alpha =  1000 The test log loss is: 1.3135797210773767
```

In [49]:
```
sig_clf_probs[0]
```

Out[49]:
```
array([0.15239007, 0.22834548, 0.03038728, 0.11576721, 0.06871158,
       0.08145478, 0.30373508, 0.00873128, 0.01047724])
```

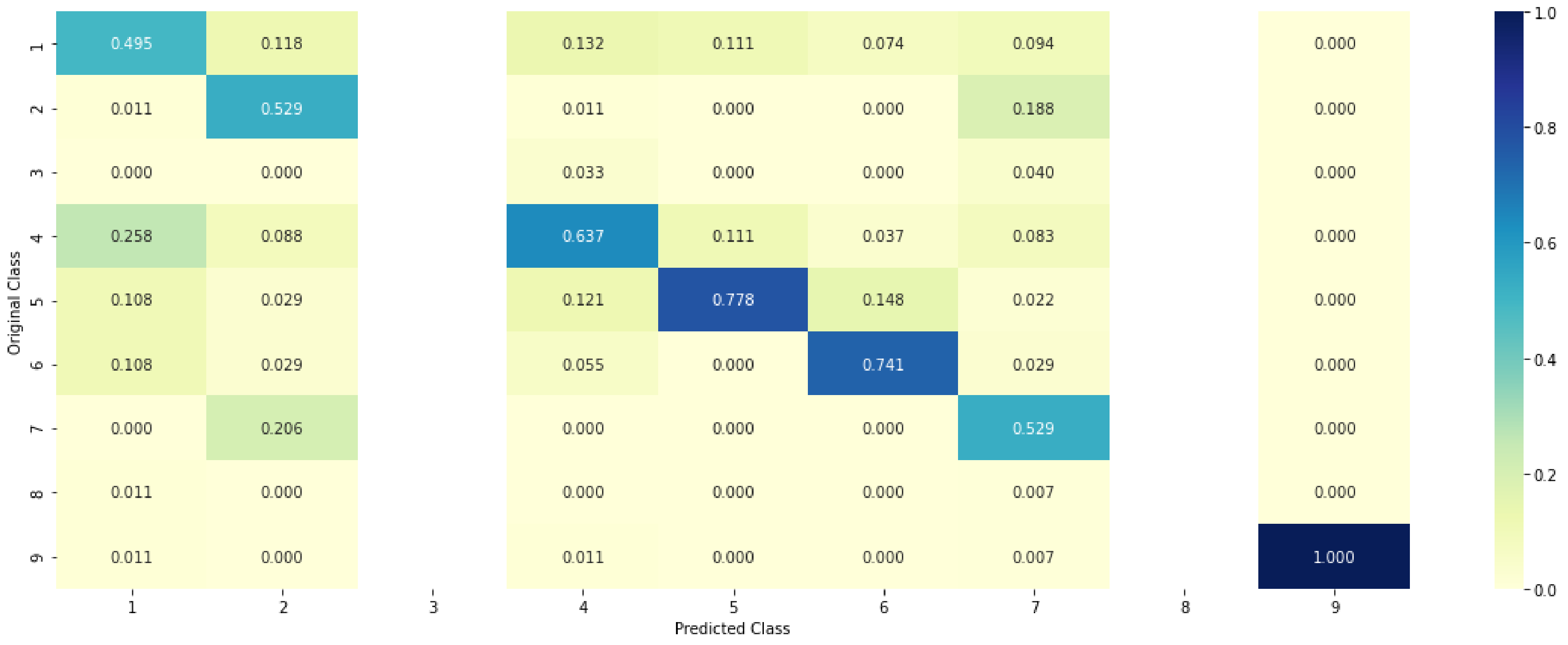
In [50]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklea
rn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.s
hape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.271721858086986
Number of missclassified point : 0.4417293233082707
-------------------- Confusion matrix --------------------
```



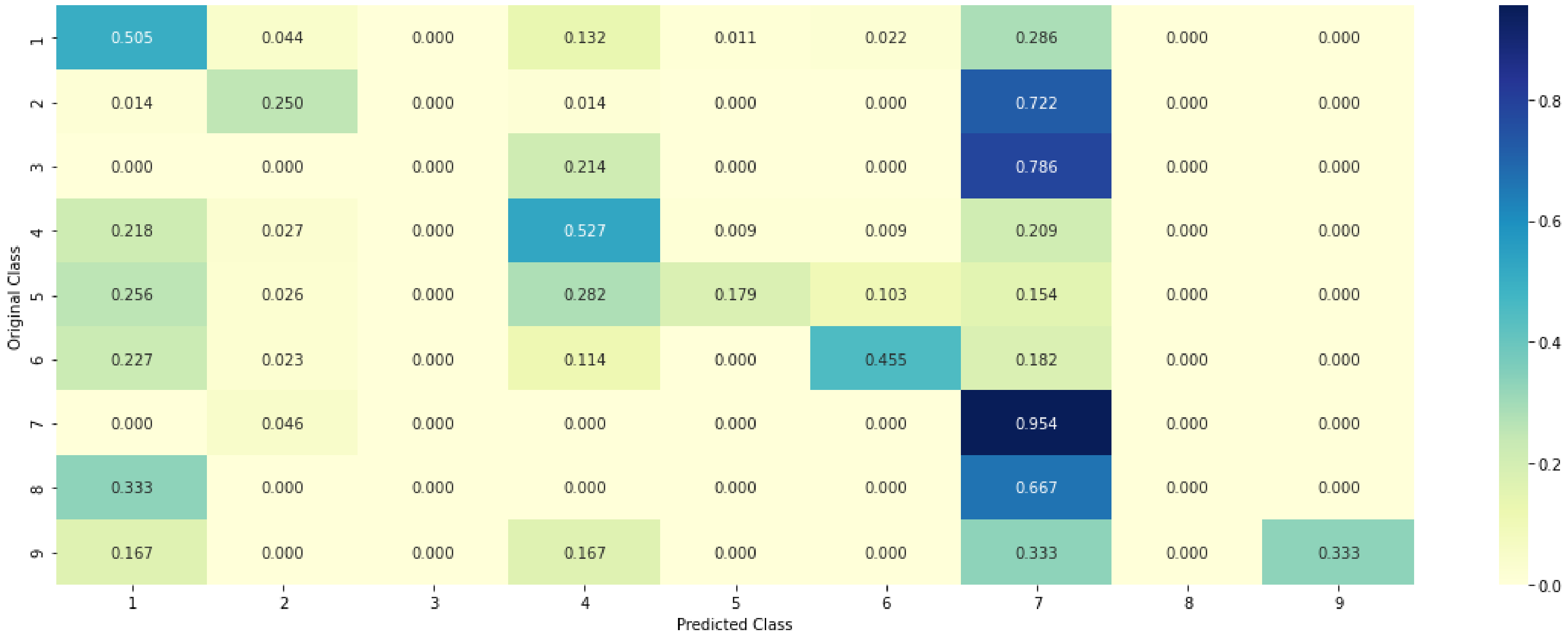
	1	2	3	4	5	6	7	8	9
1	46.000	4.000	0.000	12.000	1.000	2.000	26.000	0.000	0.000
2	1.000	18.000	0.000	1.000	0.000	0.000	52.000	0.000	0.000
3	0.000	0.000	0.000	3.000	0.000	0.000	11.000	0.000	0.000
4	24.000	3.000	0.000	58.000	1.000	1.000	23.000	0.000	0.000
5	10.000	1.000	0.000	11.000	7.000	4.000	6.000	0.000	0.000
6	10.000	1.000	0.000	5.000	0.000	20.000	8.000	0.000	0.000
7	0.000	7.000	0.000	0.000	0.000	0.000	146.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000
9	1.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.495	0.118		0.132	0.111	0.074	0.094		0.000
2	0.011	0.529		0.011	0.000	0.000	0.188		0.000
3	0.000	0.000		0.033	0.000	0.000	0.040		0.000
4	0.258	0.088		0.637	0.111	0.037	0.083		0.000
5	0.108	0.029		0.121	0.778	0.148	0.022		0.000
6	0.108	0.029		0.055	0.000	0.741	0.029		0.000
7	0.000	0.206		0.000	0.000	0.000	0.529		0.000
8	0.011	0.000		0.000	0.000	0.000	0.007		0.000
9	0.011	0.000		0.011	0.000	0.000	0.007		1.000

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



4.1.1.3. Feature Importance, Correctly classified point

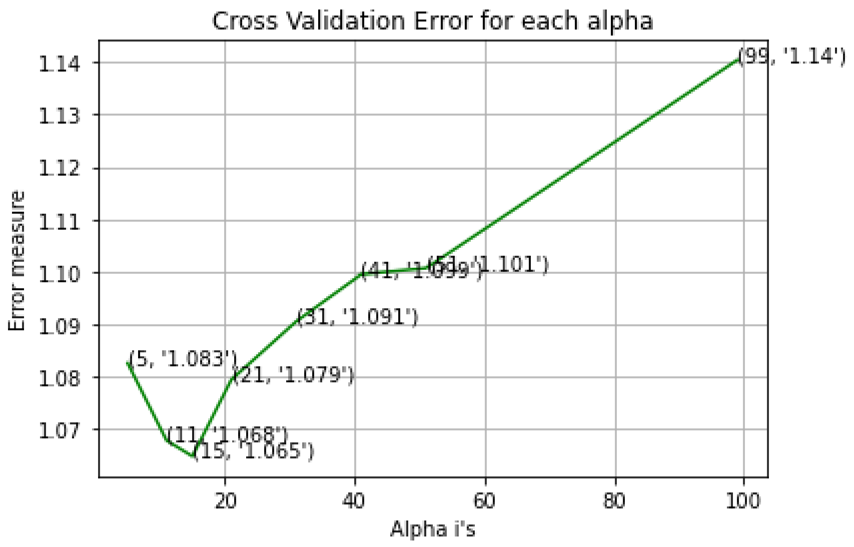
In [51]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1018 0.1965 0.031  0.1344 0.0698 0.0738 0.3735 0.0088 0.0104]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

In [52]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighb
ors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


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

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


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

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

```
for alpha = 5
Log Loss : 1.082610827783673
for alpha = 11
Log Loss : 1.0678613314087946
for alpha = 15
Log Loss : 1.06483168962463
for alpha = 21
Log Loss : 1.0794420070327913
for alpha = 31
Log Loss : 1.09054982239701
for alpha = 41
Log Loss : 1.0994936965760644
for alpha = 51
Log Loss : 1.1006399478208193
for alpha = 99
Log Loss : 1.1403548671097543
```

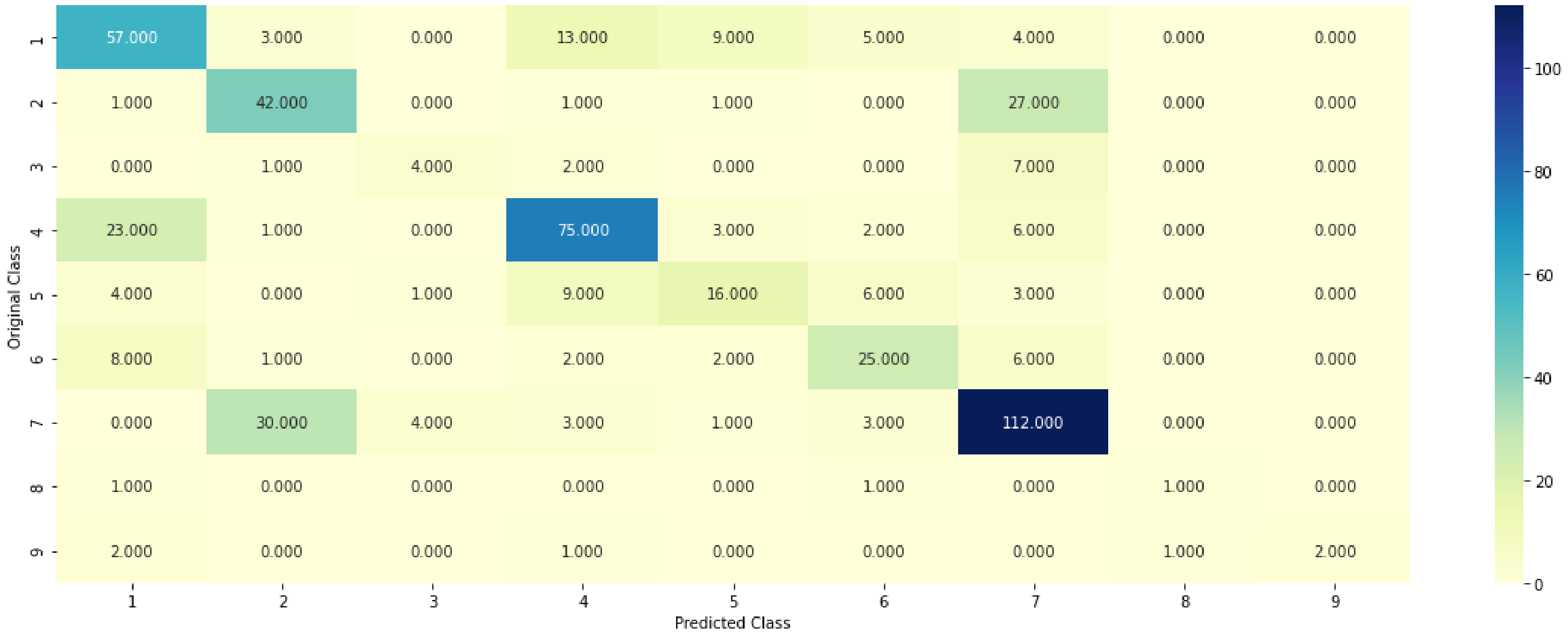


```
For values of best alpha =  15 The train log loss is: 0.6523199773832936
For values of best alpha =  15 The cross validation log loss is: 1.06483168962463
For values of best alpha =  15 The test log loss is: 1.0541105125180696
```

4.2.2. Testing the model with best hyper paramters

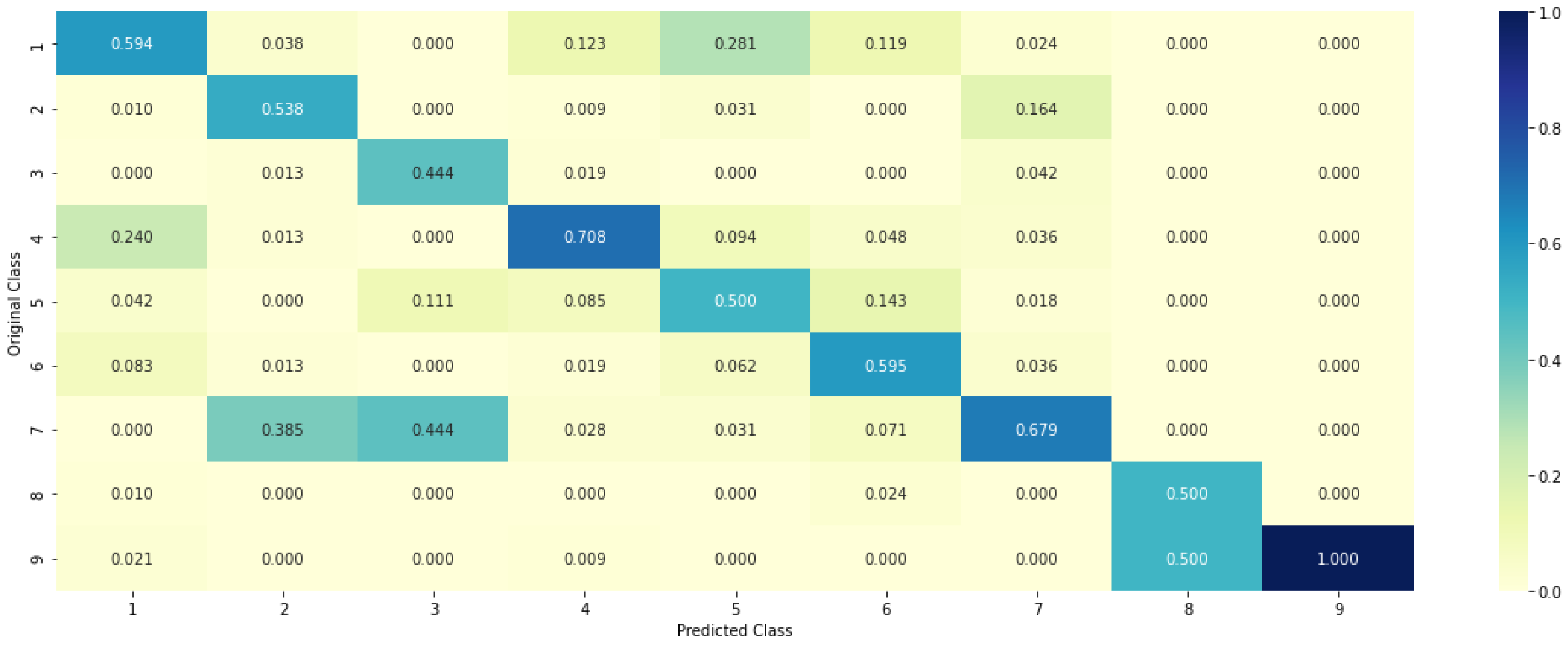
In [53]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighb
ors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.06483168962463
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

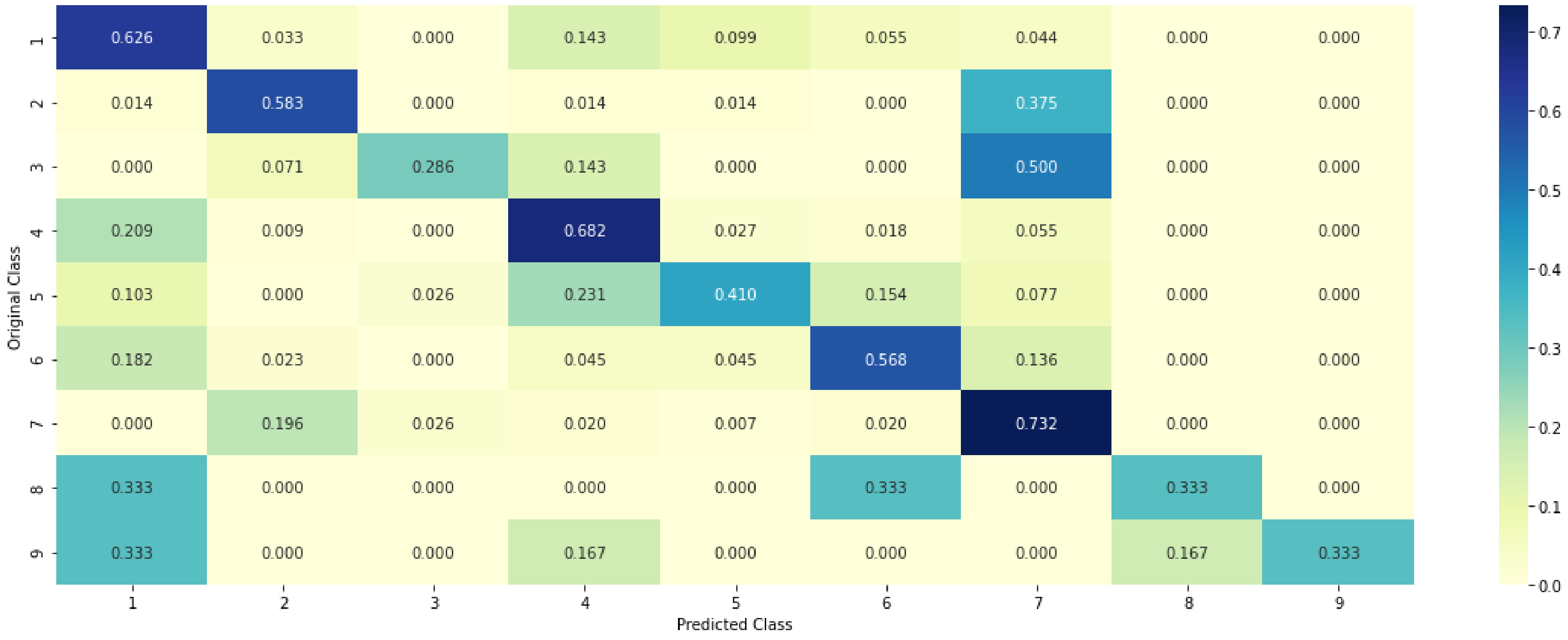


	1	2	3	4	5	6	7	8	9
1	57.000	3.000	0.000	13.000	9.000	5.000	4.000	0.000	0.000
2	1.000	42.000	0.000	1.000	1.000	0.000	27.000	0.000	0.000
3	0.000	1.000	4.000	2.000	0.000	0.000	7.000	0.000	0.000
4	23.000	1.000	0.000	75.000	3.000	2.000	6.000	0.000	0.000
5	4.000	0.000	1.000	9.000	16.000	6.000	3.000	0.000	0.000
6	8.000	1.000	0.000	2.000	2.000	25.000	6.000	0.000	0.000
7	0.000	30.000	4.000	3.000	1.000	3.000	112.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
9	2.000	0.000	0.000	1.000	0.000	0.000	0.000	1.000	2.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.626	0.033	0.000	0.143	0.099	0.055	0.044	0.000	0.000
2	0.014	0.583	0.000	0.014	0.014	0.000	0.375	0.000	0.000
3	0.000	0.071	0.286	0.143	0.000	0.000	0.500	0.000	0.000
4	0.209	0.009	0.000	0.682	0.027	0.018	0.055	0.000	0.000
5	0.103	0.000	0.026	0.231	0.410	0.154	0.077	0.000	0.000
6	0.182	0.023	0.000	0.045	0.045	0.568	0.136	0.000	0.000
7	0.000	0.196	0.026	0.020	0.007	0.020	0.732	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.333	0.000	0.333	0.000
9	0.333	0.000	0.000	0.167	0.000	0.000	0.000	0.167	0.333

4.2.3.Sample Query point -1

In [54]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 5
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [7 7 7 7 7 6 3 7 7 7 7 7 7 7 7]
Fequency of nearest points : Counter({7: 13, 6: 1, 3: 1})
```

sample2

In [55]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

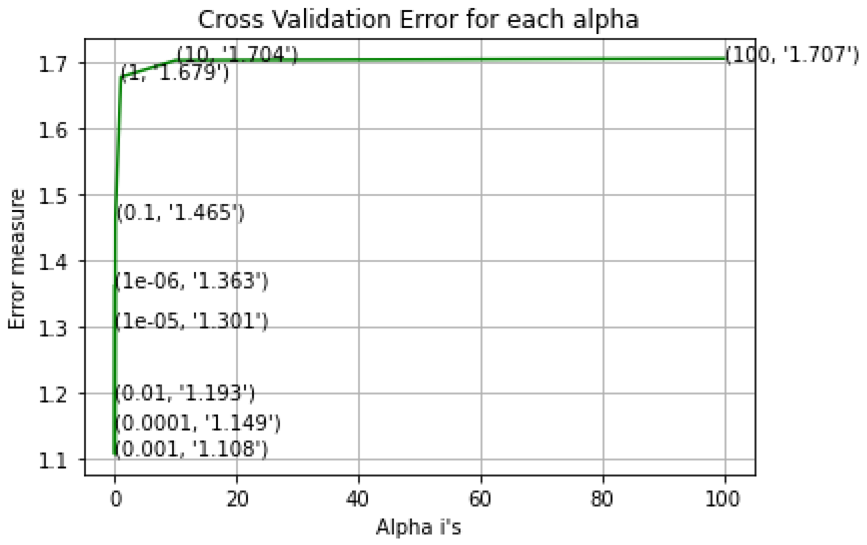
```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [7 7 7 7 7 2 7 7 7
2 2 7 7 7 2]
Fequency of nearest points : Counter({7: 11, 2: 4})
```

4.3. Logistic Regression

with class balancing

In [56]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.36294138079984
for alpha = 1e-05
Log Loss : 1.3012417620859105
for alpha = 0.0001
Log Loss : 1.149313586281625
for alpha = 0.001
Log Loss : 1.1077235053590766
for alpha = 0.01
Log Loss : 1.1925155918669383
for alpha = 0.1
Log Loss : 1.4654834143237738
for alpha = 1
Log Loss : 1.6788734324468384
for alpha = 10
Log Loss : 1.7041432979496354
for alpha = 100
Log Loss : 1.7067955464743005
```

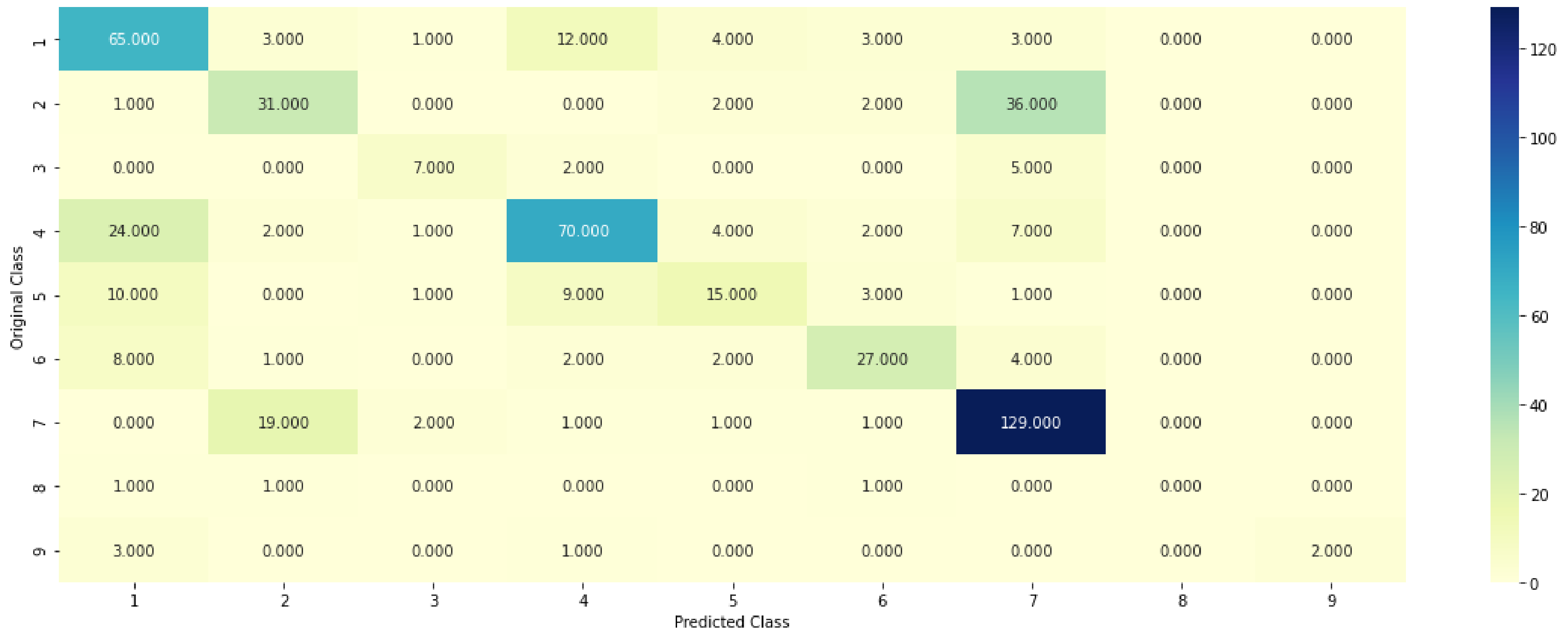


```
For values of best alpha =  0.001 The train log loss is: 0.5258212724205154
For values of best alpha =  0.001 The cross validation log loss is: 1.1077235053590766
For values of best alpha =  0.001 The test log loss is: 1.0331173361507624
```

4.3.1.2. Testing the model with best hyper paramters

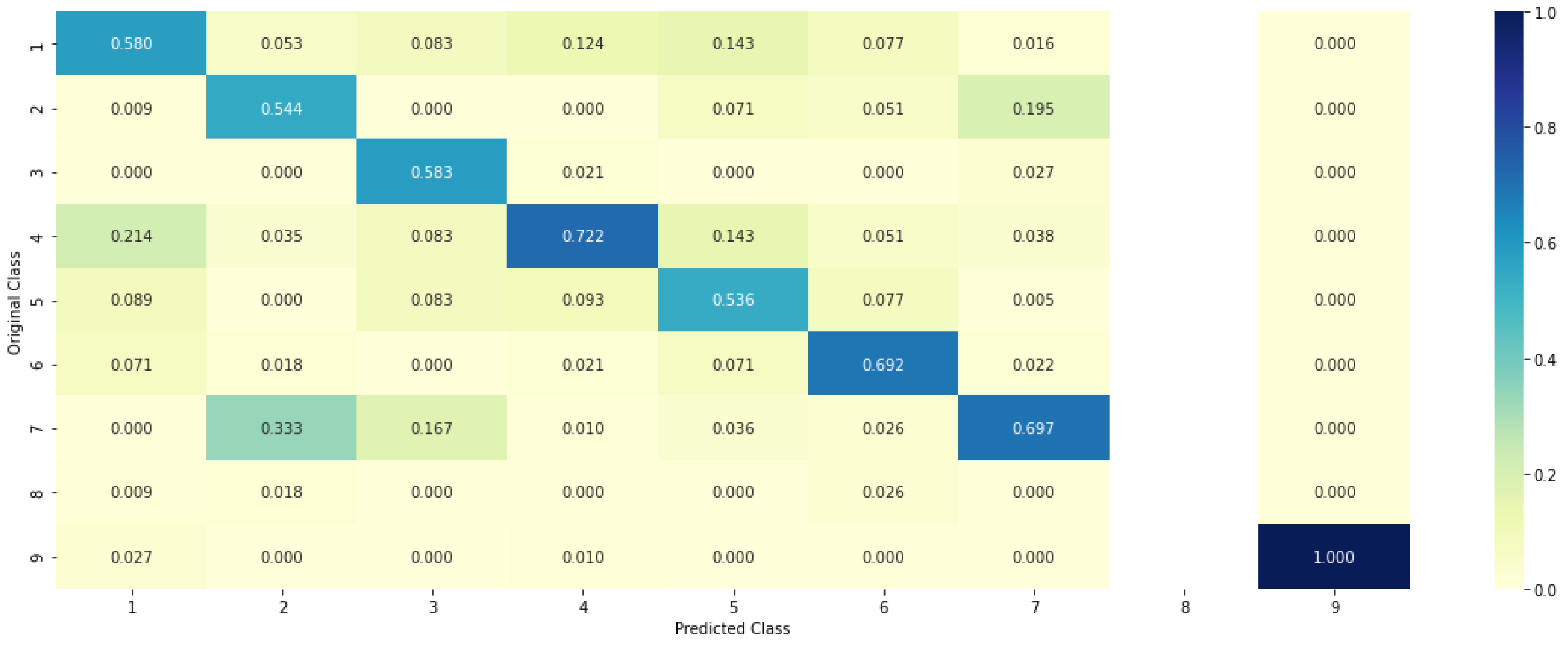
In [57]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.1077235053590766
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



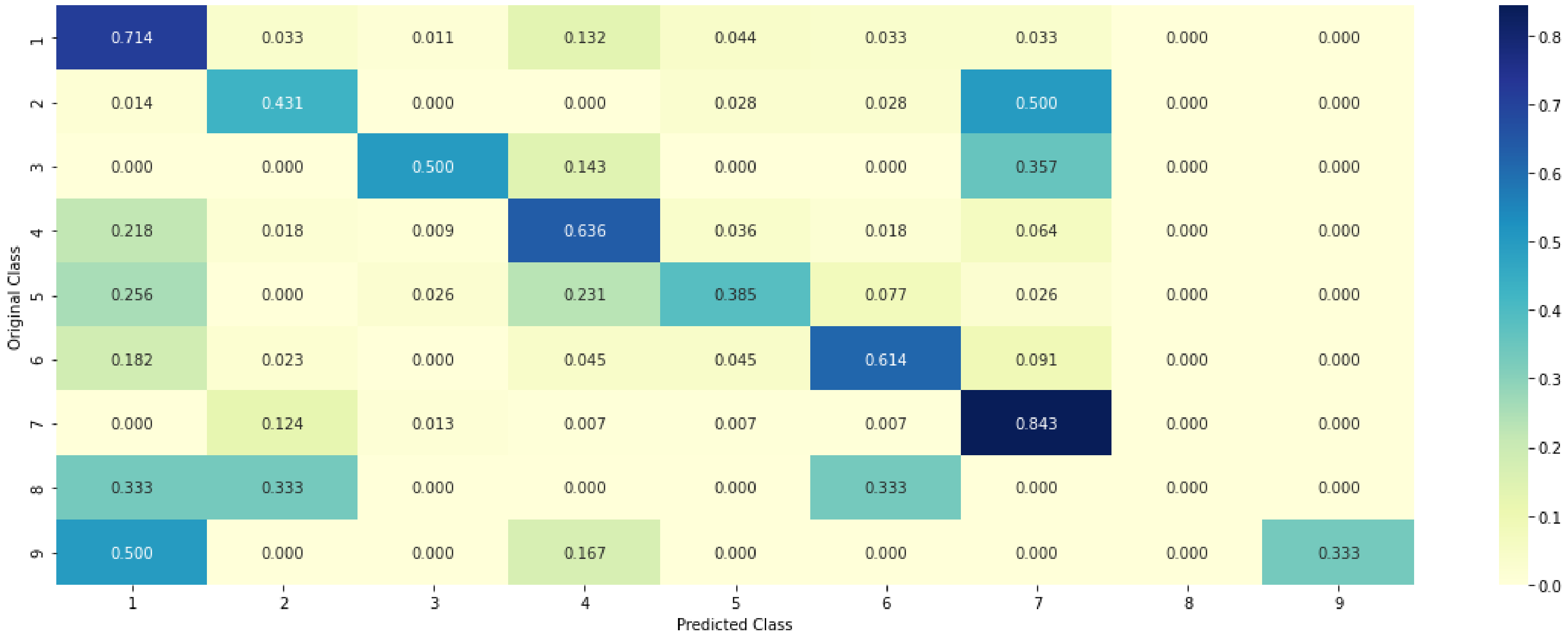
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	65.000	3.000	1.000	12.000	4.000	3.000	3.000	0.000	0.000
2	1.000	31.000	0.000	0.000	2.000	2.000	36.000	0.000	0.000
3	0.000	0.000	7.000	2.000	0.000	0.000	5.000	0.000	0.000
4	24.000	2.000	1.000	70.000	4.000	2.000	7.000	0.000	0.000
5	10.000	0.000	1.000	9.000	15.000	3.000	1.000	0.000	0.000
6	8.000	1.000	0.000	2.000	2.000	27.000	4.000	0.000	0.000
7	0.000	19.000	2.000	1.000	1.000	1.000	129.000	0.000	0.000
8	1.000	1.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
9	3.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.580	0.053	0.083	0.124	0.143	0.077	0.016		0.000
2	0.009	0.544	0.000	0.000	0.071	0.051	0.195		0.000
3	0.000	0.000	0.583	0.021	0.000	0.000	0.027		0.000
4	0.214	0.035	0.083	0.722	0.143	0.051	0.038		0.000
5	0.089	0.000	0.083	0.093	0.536	0.077	0.005		0.000
6	0.071	0.018	0.000	0.021	0.071	0.692	0.022		0.000
7	0.000	0.333	0.167	0.010	0.036	0.026	0.697		0.000
8	0.009	0.018	0.000	0.000	0.000	0.026	0.000		0.000
9	0.027	0.000	0.000	0.010	0.000	0.000	0.000		1.000

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



4.3.1.3. Feature Importance

In [58]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [59]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0997 0.1616 0.0287 0.1419 0.076  0.1318 0.3433 0.0078 0.0092]]
Actual Class : 7
--------------------------------------------------
70 Text feature [constitutive] present in test data point [True]
100 Text feature [missense] present in test data point [True]
122 Text feature [activated] present in test data point [True]
181 Text feature [activating] present in test data point [True]
191 Text feature [constitutively] present in test data point [True]
266 Text feature [function] present in test data point [True]
272 Text feature [rbd] present in test data point [True]
277 Text feature [expressing] present in test data point [True]
334 Text feature [transformation] present in test data point [True]
345 Text feature [variation] present in test data point [True]
354 Text feature [nude] present in test data point [True]
392 Text feature [downstream] present in test data point [True]
422 Text feature [activation] present in test data point [True]
457 Text feature [overexpression] present in test data point [True]
487 Text feature [stably] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

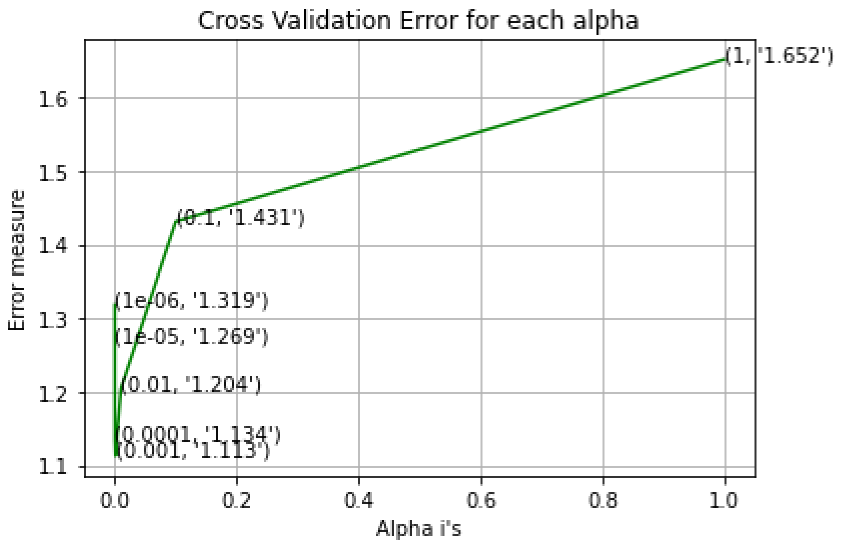
In [60]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.035  0.0746 0.0084 0.075  0.0288 0.0113 0.7558 0.0068 0.0043]]
Actual Class : 7
--------------------------------------------------
70 Text feature [constitutive] present in test data point [True]
91 Text feature [transforming] present in test data point [True]
113 Text feature [3t3] present in test data point [True]
114 Text feature [transform] present in test data point [True]
122 Text feature [activated] present in test data point [True]
129 Text feature [grew] present in test data point [True]
133 Text feature [stat5] present in test data point [True]
155 Text feature [independence] present in test data point [True]
181 Text feature [activating] present in test data point [True]
186 Text feature [technology] present in test data point [True]
191 Text feature [constitutively] present in test data point [True]
217 Text feature [oncogene] present in test data point [True]
230 Text feature [proliferate] present in test data point [True]
247 Text feature [loss] present in test data point [True]
266 Text feature [function] present in test data point [True]
272 Text feature [rbd] present in test data point [True]
276 Text feature [mitogen] present in test data point [True]
277 Text feature [expressing] present in test data point [True]
319 Text feature [cysteine] present in test data point [True]
334 Text feature [transformation] present in test data point [True]
376 Text feature [ligand] present in test data point [True]
392 Text feature [downstream] present in test data point [True]
397 Text feature [predicted] present in test data point [True]
422 Text feature [activation] present in test data point [True]
436 Text feature [deliver] present in test data point [True]
457 Text feature [overexpression] present in test data point [True]
470 Text feature [conserved] present in test data point [True]
475 Text feature [phospho] present in test data point [True]
487 Text feature [stably] present in test data point [True]
Out of the top  500  features  29 are present in query point
```

4.3.2. Without Class balancing

In [61]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3187500585146055
for alpha = 1e-05
Log Loss : 1.269409828851998
for alpha = 0.0001
Log Loss : 1.1343203408531495
for alpha = 0.001
Log Loss : 1.1129013200136924
for alpha = 0.01
Log Loss : 1.2042891998421648
for alpha = 0.1
Log Loss : 1.430907857712483
for alpha = 1
Log Loss : 1.6518348990681004
```

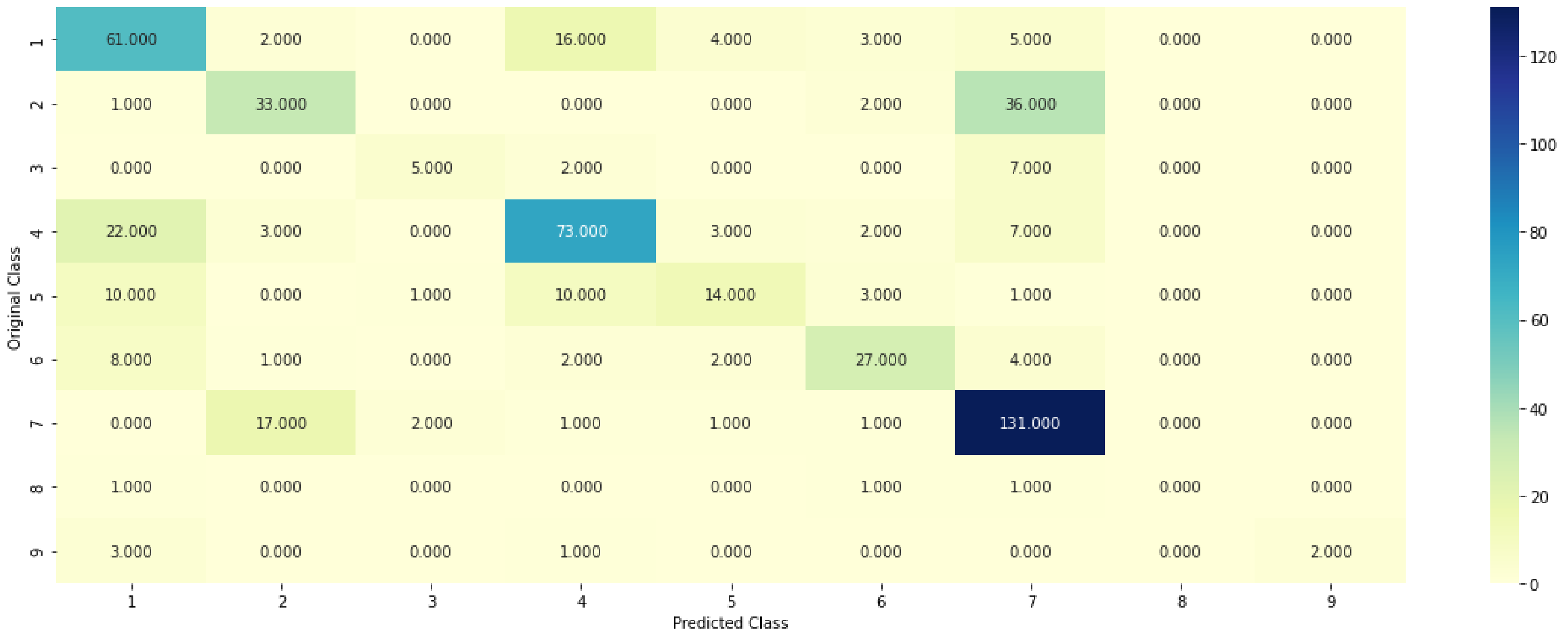


```
For values of best alpha =  0.001 The train log loss is: 0.5224967351264701
For values of best alpha =  0.001 The cross validation log loss is: 1.1129013200136924
For values of best alpha =  0.001 The test log loss is: 1.047232370782306
```

4.3.2.2. Testing model with best hyper parameters

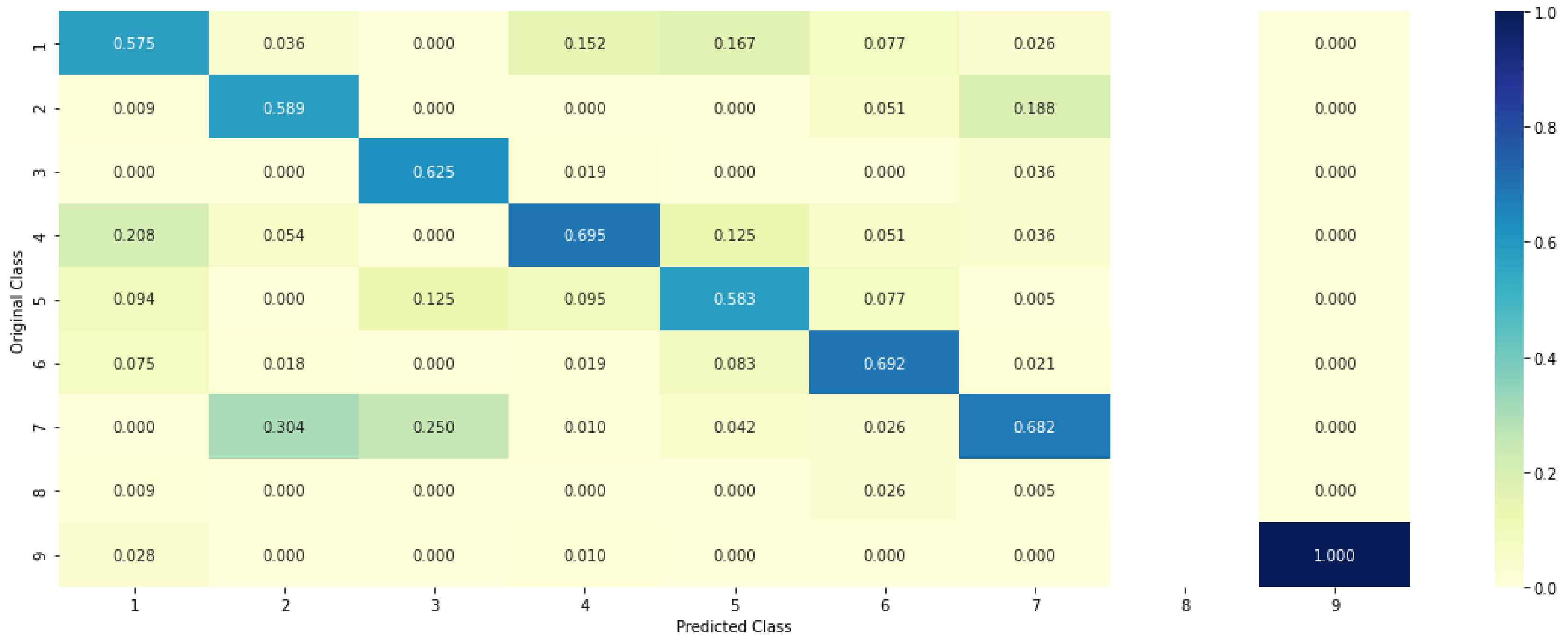
In [62]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.1129013200136924
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



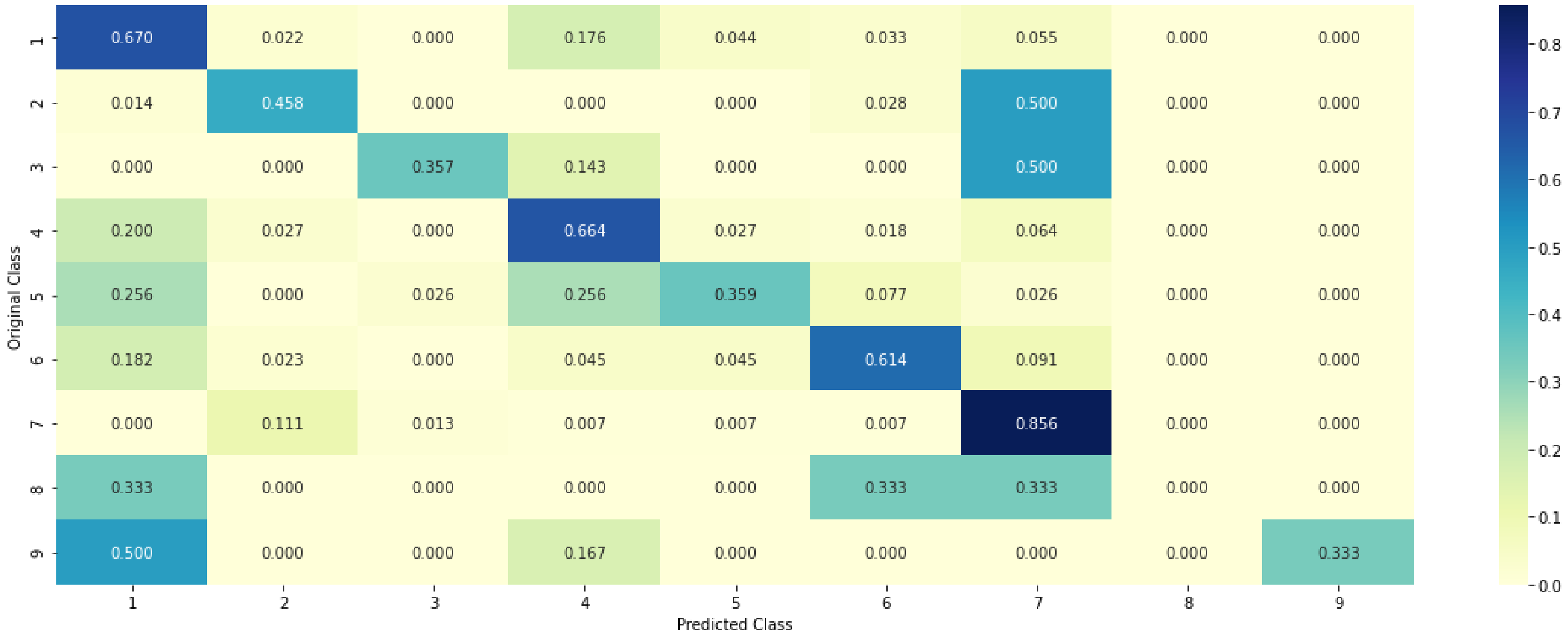
	1	2	3	4	5	6	7	8	9
1	61.000	2.000	0.000	16.000	4.000	3.000	5.000	0.000	0.000
2	1.000	33.000	0.000	0.000	0.000	2.000	36.000	0.000	0.000
3	0.000	0.000	5.000	2.000	0.000	0.000	7.000	0.000	0.000
4	22.000	3.000	0.000	73.000	3.000	2.000	7.000	0.000	0.000
5	10.000	0.000	1.000	10.000	14.000	3.000	1.000	0.000	0.000
6	8.000	1.000	0.000	2.000	2.000	27.000	4.000	0.000	0.000
7	0.000	17.000	2.000	1.000	1.000	1.000	131.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
9	3.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.575	0.036	0.000	0.152	0.167	0.077	0.026		0.000
2	0.009	0.589	0.000	0.000	0.000	0.051	0.188		0.000
3	0.000	0.000	0.625	0.019	0.000	0.000	0.036		0.000
4	0.208	0.054	0.000	0.695	0.125	0.051	0.036		0.000
5	0.094	0.000	0.125	0.095	0.583	0.077	0.005		0.000
6	0.075	0.018	0.000	0.019	0.083	0.692	0.021		0.000
7	0.000	0.304	0.250	0.010	0.042	0.026	0.682		0.000
8	0.009	0.000	0.000	0.000	0.000	0.026	0.005		0.000
9	0.028	0.000	0.000	0.010	0.000	0.000	0.000		1.000

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



	1	2	3	4	5	6	7	8	9
1	0.670	0.022	0.000	0.176	0.044	0.033	0.055	0.000	0.000
2	0.014	0.458	0.000	0.000	0.000	0.028	0.500	0.000	0.000
3	0.000	0.000	0.357	0.143	0.000	0.000	0.500	0.000	0.000
4	0.200	0.027	0.000	0.664	0.027	0.018	0.064	0.000	0.000
5	0.256	0.000	0.026	0.256	0.359	0.077	0.026	0.000	0.000
6	0.182	0.023	0.000	0.045	0.045	0.614	0.091	0.000	0.000
7	0.000	0.111	0.013	0.007	0.007	0.007	0.856	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.333	0.333	0.000	0.000
9	0.500	0.000	0.000	0.167	0.000	0.000	0.000	0.000	0.333

4.3.2.3. Feature Importance, Correctly Classified point

In [63]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0975 0.1655 0.0342 0.136  0.0813 0.1355 0.3316 0.0084 0.0101]]
Actual Class : 7
--------------------------------------------------
108 Text feature [constitutive] present in test data point [True]
232 Text feature [activating] present in test data point [True]
251 Text feature [activated] present in test data point [True]
287 Text feature [constitutively] present in test data point [True]
332 Text feature [missense] present in test data point [True]
359 Text feature [expressing] present in test data point [True]
387 Text feature [transformation] present in test data point [True]
445 Text feature [rbd] present in test data point [True]
486 Text feature [nude] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

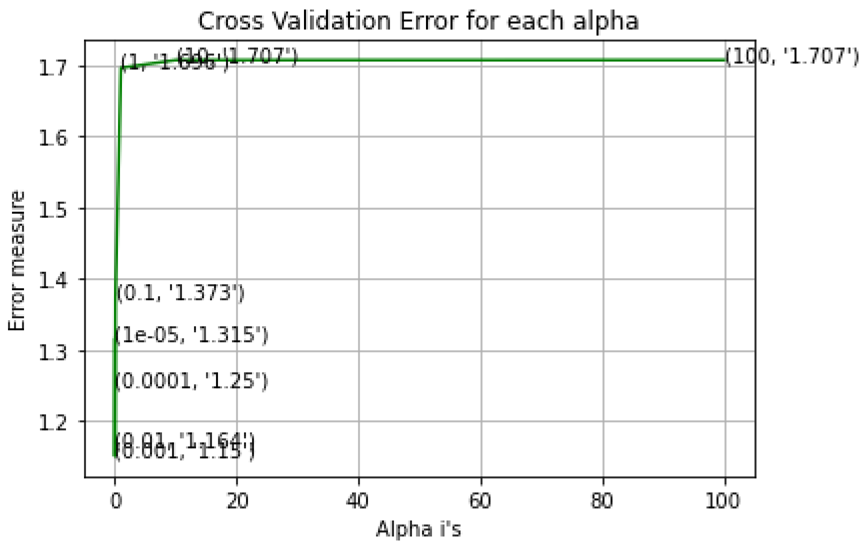
In [64]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0341 0.0723 0.0074 0.0752 0.0268 0.011  0.7642 0.0054 0.0038]]
Actual Class : 7
--------------------------------------------------
108 Text feature [constitutive] present in test data point [True]
128 Text feature [transforming] present in test data point [True]
189 Text feature [3t3] present in test data point [True]
201 Text feature [transform] present in test data point [True]
225 Text feature [independence] present in test data point [True]
232 Text feature [activating] present in test data point [True]
251 Text feature [activated] present in test data point [True]
269 Text feature [stat5] present in test data point [True]
287 Text feature [constitutively] present in test data point [True]
289 Text feature [technology] present in test data point [True]
294 Text feature [grew] present in test data point [True]
356 Text feature [cysteine] present in test data point [True]
359 Text feature [expressing] present in test data point [True]
387 Text feature [transformation] present in test data point [True]
399 Text feature [oncogene] present in test data point [True]
400 Text feature [proliferate] present in test data point [True]
445 Text feature [rbd] present in test data point [True]
Out of the top  500  features  17 are present in query point
```

4.4. Linear Support Vector Machines

In [65]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/gen
erated/sklearn.svm.SVC.html

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

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_stat
e=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for C = 1e-05
Log Loss : 1.314728993307713
for C = 0.0001
Log Loss : 1.2502178546124403
for C = 0.001
Log Loss : 1.1500876328835536
for C = 0.01
Log Loss : 1.1637865648100139
for C = 0.1
Log Loss : 1.3733430875163954
for C = 1
Log Loss : 1.6962686297971055
for C = 10
Log Loss : 1.7072212602074643
for C = 100
Log Loss : 1.7072693840971944
```

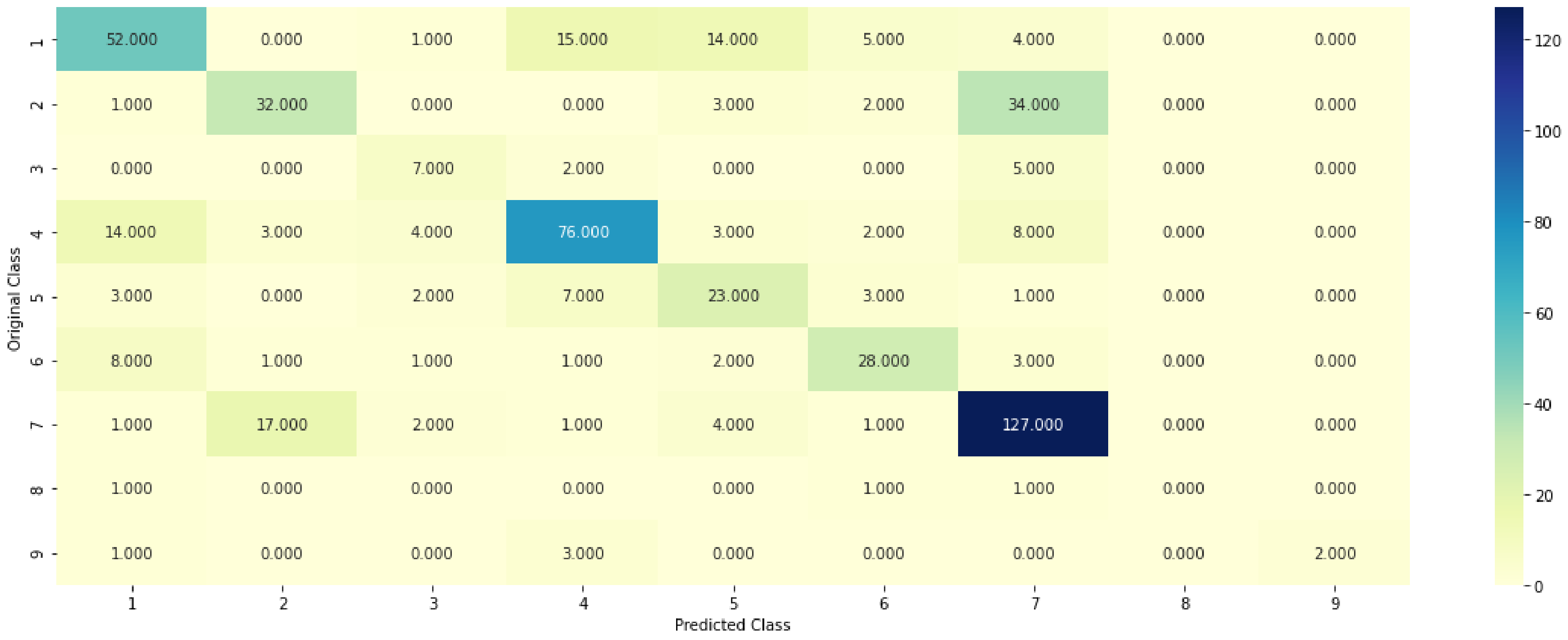


```
For values of best alpha =  0.001 The train log loss is: 0.5297204834298374
For values of best alpha =  0.001 The cross validation log loss is: 1.1500876328835536
For values of best alpha =  0.001 The test log loss is: 1.1038364842102042
```

4.4.2. Testing model with best hyper parameters

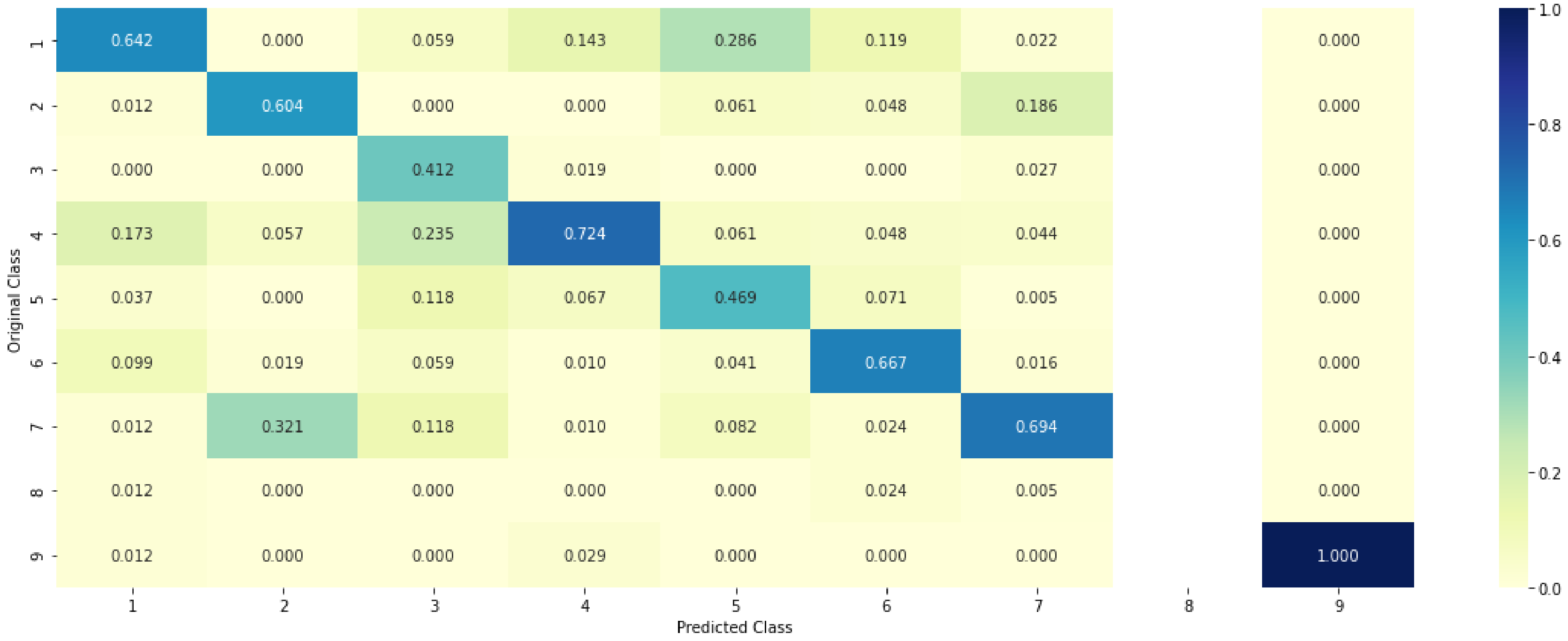
In [66]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/gen
erated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 1.1500876328835536
Number of mis-classified points : 0.347744360090225563
-------------------- Confusion matrix --------------------
```

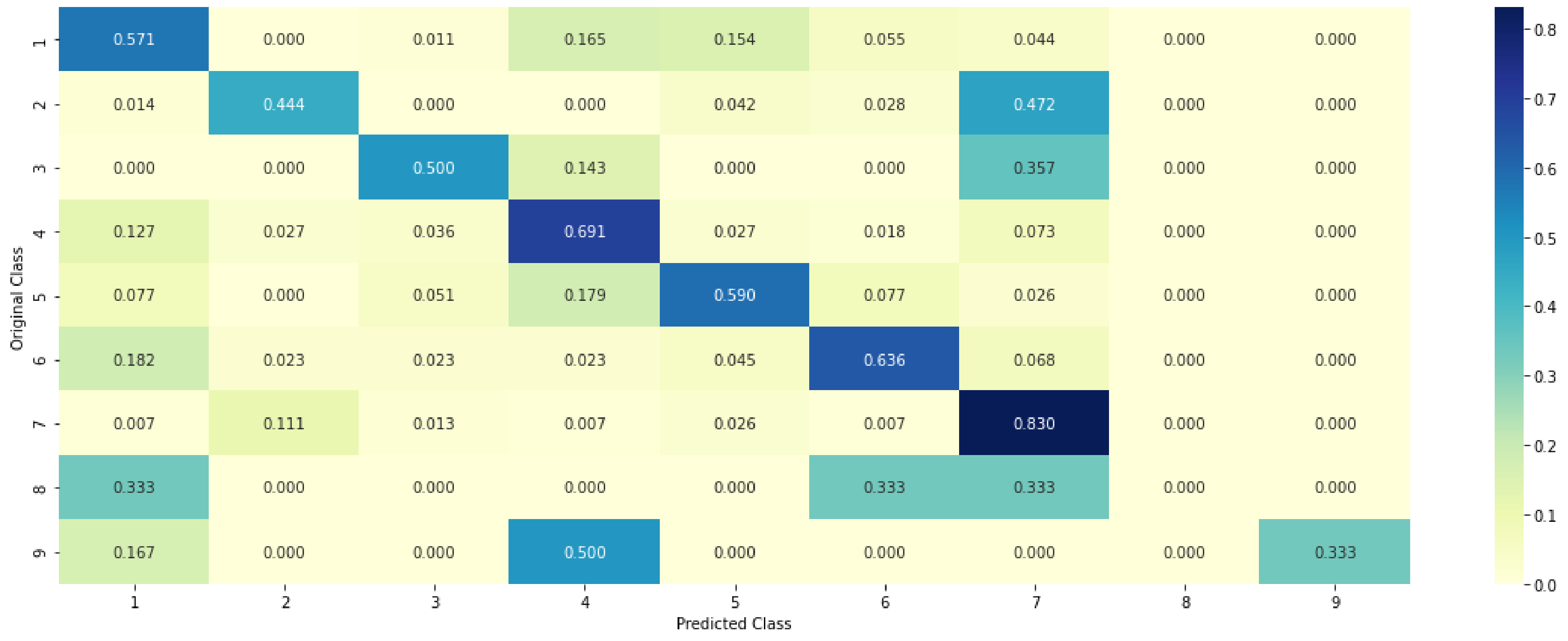


	1	2	3	4	5	6	7	8	9
1	52.000	0.000	1.000	15.000	14.000	5.000	4.000	0.000	0.000
2	1.000	32.000	0.000	0.000	3.000	2.000	34.000	0.000	0.000
3	0.000	0.000	7.000	2.000	0.000	0.000	5.000	0.000	0.000
4	14.000	3.000	4.000	76.000	3.000	2.000	8.000	0.000	0.000
5	3.000	0.000	2.000	7.000	23.000	3.000	1.000	0.000	0.000
6	8.000	1.000	1.000	1.000	2.000	28.000	3.000	0.000	0.000
7	1.000	17.000	2.000	1.000	4.000	1.000	127.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
9	1.000	0.000	0.000	3.000	0.000	0.000	0.000	0.000	2.000

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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [67]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1134 0.1152 0.0325 0.1662 0.0836 0.1927 0.2793 0.0088 0.0082]]
Actual Class : 7
--------------------------------------------------
415 Text feature [constitutive] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [68]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index
],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0538 0.0592 0.0132 0.1053 0.0424 0.0125 0.7036 0.0051 0.0048]]
Actual Class : 7
--------------------------------------------------
415 Text feature [constitutive] present in test data point [True]
436 Text feature [3t3] present in test data point [True]
437 Text feature [transform] present in test data point [True]
439 Text feature [independence] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [69]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_spli
t=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_de
crease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_star
t=False,
# class_weight=None)

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

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

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
```

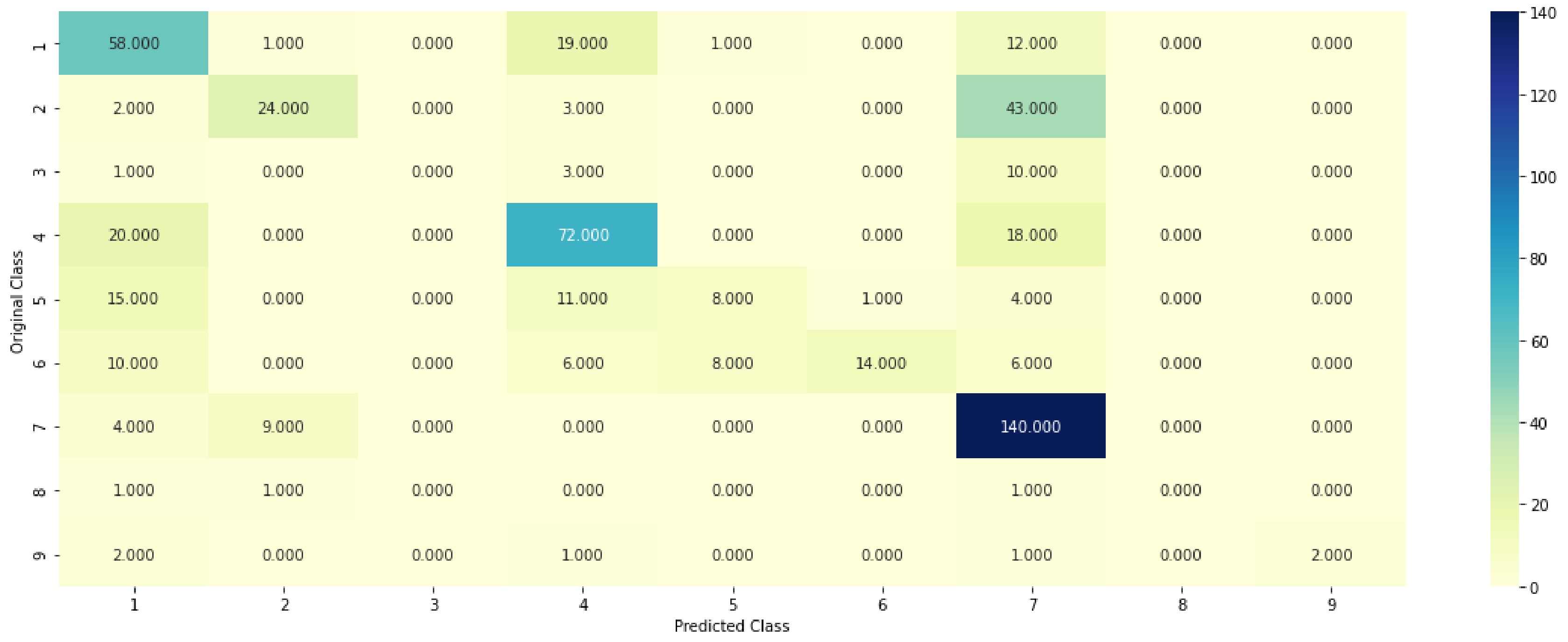
```
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-
1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2510752385857697
for n_estimators = 100 and max depth =  10
Log Loss : 1.1710949046215324
for n_estimators = 200 and max depth =  5
Log Loss : 1.2425918514711494
for n_estimators = 200 and max depth =  10
Log Loss : 1.1658030263909924
for n_estimators = 500 and max depth =  5
Log Loss : 1.232035305318242
for n_estimators = 500 and max depth =  10
Log Loss : 1.160224784941954
for n_estimators = 1000 and max depth =  5
Log Loss : 1.226966071110765
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1566287006971108
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2255781767795293
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1559219378180812
For values of best estimator =  2000 The train log loss is: 0.6955228505989779
For values of best estimator =  2000 The cross validation log loss is: 1.1559219378180812
For values of best estimator =  2000 The test log loss is: 1.1411004344682578
```

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

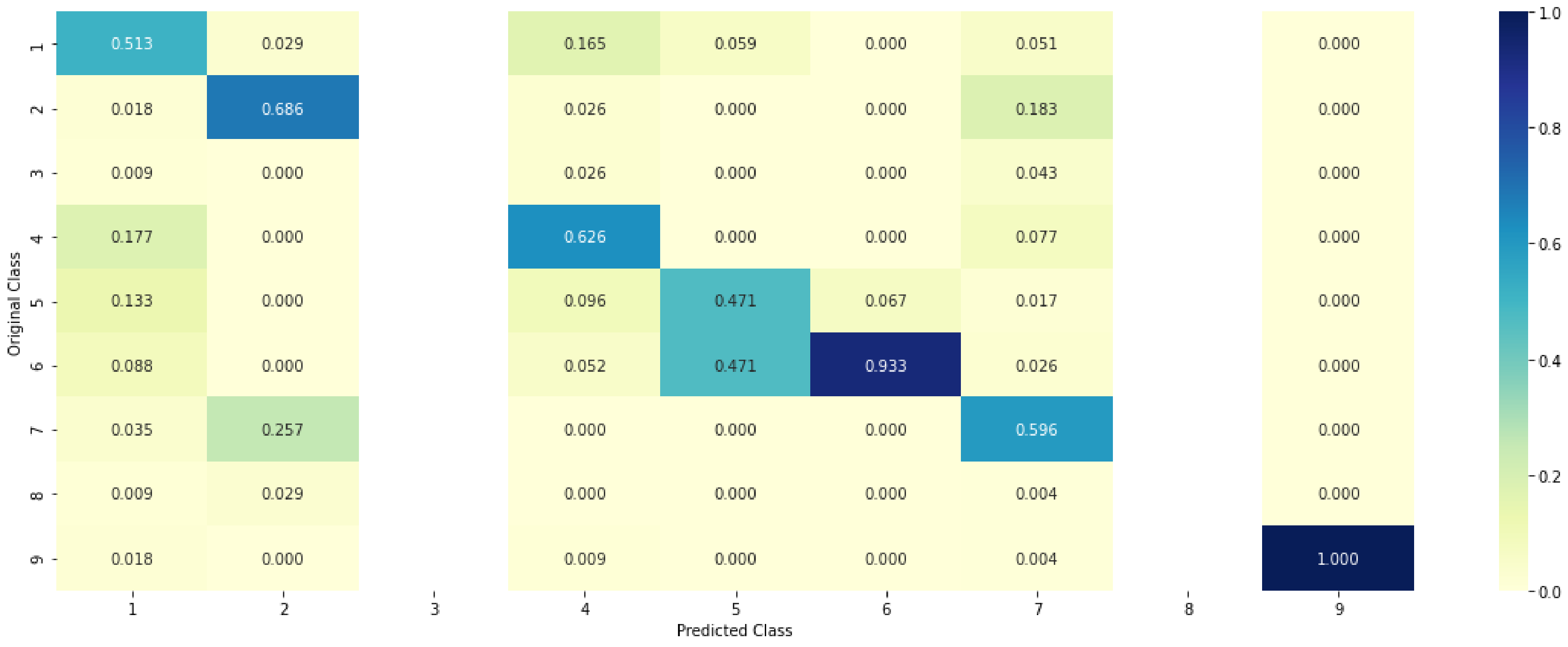
In [70]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_spli
t=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_de
crease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_star
t=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.1559219378180812
Number of mis-classified points : 0.40225563909774437
-------------------- Confusion matrix --------------------
```

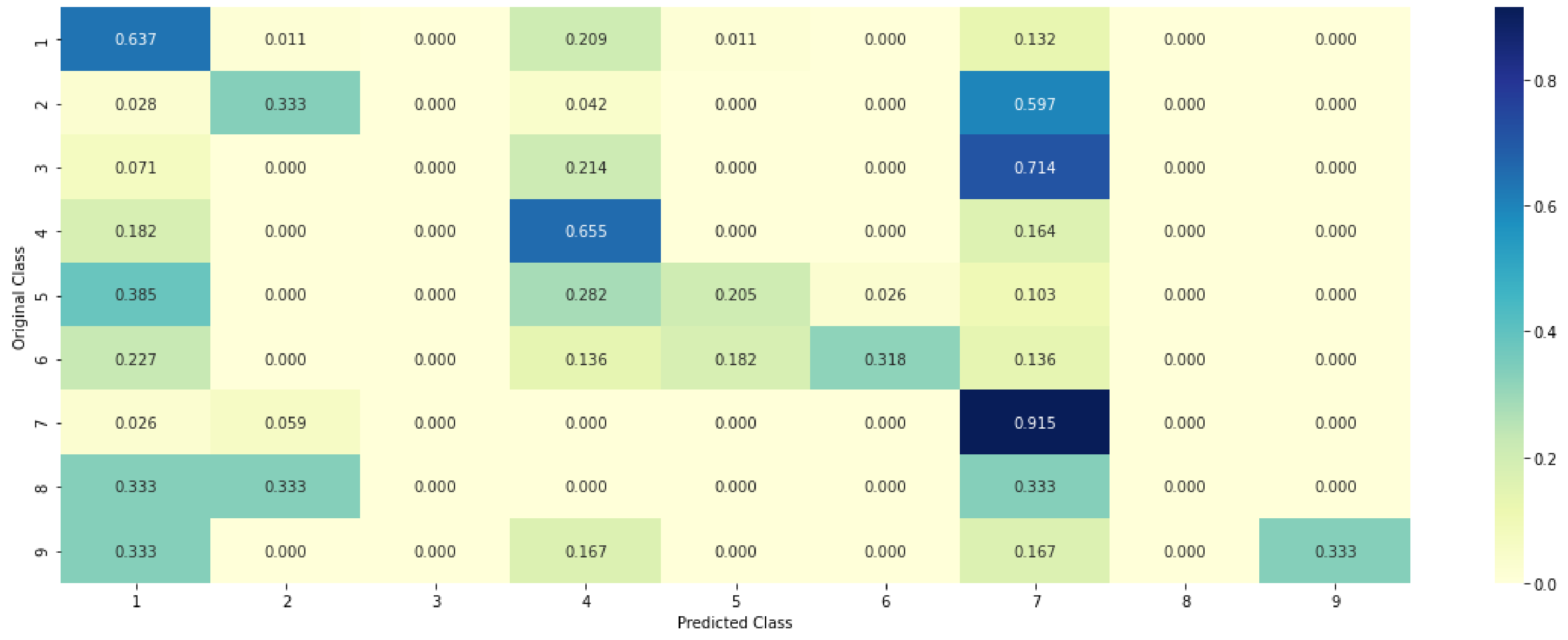


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	58.000	1.000	0.000	19.000	1.000	0.000	12.000	0.000	0.000
2	2.000	24.000	0.000	3.000	0.000	0.000	43.000	0.000	0.000
3	1.000	0.000	0.000	3.000	0.000	0.000	10.000	0.000	0.000
4	20.000	0.000	0.000	72.000	0.000	0.000	18.000	0.000	0.000
5	15.000	0.000	0.000	11.000	8.000	1.000	4.000	0.000	0.000
6	10.000	0.000	0.000	6.000	8.000	14.000	6.000	0.000	0.000
7	4.000	9.000	0.000	0.000	0.000	0.000	140.000	0.000	0.000
8	1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	2.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	2.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.637	0.011	0.000	0.209	0.011	0.000	0.132	0.000	0.000
2	0.028	0.333	0.000	0.042	0.000	0.000	0.597	0.000	0.000
3	0.071	0.000	0.000	0.214	0.000	0.000	0.714	0.000	0.000
4	0.182	0.000	0.000	0.655	0.000	0.000	0.164	0.000	0.000
5	0.385	0.000	0.000	0.282	0.205	0.026	0.103	0.000	0.000
6	0.227	0.000	0.000	0.136	0.182	0.318	0.136	0.000	0.000
7	0.026	0.059	0.000	0.000	0.000	0.000	0.915	0.000	0.000
8	0.333	0.333	0.000	0.000	0.000	0.000	0.333	0.000	0.000
9	0.333	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.333

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [71]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int
(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0943 0.2246 0.0245 0.0926 0.0611 0.0488 0.4377 0.0086 0.0077]]
Actual Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
4 Text feature [missense] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [treatment] present in test data point [True]
7 Text feature [activated] present in test data point [True]
9 Text feature [function] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
15 Text feature [erk] present in test data point [True]
16 Text feature [growth] present in test data point [True]
18 Text feature [signaling] present in test data point [True]
20 Text feature [oncogenic] present in test data point [True]
24 Text feature [functional] present in test data point [True]
25 Text feature [receptor] present in test data point [True]
27 Text feature [akt] present in test data point [True]
29 Text feature [cells] present in test data point [True]
31 Text feature [therapy] present in test data point [True]
35 Text feature [activate] present in test data point [True]
40 Text feature [patients] present in test data point [True]
48 Text feature [lines] present in test data point [True]
49 Text feature [constitutively] present in test data point [True]
51 Text feature [mek] present in test data point [True]
52 Text feature [expression] present in test data point [True]
53 Text feature [expressing] present in test data point [True]
55 Text feature [protein] present in test data point [True]
58 Text feature [egfr] present in test data point [True]
59 Text feature [variants] present in test data point [True]
60 Text feature [downstream] present in test data point [True]
62 Text feature [advanced] present in test data point [True]
64 Text feature [resistance] present in test data point [True]
65 Text feature [cell] present in test data point [True]
66 Text feature [patient] present in test data point [True]
67 Text feature [defective] present in test data point [True]
71 Text feature [phosphorylated] present in test data point [True]
74 Text feature [factor] present in test data point [True]
80 Text feature [erk1] present in test data point [True]
81 Text feature [dna] present in test data point [True]
85 Text feature [extracellular] present in test data point [True]
88 Text feature [proliferation] present in test data point [True]
```

```
91 Text feature [clinical] present in test data point [True]
93 Text feature [transformation] present in test data point [True]
Out of the top  100  features  41 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [72]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index
]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_p
oint_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0486 0.1125 0.02   0.0497 0.0463 0.0356 0.6762 0.005  0.006 ]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [activating] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [treatment] present in test data point [True]
7 Text feature [activated] present in test data point [True]
9 Text feature [function] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
15 Text feature [erk] present in test data point [True]
16 Text feature [growth] present in test data point [True]
17 Text feature [loss] present in test data point [True]
18 Text feature [signaling] present in test data point [True]
20 Text feature [oncogenic] present in test data point [True]
24 Text feature [functional] present in test data point [True]
25 Text feature [receptor] present in test data point [True]
27 Text feature [akt] present in test data point [True]
28 Text feature [kinases] present in test data point [True]
29 Text feature [cells] present in test data point [True]
30 Text feature [transforming] present in test data point [True]
35 Text feature [activate] present in test data point [True]
37 Text feature [therapeutic] present in test data point [True]
40 Text feature [patients] present in test data point [True]
41 Text feature [treated] present in test data point [True]
43 Text feature [mitogen] present in test data point [True]
46 Text feature [oncogene] present in test data point [True]
48 Text feature [lines] present in test data point [True]
49 Text feature [constitutively] present in test data point [True]
51 Text feature [mek] present in test data point [True]
52 Text feature [expression] present in test data point [True]
53 Text feature [expressing] present in test data point [True]
55 Text feature [protein] present in test data point [True]
56 Text feature [3t3] present in test data point [True]
60 Text feature [downstream] present in test data point [True]
61 Text feature [ligand] present in test data point [True]
63 Text feature [drug] present in test data point [True]
65 Text feature [cell] present in test data point [True]
```

```
66 Text feature [patient] present in test data point [True]
70 Text feature [phospho] present in test data point [True]
71 Text feature [phosphorylated] present in test data point [True]
73 Text feature [il] present in test data point [True]
74 Text feature [factor] present in test data point [True]
77 Text feature [variant] present in test data point [True]
81 Text feature [dna] present in test data point [True]
84 Text feature [transform] present in test data point [True]
85 Text feature [extracellular] present in test data point [True]
86 Text feature [starved] present in test data point [True]
88 Text feature [proliferation] present in test data point [True]
89 Text feature [survival] present in test data point [True]
90 Text feature [serum] present in test data point [True]
93 Text feature [transformation] present in test data point [True]
97 Text feature [independence] present in test data point [True]
99 Text feature [respond] present in test data point [True]
Out of the top  100  features  55 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [73]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_spli
t=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_de
crease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_star
t=False,
# class_weight=None)

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

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

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
```

```
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-
1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''


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

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

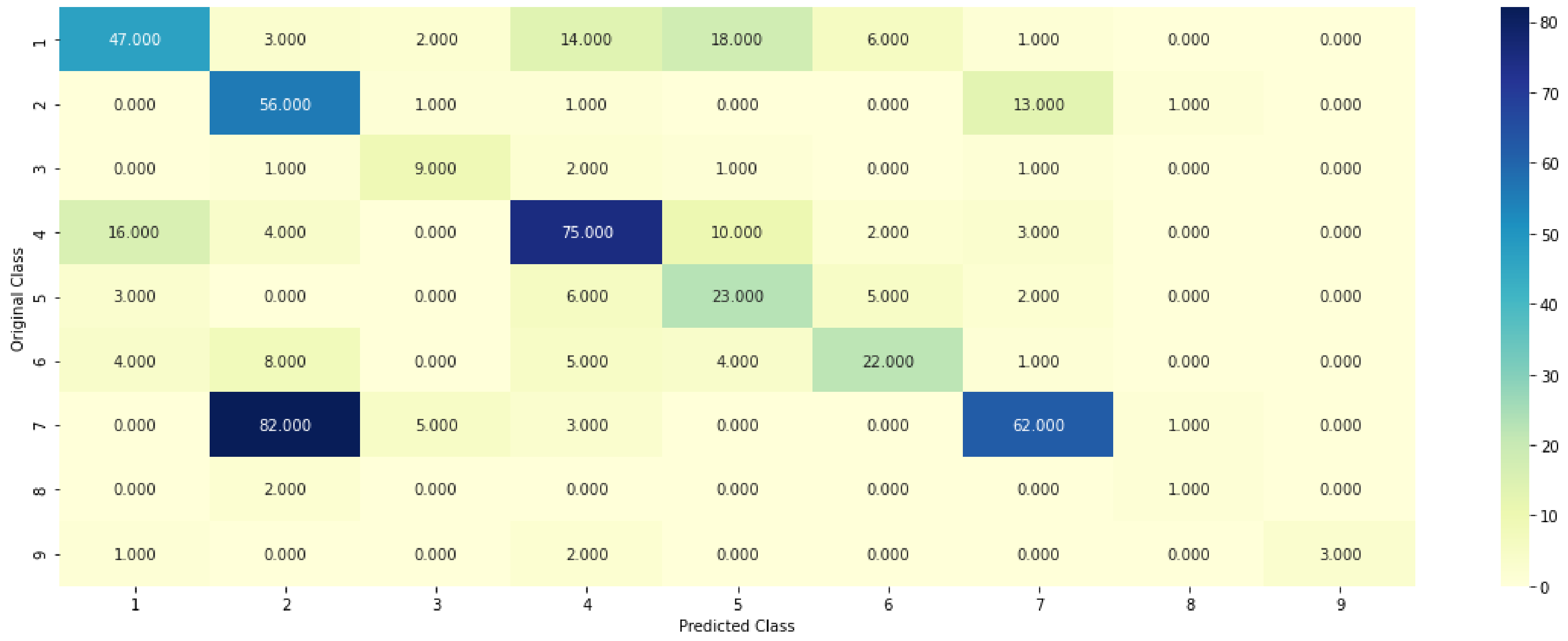
```
for n_estimators = 10 and max depth =  2
Log Loss : 1.9861677450954627
for n_estimators = 10 and max depth =  3
Log Loss : 1.5435362939196324
for n_estimators = 10 and max depth =  5
Log Loss : 1.4469034978342659
for n_estimators = 10 and max depth =  10
Log Loss : 1.6132637947462776
for n_estimators = 50 and max depth =  2
Log Loss : 1.6004693454582304
for n_estimators = 50 and max depth =  3
Log Loss : 1.3614347051222064
for n_estimators = 50 and max depth =  5
Log Loss : 1.3081183334077289
for n_estimators = 50 and max depth =  10
Log Loss : 1.5830274330364542
for n_estimators = 100 and max depth =  2
Log Loss : 1.4976846855755506
for n_estimators = 100 and max depth =  3
Log Loss : 1.4331341767337324
for n_estimators = 100 and max depth =  5
Log Loss : 1.2500037185500072
for n_estimators = 100 and max depth =  10
Log Loss : 1.6364236736649644
for n_estimators = 200 and max depth =  2
Log Loss : 1.4868282085215685
for n_estimators = 200 and max depth =  3
Log Loss : 1.4073936352601721
for n_estimators = 200 and max depth =  5
Log Loss : 1.2823744982928649
for n_estimators = 200 and max depth =  10
Log Loss : 1.6522884335788173
for n_estimators = 500 and max depth =  2
Log Loss : 1.5485391457158015
for n_estimators = 500 and max depth =  3
Log Loss : 1.4436126700101428
for n_estimators = 500 and max depth =  5
Log Loss : 1.3055032547490981
for n_estimators = 500 and max depth =  10
Log Loss : 1.6589038057131122
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5380263905561016
for n_estimators = 1000 and max depth =  3
```

```
Log Loss : 1.4575548295767202
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2937629123104881
for n_estimators = 1000 and max depth =  10
Log Loss : 1.633210876353533
For values of best alpha =  100 The train log loss is: 0.070037434300006942
For values of best alpha =  100 The cross validation log loss is: 1.2500037185500075
For values of best alpha =  100 The test log loss is: 1.2708388825255514
```

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

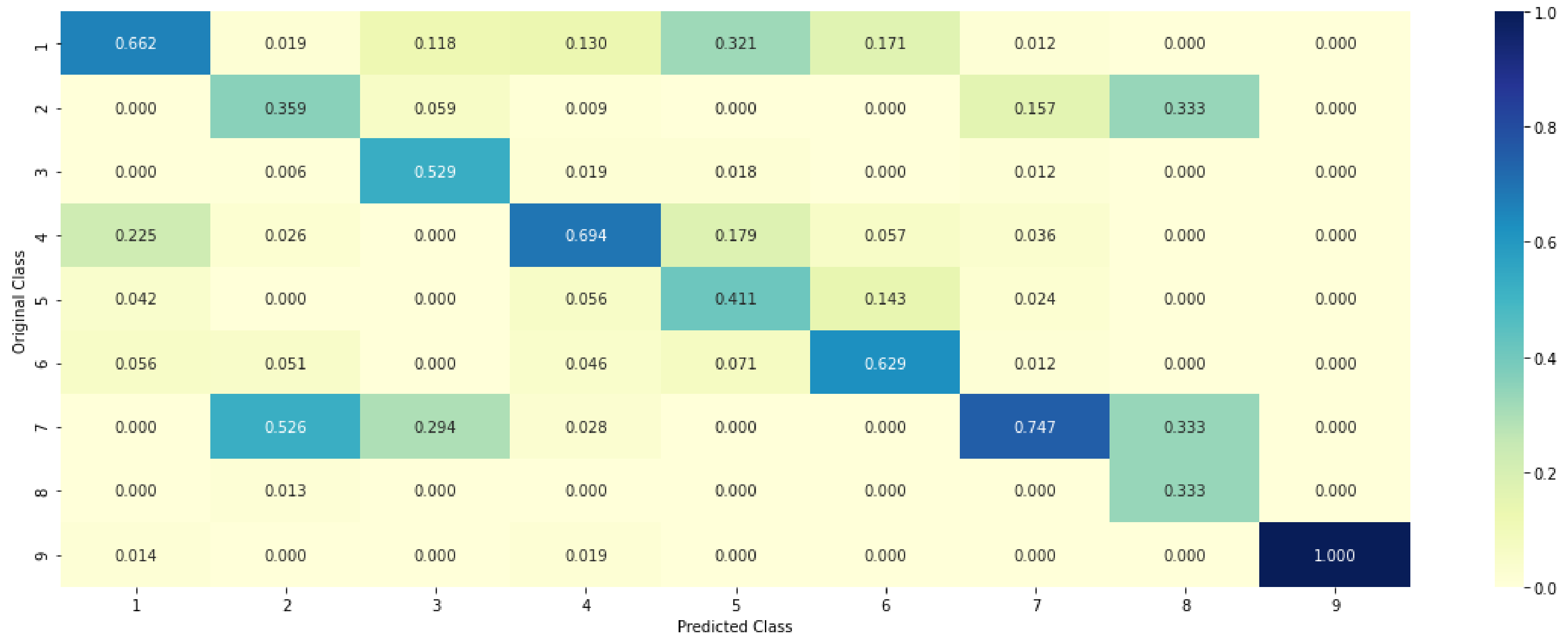
In [74]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_spli
t=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_de
crease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_star
t=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.2500037185500072
Number of mis-classified points : 0.4398496240601504
-------------------- Confusion matrix --------------------
```



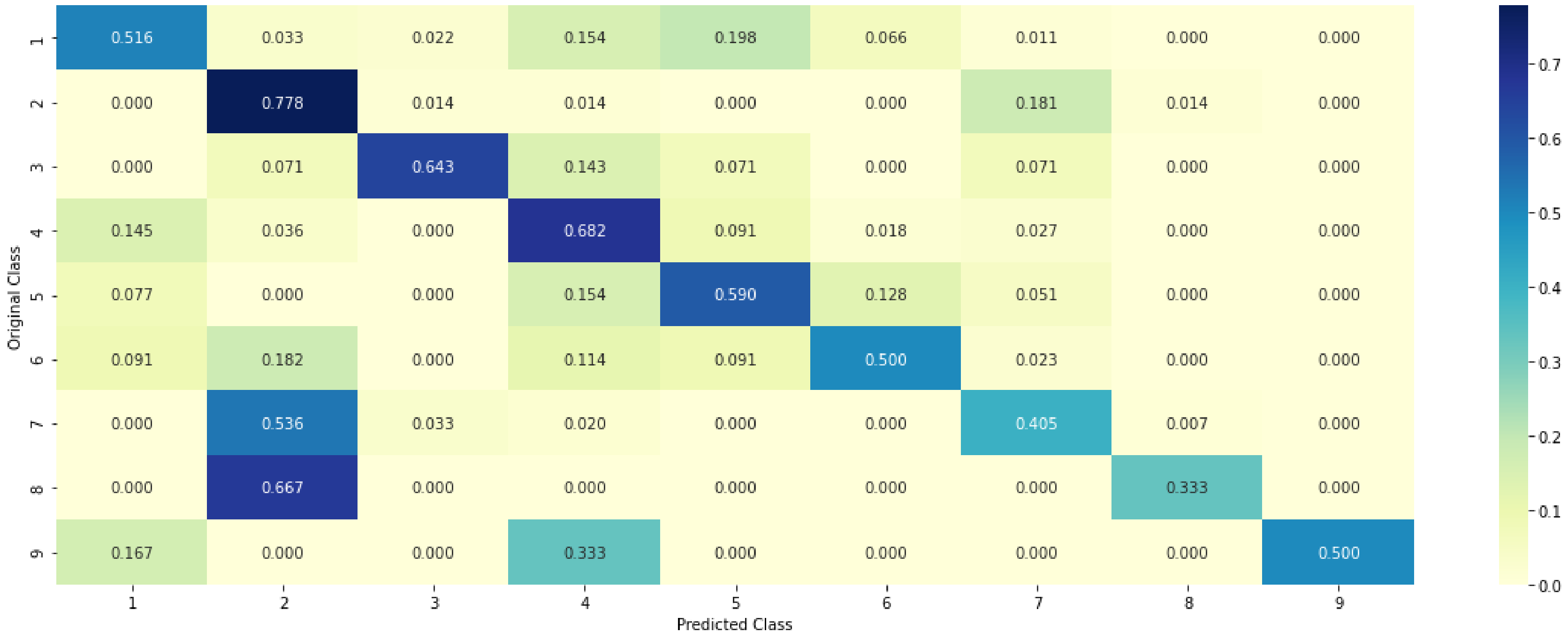
Original \ Predicted	1	2	3	4	5	6	7	8	9
1	47.000	3.000	2.000	14.000	18.000	6.000	1.000	0.000	0.000
2	0.000	56.000	1.000	1.000	0.000	0.000	13.000	1.000	0.000
3	0.000	1.000	9.000	2.000	1.000	0.000	1.000	0.000	0.000
4	16.000	4.000	0.000	75.000	10.000	2.000	3.000	0.000	0.000
5	3.000	0.000	0.000	6.000	23.000	5.000	2.000	0.000	0.000
6	4.000	8.000	0.000	5.000	4.000	22.000	1.000	0.000	0.000
7	0.000	82.000	5.000	3.000	0.000	0.000	62.000	1.000	0.000
8	0.000	2.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
9	1.000	0.000	0.000	2.000	0.000	0.000	0.000	0.000	3.000

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



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.662	0.019	0.118	0.130	0.321	0.171	0.012	0.000	0.000
2	0.000	0.359	0.059	0.009	0.000	0.000	0.157	0.333	0.000
3	0.000	0.006	0.529	0.019	0.018	0.000	0.012	0.000	0.000
4	0.225	0.026	0.000	0.694	0.179	0.057	0.036	0.000	0.000
5	0.042	0.000	0.000	0.056	0.411	0.143	0.024	0.000	0.000
6	0.056	0.051	0.000	0.046	0.071	0.629	0.012	0.000	0.000
7	0.000	0.526	0.294	0.028	0.000	0.000	0.747	0.333	0.000
8	0.000	0.013	0.000	0.000	0.000	0.000	0.000	0.333	0.000
9	0.014	0.000	0.000	0.019	0.000	0.000	0.000	0.000	1.000

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



4.7 Stack the models

In [75]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SG
DClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, t
ol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power
_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/gen
erated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_spli
t=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_de
crease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_star
t=False,
# class_weight=None)

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

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

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


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

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

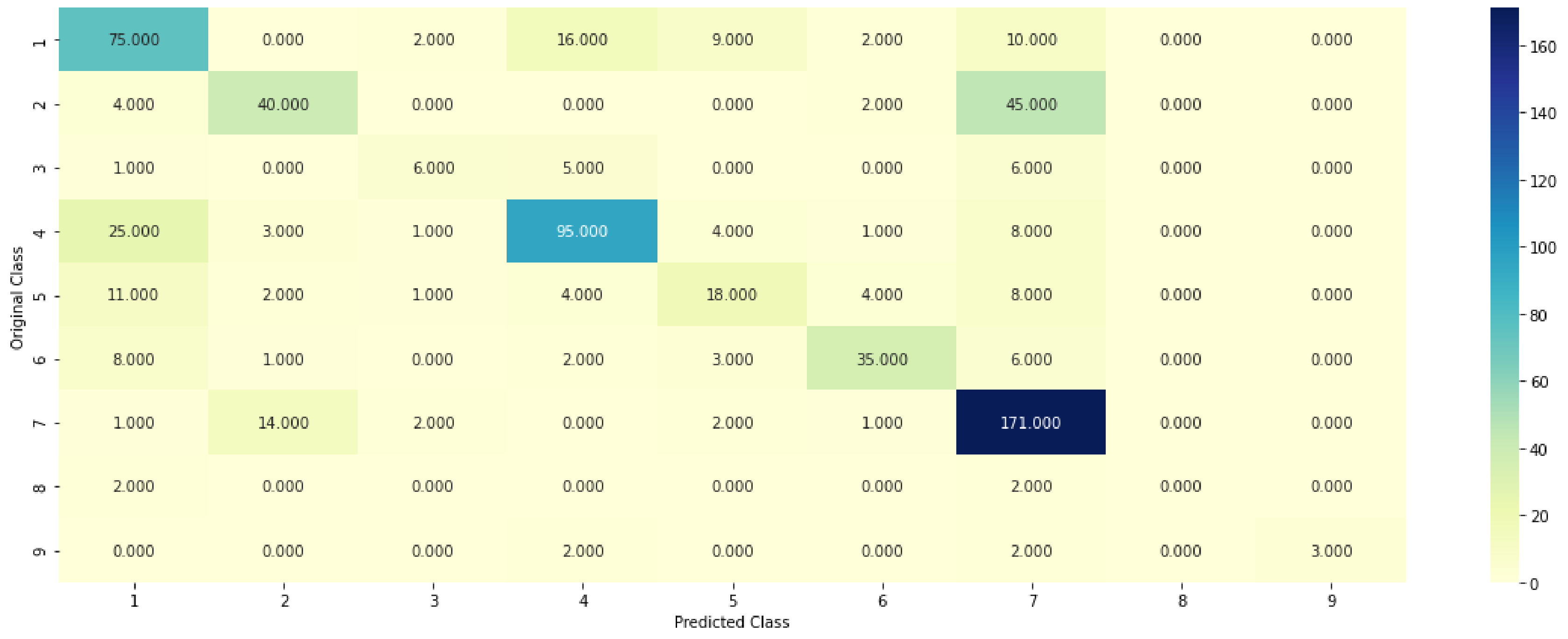
```
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True
)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict
_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.70
Naive Bayes : Log Loss: 1.27
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.818
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.724
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.334
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.180
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.469
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.825
```

4.7.2 testing the model with the best hyper parameters

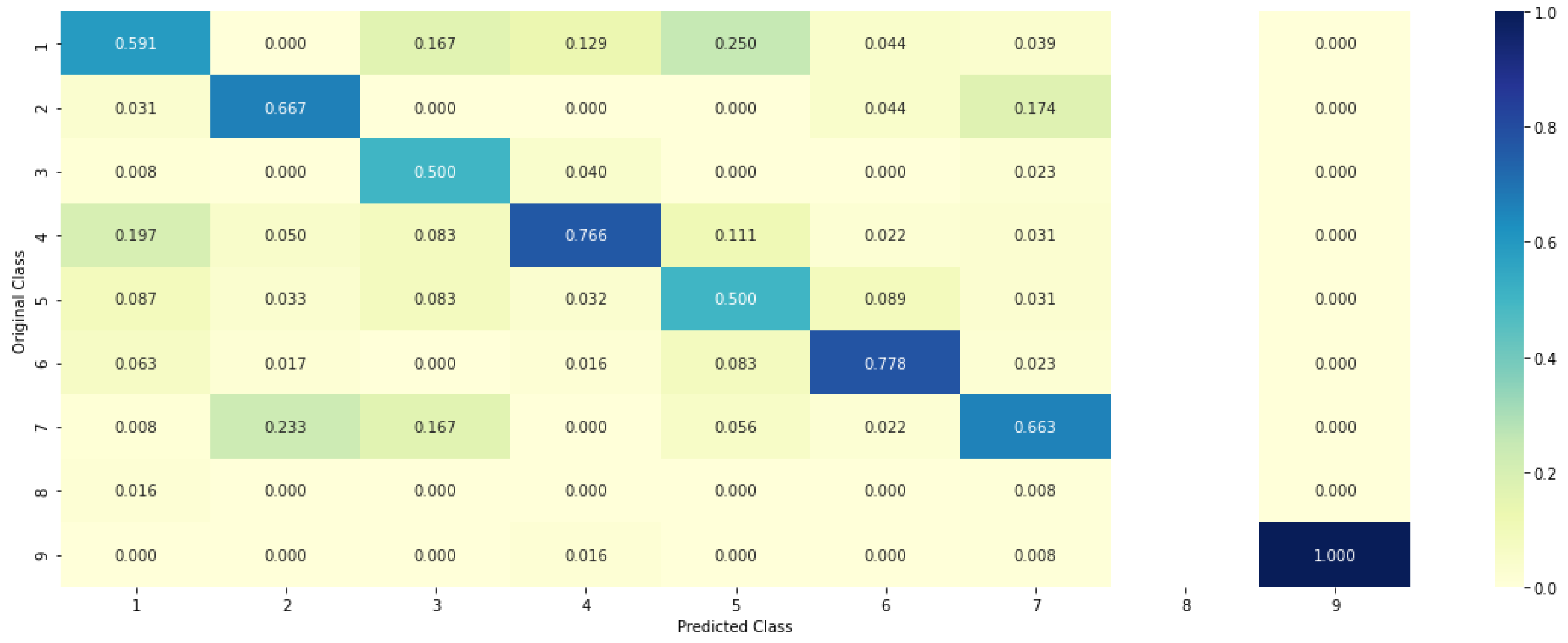
In [76]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.49601922782285474
Log loss (CV) on the stacking classifier : 1.179513082089586
Log loss (test) on the stacking classifier : 1.129657222184161
Number of missclassified point : 0.33383458646616543
-------------------- Confusion matrix --------------------
```

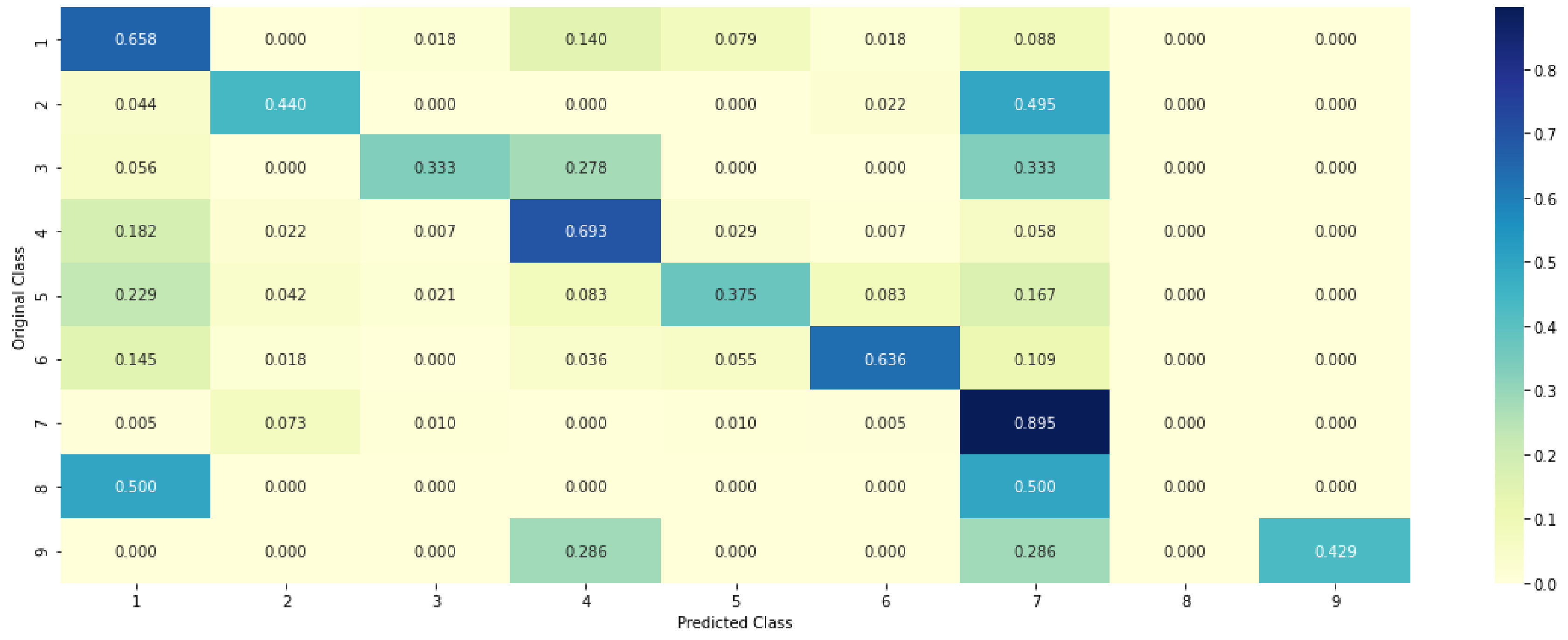


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	75.000	0.000	2.000	16.000	9.000	2.000	10.000	0.000	0.000
2	4.000	40.000	0.000	0.000	0.000	2.000	45.000	0.000	0.000
3	1.000	0.000	6.000	5.000	0.000	0.000	6.000	0.000	0.000
4	25.000	3.000	1.000	95.000	4.000	1.000	8.000	0.000	0.000
5	11.000	2.000	1.000	4.000	18.000	4.000	8.000	0.000	0.000
6	8.000	1.000	0.000	2.000	3.000	35.000	6.000	0.000	0.000
7	1.000	14.000	2.000	0.000	2.000	1.000	171.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000
9	0.000	0.000	0.000	2.000	0.000	0.000	2.000	0.000	3.000

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



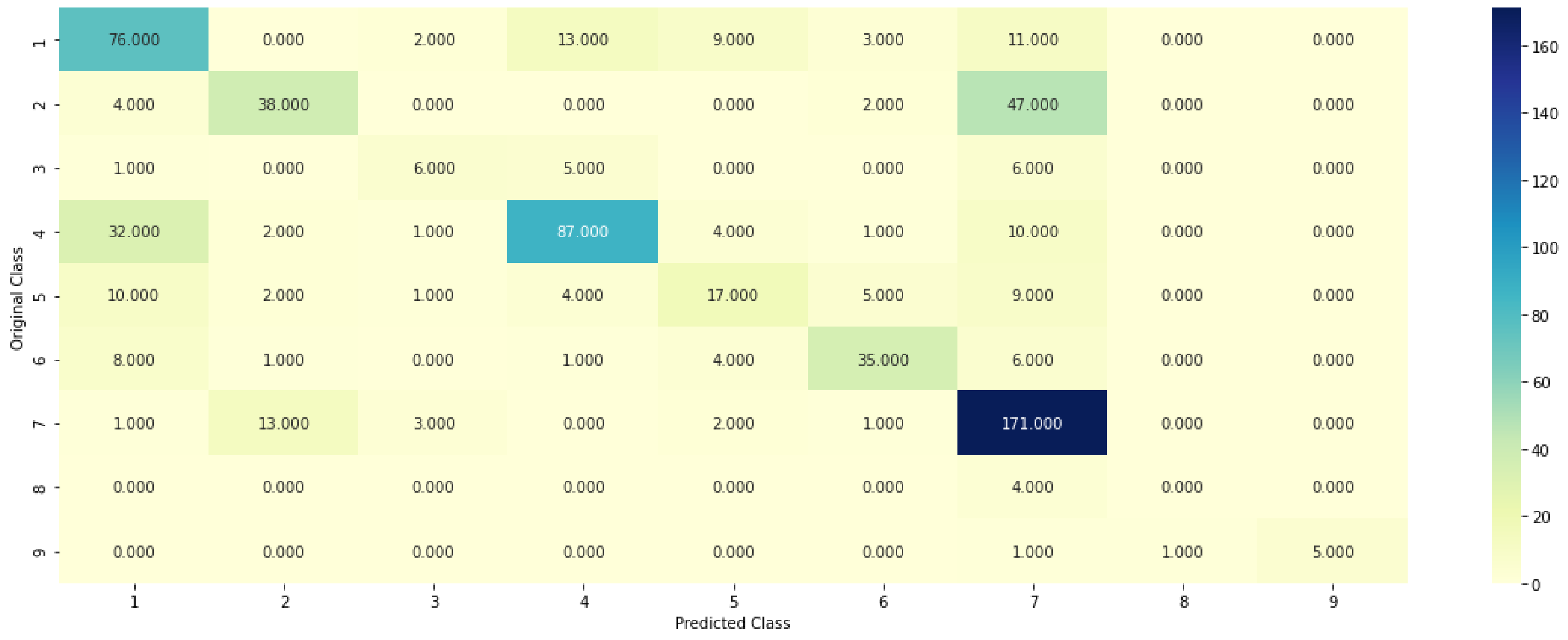
```
-------------------- Recall matrix (Row sum=1) --------------------
```



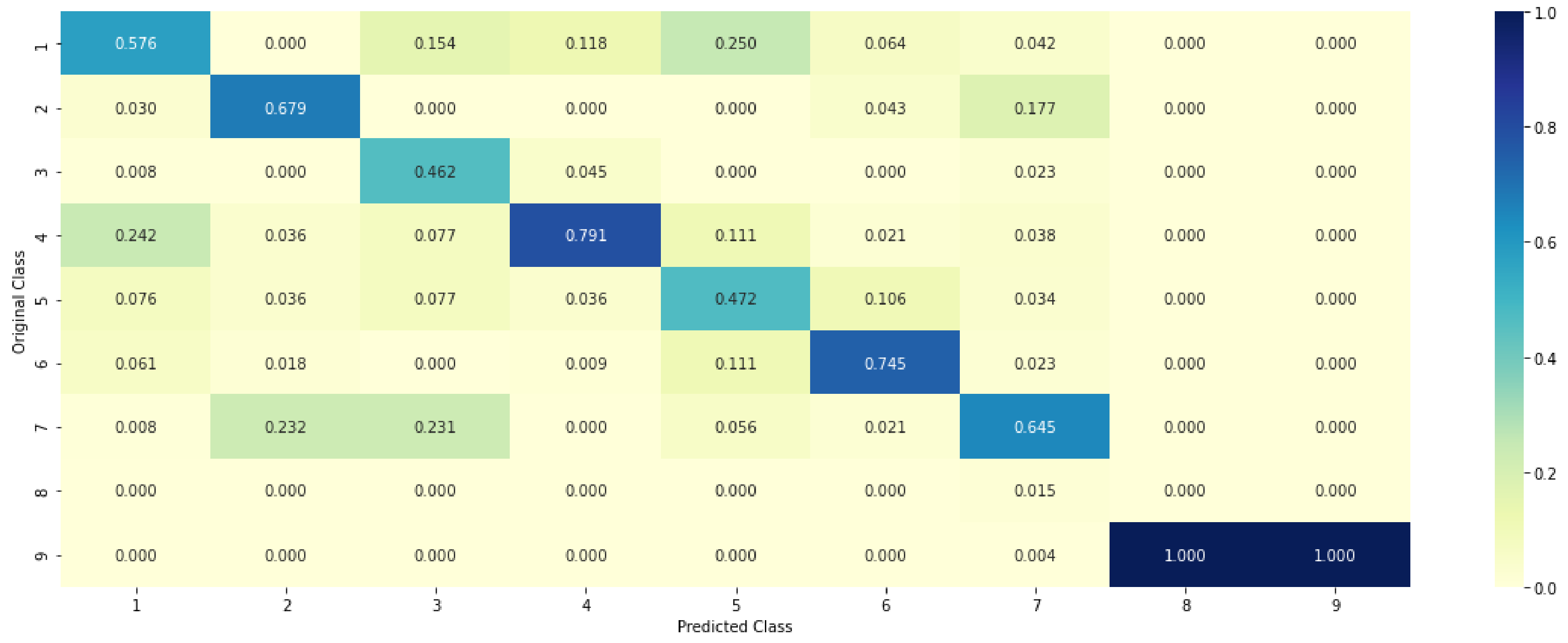
4.7.3 Maximum Voting classifier

In [77]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding
)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_
y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.8625166091555341
Log loss (CV) on the VotingClassifier : 1.181574347676941
Log loss (test) on the VotingClassifier : 1.1582256363681527
Number of missclassified point : 0.3458646616541353
-------------------- Confusion matrix --------------------
```



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



	1	2	3	4	5	6	7	8	9
1	0.576	0.000	0.154	0.118	0.250	0.064	0.042	0.000	0.000
2	0.030	0.679	0.000	0.000	0.000	0.043	0.177	0.000	0.000
3	0.008	0.000	0.462	0.045	0.000	0.000	0.023	0.000	0.000
4	0.242	0.036	0.077	0.791	0.111	0.021	0.038	0.000	0.000
5	0.076	0.036	0.077	0.036	0.472	0.106	0.034	0.000	0.000
6	0.061	0.018	0.000	0.009	0.111	0.745	0.023	0.000	0.000
7	0.008	0.232	0.231	0.000	0.056	0.021	0.645	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.015	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.004	1.000	1.000

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