Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
    - training_variants (ID , Gene, Variations, Class)
    - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

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

knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6).
...

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- Interpretability
- Class probabilities are needed.

- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

3. Exploratory Data Analysis

In [4]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import RandomForestClassifier
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
```

```python
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.cross_validation import StratifiedKfold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import nltk
nltk.download('stopwords')
```

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


True
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [5]:
```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

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

training/training_variants is a comma separated file containing the description of the genetic mutations used for training.
Fields are

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

3.1.2. Reading Text Data

In [6]:
```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID",
"TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

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

3.1.3. Preprocessing of text

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 : 183.39537900000002 seconds
```

In [9]:

```
#merging both gene_variations and text data based on ID
```

```
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

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

```
In [10]: result[result.isnull().any(axis=1)]
```

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

```
In [11]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [12]: result[result['ID']==1109]
```

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

3.1.4. Test, Train and Cross Validation Split

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

In [13]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

In [14]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

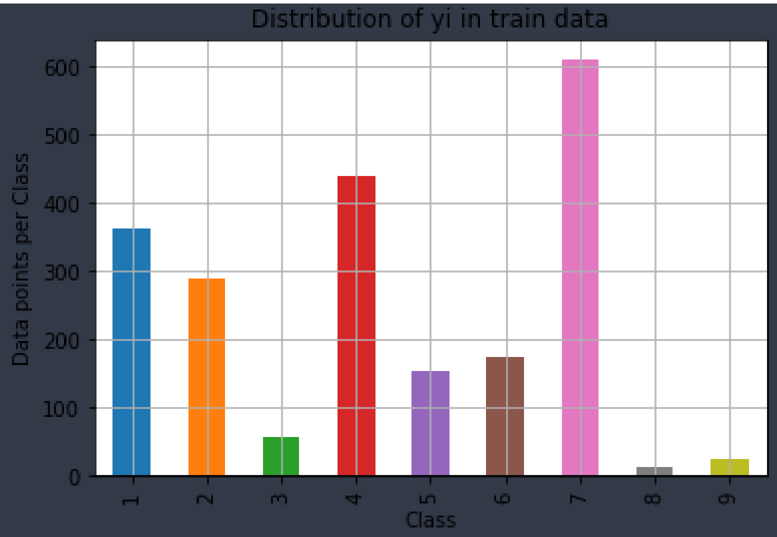
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

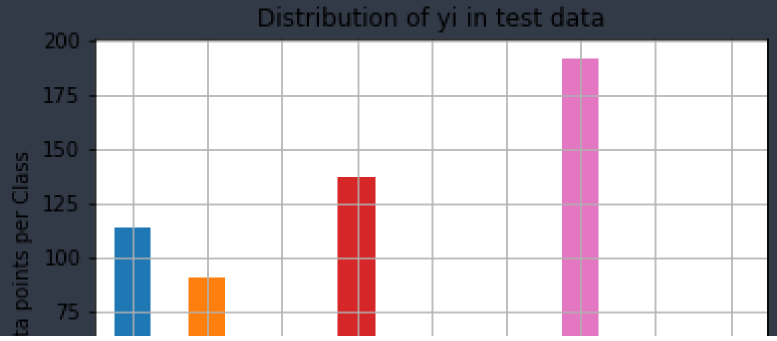
In [15]:
```
# it returns a dict, keys as class labels and values as the number of data points in tha
t class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
```

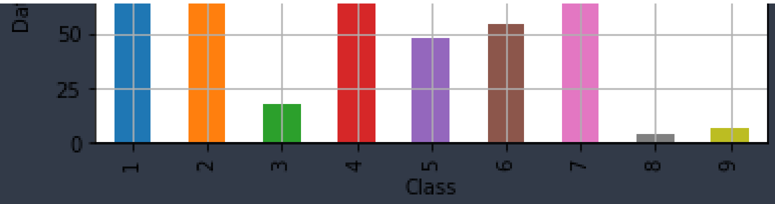
```
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

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

```
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],
'(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

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

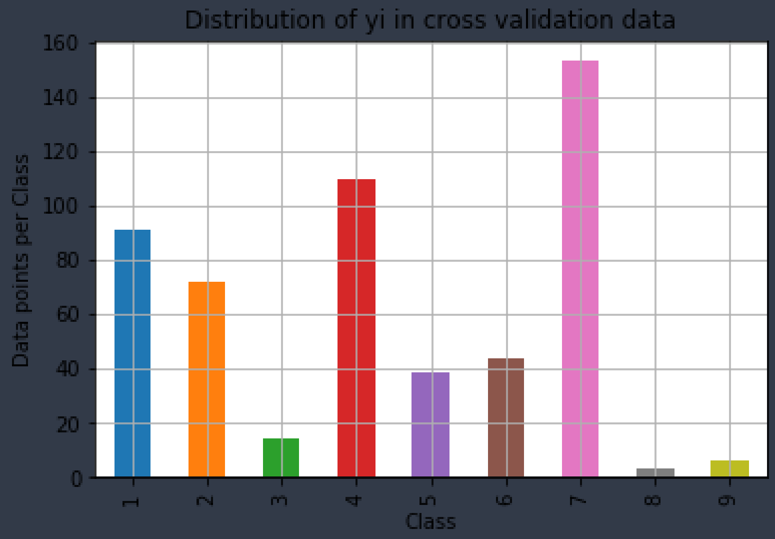


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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





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



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

3.2 Prediction using a 'Random' Model

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

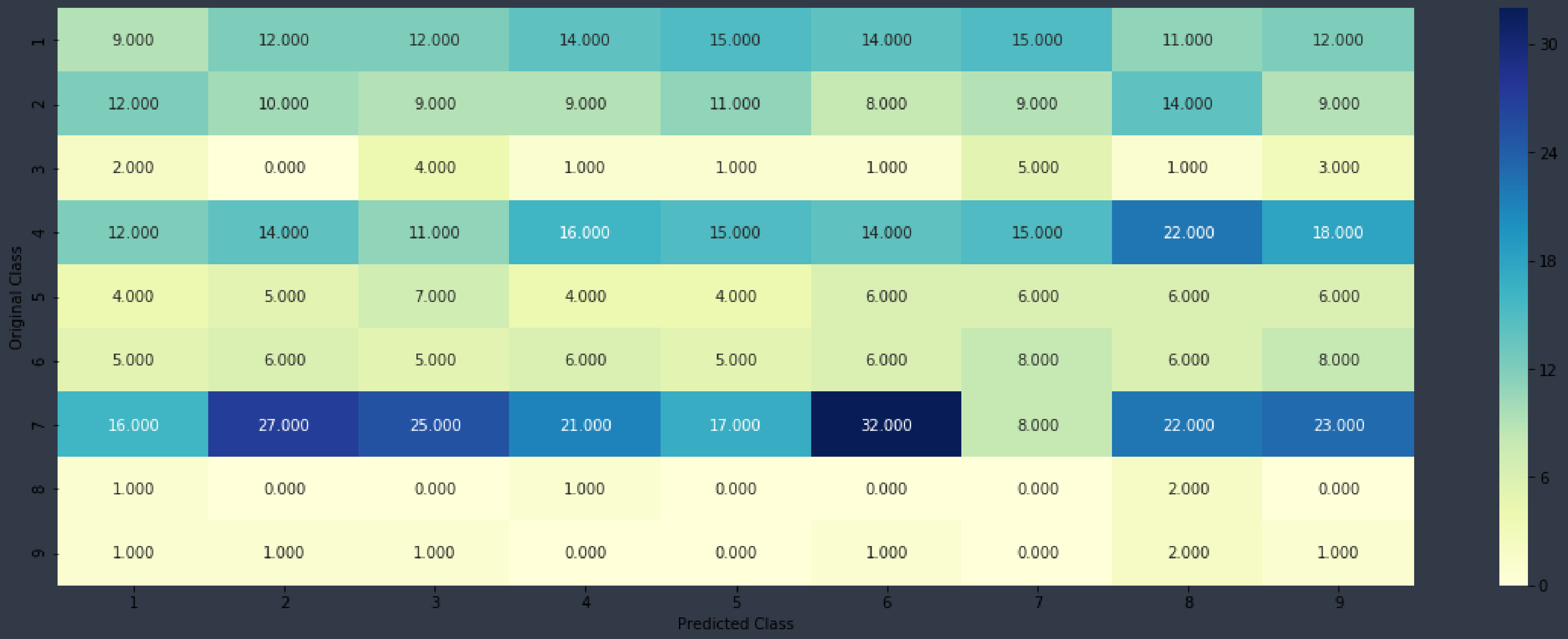
In [16]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predict
ed class j

    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 tw
o 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 tw
o 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 [17]:

```
# 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))
```

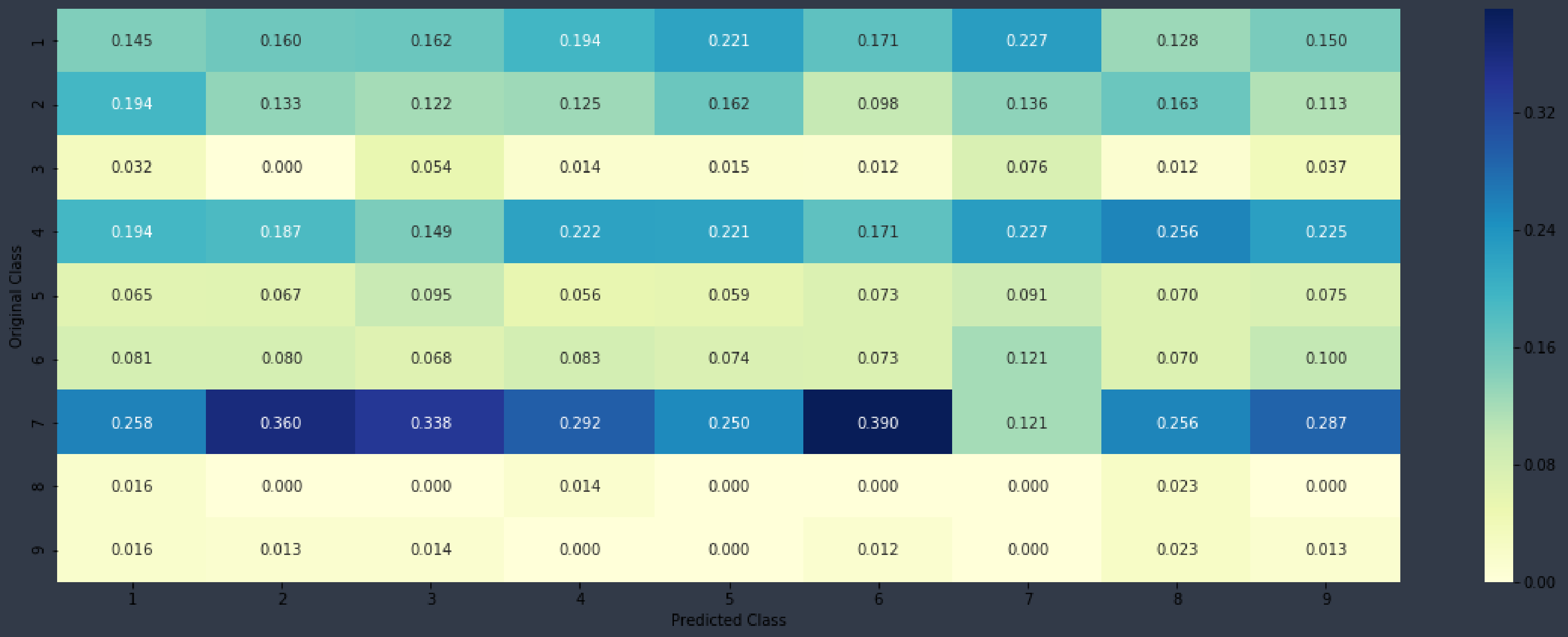
```
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1
e-15))

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 2.551212409664783
Log loss on Test Data using Random Model 2.56965616863993
-------------------- Confusion matrix --------------------
```



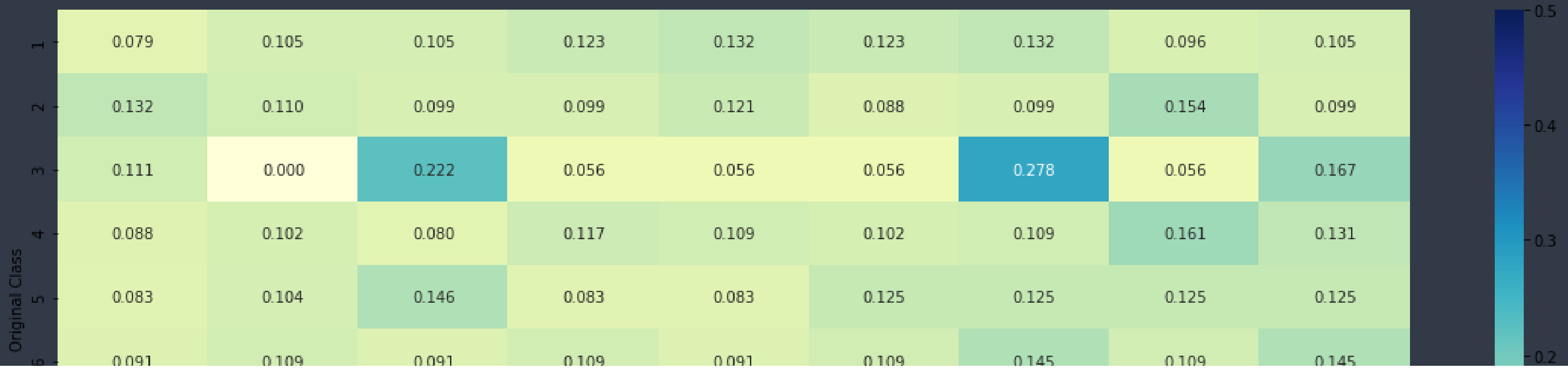
```
-------------------- Precision matrix (Column Sum=1) --------------------
```

Precision matrix (Column Sum=1)

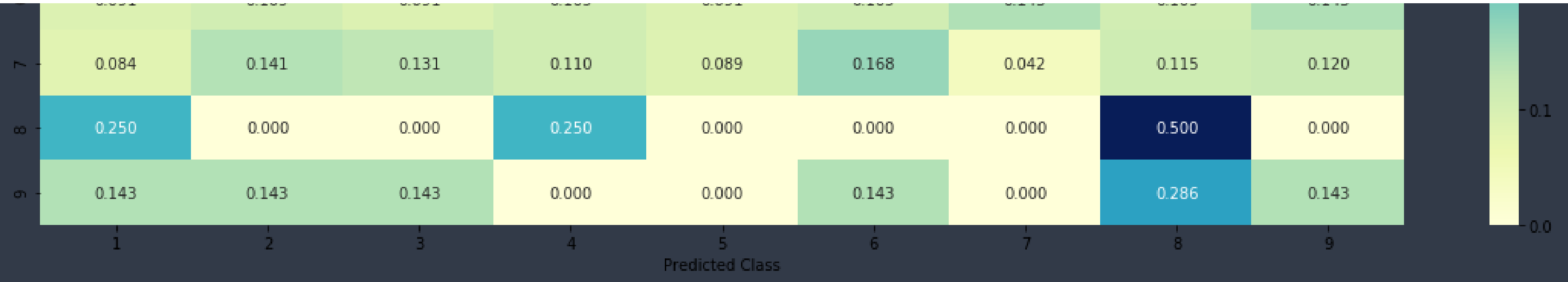


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.145	0.160	0.162	0.194	0.221	0.171	0.227	0.128	0.150
2	0.194	0.133	0.122	0.125	0.162	0.098	0.136	0.163	0.113
3	0.032	0.000	0.054	0.014	0.015	0.012	0.076	0.012	0.037
4	0.194	0.187	0.149	0.222	0.221	0.171	0.227	0.256	0.225
5	0.065	0.067	0.095	0.056	0.059	0.073	0.091	0.070	0.075
6	0.081	0.080	0.068	0.083	0.074	0.073	0.121	0.070	0.100
7	0.258	0.360	0.338	0.292	0.250	0.390	0.121	0.256	0.287
8	0.016	0.000	0.000	0.014	0.000	0.000	0.000	0.023	0.000
9	0.016	0.013	0.014	0.000	0.000	0.012	0.000	0.023	0.013

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



Original									
1	0.079	0.105	0.105	0.123	0.132	0.123	0.132	0.096	0.105
2	0.132	0.110	0.099	0.099	0.121	0.088	0.099	0.154	0.099
3	0.111	0.000	0.222	0.056	0.056	0.056	0.278	0.056	0.167
4	0.088	0.102	0.080	0.117	0.109	0.102	0.109	0.161	0.131
5	0.083	0.104	0.146	0.083	0.083	0.125	0.125	0.125	0.125
6	0.091	0.109	0.091	0.109	0.091	0.109	0.145	0.109	0.145



3.3 Univariate Analysis

In [18]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data
 dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*
alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------
# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()
    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/
variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in who
le data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticu
lar 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 particula
r 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, 0.06818181818181818177, 0.1
3636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2
7040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.05
1020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818181818
177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.
078787878787878782, 0.13939393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060
606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.
46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.0
62893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07
2847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066
225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0
73333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.06
6666666666666666, 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 t
he train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [19]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
```

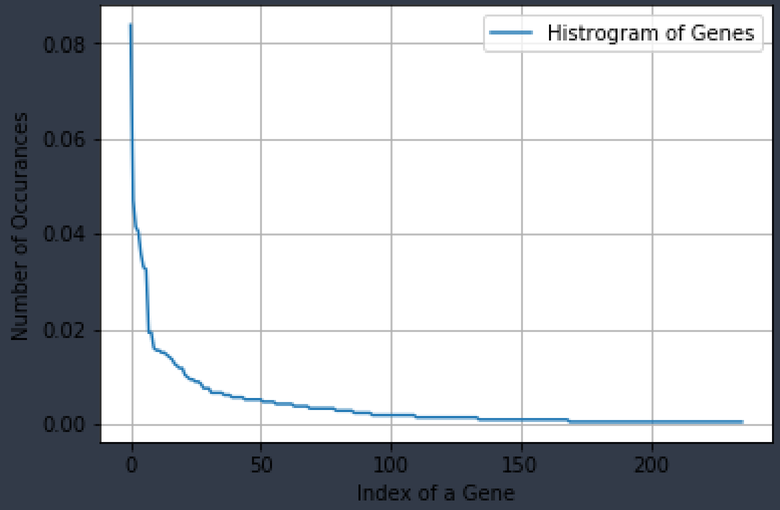
```python
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 236
BRCA1    178
TP53     100
PTEN      88
EGFR      86
BRCA2     76
BRAF      70
KIT       69
ERBB2     41
ALK       41
TSC2      34
Name: Gene, dtype: int64
```

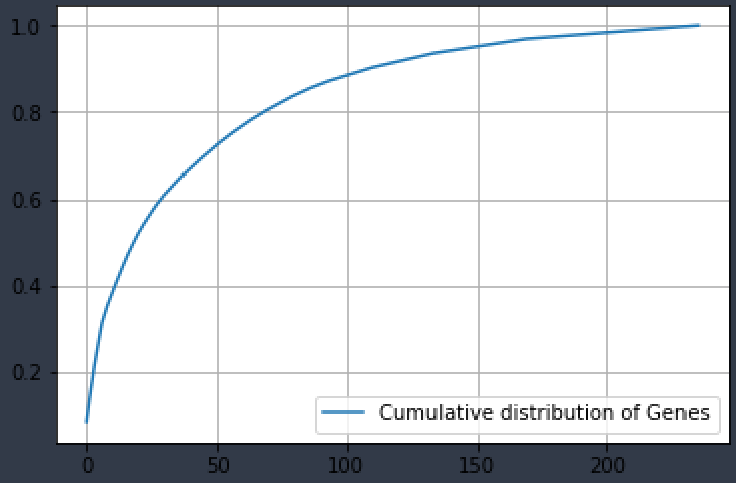
In [20]:
```python
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the tra
in data, and they are distibuted as follows",)
```

```
Ans: There are 236 different categories of genes in the train data, and they are distibuted as follows
```

In [21]:
```python
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [22]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [23]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
```

```
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [24]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding metho
d. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [25]:
```
# 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 [26]:
```
train_df['Gene'].head()
```

```
564     SMAD3
1400    FGFR3
257      EGFR
1698     PMS2
1933      SMO
Name: Gene, dtype: object
```

In [27]:
```
gene_vectorizer.get_feature_names()
```

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
                'araf',
                'arid1a',
                'arid1b',
                'arid5b',
                'asxl1',
                'atm',
                'atr',
                'atrx',
                'aurka',
                'aurkb',
                'axin1',
                'axl',
                'b2m',
                'bap1',
                'bard1',
                'bcl10',
                'bcl2l11',
                'bcor',
                'braf',
                'brca1',
                'brca2',
                'brd4',
                'brip1',
                'btk',
                'card11',
                'carm1',
                'casp8',
                'cbl',
                'ccnd1',
                'ccnd3',
                'ccne1',
                'cdh1',
                'cdk12',
                'cdk4',
                'cdk6',
                'cdk8',
                'cdkn1a',
                'cdkn1b',
                'cdkn2a',
                'cdkn2b',
                'cdkn2c',
                'cebpa',
                'chek2',
                'cic',
                'crebbp',
                'ctcf',
                'ctla4',
                'ctnnb1',
                'ddr2',
                'dicer1',
                'dnmt3a',
                'dnmt3b',
                'dusp4',
                'egfr',
                'eif1ax',
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'errfi1',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fam58a',
                'fanca',
                'fancc',
                'fat1',
                'fbxw7',
                'fgf3',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxo1',
                'foxp1',
                'gata3',
                'gli1',
                'gna11',
                'gnas',
                'h3f3a',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikbke',
                'il7r',
                'jak1',
                'jak2',
                'jun',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
                'kmt2a',
                'kmt2b',
                'kmt2c',
                'kmt2d',
                'knstrn',
                'kras',
                'lats1',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'med12',
                'mef2b',
                'met',
                'mga',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'myod1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'notch2',
                'npm1',
                'nras',
                'nsd1',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pim1',
                'pms1',
                'pms2',
                'pole',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
                'rab35',
                'rac1',
                'rad21',
                'rad50',
                'rad51c',
                'rad51d',
                'rad54l',
                'raf1',
                'rasa1',
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rictor',
                'rit1',
                'rnf43',
                'ros1',
                'runx1',
                'rxra',
                'setd2',
                'sf3b1',
                'shoc2',
                'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'yap1']
```

In [28]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding metho
d. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

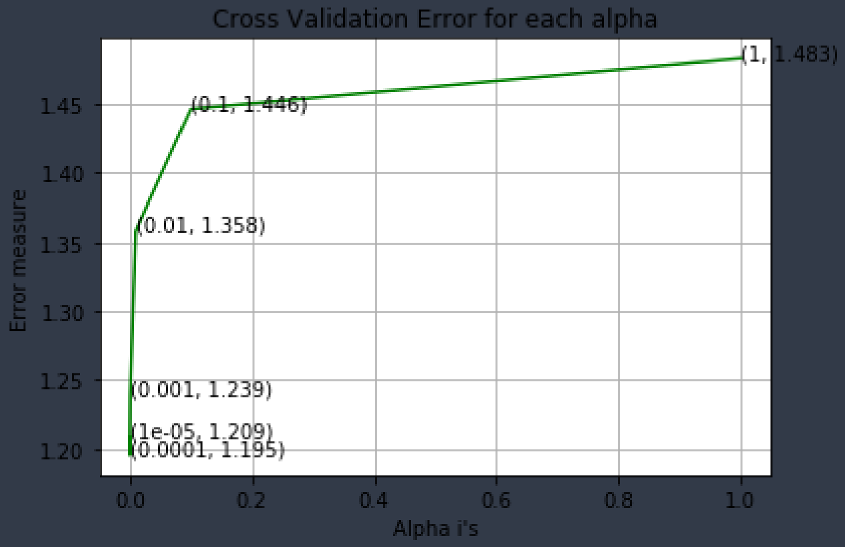
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 236)
```

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

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

In [29]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t Descent.
# predict(X)    Predict class labels for samples in X.

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



cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labe
ls=clf.classes_, eps=1e-15))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.208702688586516
For values of alpha =  0.0001 The log loss is: 1.1953639079706706
For values of alpha =  0.001 The log loss is: 1.2390173936800735
For values of alpha =  0.01 The log loss is: 1.3582071666325053
For values of alpha =  0.1 The log loss is: 1.4463552970448768
For values of alpha =  1 The log loss is: 1.4832238937041724
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0015663922665903
For values of best alpha =  0.0001 The cross validation log loss is: 1.1953639079706706
For values of best alpha =  0.0001 The test log loss is: 1.1652301439734927
```

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 [30]:
```python
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  236  genes in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 520 out of  532 : 97.74436090225564
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [31]:
```python
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1918
```

```
Truncating_Mutations    63
Deletion                51
Amplification           47
Fusions                 20
Overexpression           5
Q61L                     3
Q61H                     3
G12V                     3
Q61R                     3
TMPRSS2-ETV1_Fusion      2
Name: Variation, dtype: int64
```

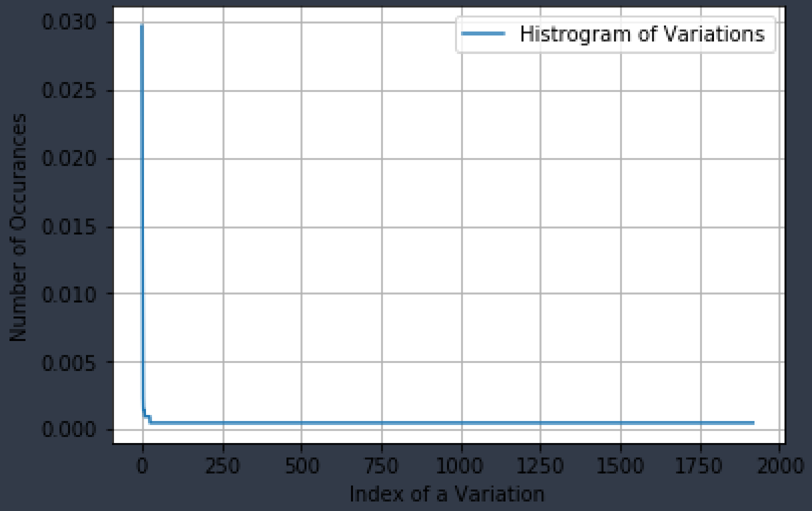
In [32]:

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

```
Ans: There are 1918 different categories of variations in the train data, and they are distibuted as follows
```

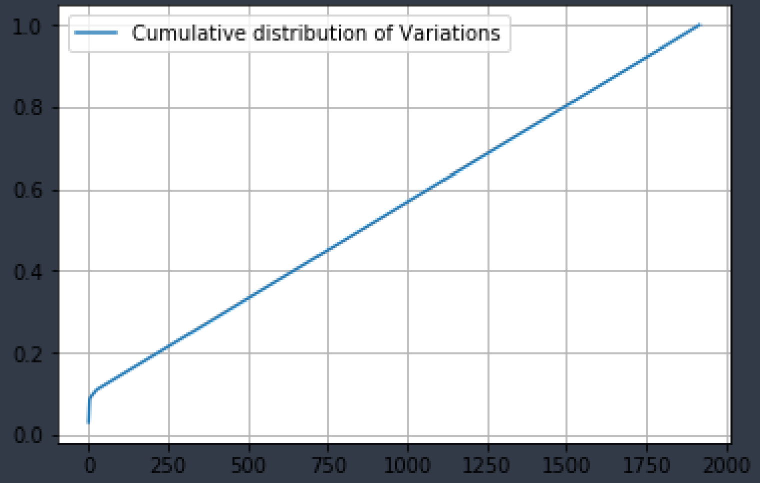
In [33]:

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



In [34]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02966102 0.05367232 0.07580038 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [35]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", tra
in_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 [36]:
```
print("train_variation_feature_responseCoding is a converted feature using the response
 coding method. The shape of Variation feature:", train_variation_feature_responseCoding
.shape)
```

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

In [37]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Vari
ation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'
])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [38]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot enc
oding method. The shape of Variation feature:", train_variation_feature_onehotCoding.sha
pe)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation featur
e: (2124, 1944)
```

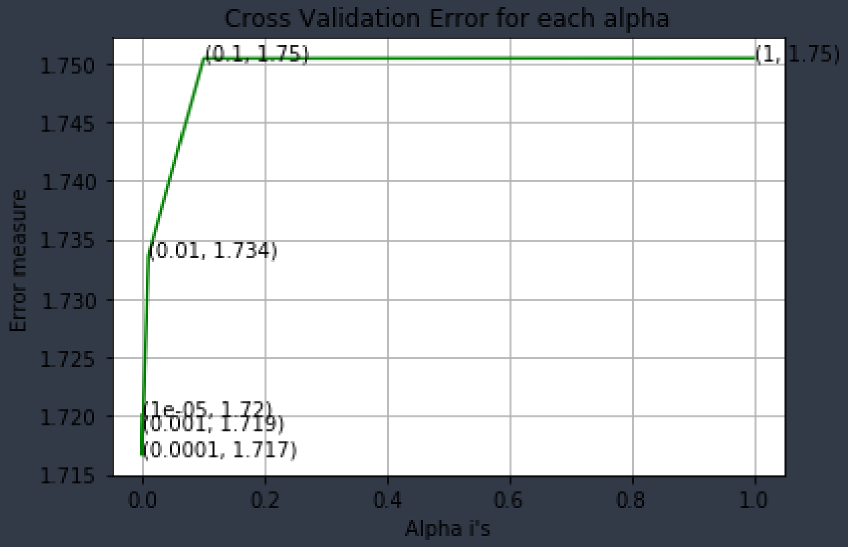
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [39]:
```
alpha = [10 ** x for x in range(-5, 1)]
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t 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",cv=None)
    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, labe
ls=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",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.720094498695089
For values of alpha =  0.0001 The log loss is: 1.716645969883596
For values of alpha =  0.001 The log loss is: 1.7189894501801914
For values of alpha =  0.01 The log loss is: 1.7335778163205122
For values of alpha =  0.1 The log loss is: 1.750425928585354
For values of alpha =  1 The log loss is: 1.750451737622075
```



```
For values of best alpha =  0.0001 The train log loss is: 0.754021684361103
For values of best alpha =  0.0001 The cross validation log loss is: 1.716645969883596
For values of best alpha =  0.0001 The test log loss is: 1.7044771255131872
```

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 [40]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
```

```
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverag
e/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cover
age/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1918  genes in test and cross validation data sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 48 out of  532 : 9.022556390977442
```

3.2.3 Univariate Analysis on Text Feature

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

In [41]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

```
In [42]: 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(wor
         d,0)+90)))
                     text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'
         ].split()))
                     row_index += 1
             return text_feature_responseCoding
```

```
In [43]: # building a CountVectorizer with all the words that occured minimum 3 times in train da
         ta
         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*numbe
         r 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 o
         ccured
         text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
```

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

```
Total number of unique words in train data : 53645
```

In [44]:
```python
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [45]:
```python
#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 [46]:
```
# 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_feat
ure_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature
_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_respo
nseCoding.sum(axis=1)).T
```

In [47]:
```
# 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 [48]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=T
rue))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [49]:
```
# Number of words for a given frequency.
```

```python
print(Counter(sorted_text_occur))
```

```
Counter({3: 5914, 4: 3363, 6: 3015, 5: 2856, 7: 2039, 8: 1879, 9: 1660, 12: 1579, 10: 1472, 15: 1064, 11: 933, 17: 856, 14:
832, 13: 803, 18: 715, 16: 700, 20: 599, 24: 575, 21: 512, 19: 474, 23: 453, 28: 439, 22: 405, 30: 372, 40: 370, 29: 354, 2
5: 348, 26: 336, 27: 306, 36: 294, 31: 276, 53: 275, 33: 261, 34: 261, 32: 255, 35: 246, 37: 216, 42: 211, 41: 202, 38: 197,
39: 189, 47: 185, 45: 179, 60: 179, 54: 169, 48: 168, 44: 167, 46: 165, 56: 162, 43: 159, 51: 159, 57: 155, 52: 150, 50: 14
9, 49: 144, 55: 129, 65: 119, 69: 117, 58: 115, 63: 111, 59: 109, 64: 103, 73: 103, 62: 102, 72: 102, 66: 101, 68: 97, 70: 9
5, 61: 93, 74: 93, 94: 91, 76: 89, 80: 88, 75: 87, 91: 84, 67: 78, 78: 77, 77: 72, 81: 72, 84: 72, 87: 72, 71: 71, 93: 71, 8
5: 70, 89: 68, 86: 67, 102: 67, 90: 65, 99: 65, 96: 64, 106: 64, 79: 60, 92: 59, 120: 59, 83: 58, 95: 58, 101: 58, 108: 56,
88: 55, 109: 54, 82: 53, 97: 52, 105: 52, 112: 50, 113: 50, 110: 49, 98: 48, 116: 48, 119: 47, 130: 47, 100: 46, 107: 46, 11
1: 46, 159: 46, 127: 45, 133: 45, 104: 43, 117: 43, 118: 43, 141: 43, 151: 43, 145: 42, 114: 41, 124: 40, 144: 40, 103: 39,
115: 39, 128: 38, 131: 38, 134: 37, 121: 37, 126: 36, 132: 36, 135: 35, 136: 35, 150: 35, 122: 33, 138: 33, 140: 33, 142: 3
3, 183: 33, 123: 32, 137: 32, 139: 32, 146: 32, 155: 32, 143: 30, 147: 30, 171: 30, 160: 29, 161: 29, 222: 29, 129: 28, 167:
28, 168: 28, 173: 28, 148: 27, 176: 27, 193: 27, 208: 27, 162: 26, 188: 26, 191: 26, 154: 25, 156: 25, 164: 25, 184: 25, 12
5: 24, 166: 24, 182: 24, 225: 24, 149: 23, 153: 23, 177: 23, 180: 23, 181: 23, 192: 23, 196: 23, 207: 23, 277: 23, 165: 22,
189: 22, 205: 22, 211: 22, 234: 22, 258: 22, 157: 21, 158: 21, 178: 21, 199: 21, 206: 21, 209: 21, 216: 21, 152: 20, 163: 2
0, 170: 20, 172: 20, 174: 20, 194: 20, 200: 20, 203: 20, 212: 20, 220: 20, 227: 20, 244: 20, 250: 20, 204: 19, 210: 19, 223:
19, 236: 19, 290: 19, 195: 19, 169: 18, 186: 18, 231: 18, 263: 18, 269: 18, 285: 18, 197: 17, 213: 17, 218: 17, 233: 17, 25
9: 17, 185: 16, 187: 16, 214: 16, 224: 16, 232: 16, 242: 16, 254: 16, 313: 16, 344: 16, 406: 16, 175: 15, 198: 15, 202: 15,
221: 15, 228: 15, 230: 15, 248: 15, 268: 15, 272: 15, 299: 15, 308: 15, 314: 15, 217: 14, 219: 14, 243: 14, 247: 14, 262: 1
4, 288: 14, 301: 14, 311: 14, 320: 14, 329: 14, 190: 13, 237: 13, 245: 13, 249: 13, 251: 13, 257: 13, 265: 13, 296: 13, 297:
13, 303: 13, 315: 13, 229: 12, 239: 12, 261: 12, 264: 12, 270: 12, 271: 12, 273: 12, 276: 12, 286: 12, 289: 12, 291: 12, 30
0: 12, 346: 12, 354: 12, 386: 12, 393: 12, 179: 11, 226: 11, 235: 11, 241: 11, 246: 11, 255: 11, 260: 11, 266: 11, 267: 11,
279: 11, 284: 11, 302: 11, 324: 11, 332: 11, 350: 11, 362: 11, 373: 11, 381: 11, 441: 11, 295: 11, 215: 10, 240: 10, 253: 1
0, 274: 10, 275: 10, 280: 10, 287: 10, 293: 10, 294: 10, 304: 10, 310: 10, 316: 10, 317: 10, 319: 10, 326: 10, 333: 10, 338:
10, 341: 10, 348: 10, 357: 10, 377: 10, 415: 10, 425: 10, 436: 10, 282: 9, 283: 9, 292: 9, 298: 9, 309: 9, 312: 9, 318: 9, 3
55: 9, 364: 9, 368: 9, 392: 9, 403: 9, 442: 9, 446: 9, 481: 9, 489: 9, 500: 9, 278: 8, 281: 8, 307: 8, 323: 8, 325: 8, 328:
8, 330: 8, 331: 8, 337: 8, 342: 8, 343: 8, 360: 8, 361: 8, 365: 8, 372: 8, 374: 8, 378: 8, 380: 8, 387: 8, 394: 8, 399: 8, 4
02: 8, 414: 8, 420: 8, 424: 8, 437: 8, 445: 8, 450: 8, 456: 8, 457: 8, 467: 8, 478: 8, 492: 8, 493: 8, 583: 8, 238: 7, 252:
7, 321: 7, 322: 7, 336: 7, 347: 7, 349: 7, 352: 7, 353: 7, 369: 7, 376: 7, 384: 7, 390: 7, 396: 7, 409: 7, 411: 7, 413: 7, 4
16: 7, 430: 7, 431: 7, 434: 7, 440: 7, 444: 7, 448: 7, 459: 7, 460: 7, 468: 7, 469: 7, 472: 7, 475: 7, 476: 7, 488: 7, 498:
7, 503: 7, 520: 7, 522: 7, 548: 7, 560: 7, 571: 7, 572: 7, 588: 7, 593: 7, 612: 7, 631: 7, 648: 7, 680: 7, 1073: 7, 201: 6,
305: 6, 327: 6, 339: 6, 351: 6, 363: 6, 367: 6, 375: 6, 389: 6, 391: 6, 395: 6, 398: 6, 401: 6, 405: 6, 408: 6, 419: 6, 421:
6, 422: 6, 432: 6, 433: 6, 451: 6, 452: 6, 453: 6, 464: 6, 470: 6, 482: 6, 485: 6, 507: 6, 513: 6, 517: 6, 519: 6, 521: 6, 5
30: 6, 533: 6, 536: 6, 552: 6, 559: 6, 563: 6, 564: 6, 567: 6, 586: 6, 619: 6, 642: 6, 652: 6, 667: 6, 692: 6, 693: 6, 695:
6, 725: 6, 747: 6, 750: 6, 751: 6, 761: 6, 767: 6, 777: 6, 797: 6, 887: 6, 964: 6, 557: 6, 256: 5, 334: 5, 340: 5, 345: 5, 3
56: 5, 359: 5, 371: 5, 382: 5, 404: 5, 412: 5, 418: 5, 427: 5, 428: 5, 435: 5, 462: 5, 465: 5, 471: 5, 505: 5, 512: 5, 515:
5, 524: 5, 537: 5, 541: 5, 542: 5, 556: 5, 558: 5, 562: 5, 584: 5, 596: 5, 603: 5, 608: 5, 609: 5, 610: 5, 620: 5, 638: 5, 6
45: 5, 655: 5, 656: 5, 662: 5, 673: 5, 677: 5, 681: 5, 682: 5, 684: 5, 685: 5, 702: 5, 707: 5, 749: 5, 760: 5, 772: 5, 775:
```

```
5, 811: 5, 812: 5, 828: 5, 837: 5, 840: 5, 862: 5, 864: 5, 871: 5, 899: 5, 909: 5, 928: 5, 1187: 5, 1347: 5, 539: 5, 647: 5,
756: 5, 306: 4, 335: 4, 358: 4, 366: 4, 383: 4, 400: 4, 407: 4, 410: 4, 429: 4, 438: 4, 449: 4, 454: 4, 461: 4, 479: 4, 484:
4, 487: 4, 490: 4, 491: 4, 497: 4, 501: 4, 508: 4, 509: 4, 516: 4, 527: 4, 532: 4, 534: 4, 543: 4, 546: 4, 549: 4, 561: 4, 5
68: 4, 576: 4, 577: 4, 578: 4, 580: 4, 589: 4, 594: 4, 595: 4, 597: 4, 601: 4, 611: 4, 614: 4, 617: 4, 621: 4, 624: 4, 625:
4, 633: 4, 635: 4, 637: 4, 639: 4, 643: 4, 650: 4, 659: 4, 669: 4, 670: 4, 672: 4, 688: 4, 689: 4, 699: 4, 705: 4, 706: 4, 7
11: 4, 716: 4, 765: 4, 770: 4, 786: 4, 789: 4, 790: 4, 793: 4, 795: 4, 805: 4, 806: 4, 814: 4, 817: 4, 832: 4, 846: 4, 858:
4, 869: 4, 878: 4, 881: 4, 882: 4, 916: 4, 924: 4, 973: 4, 977: 4, 980: 4, 1021: 4, 1030: 4, 1047: 4, 1059: 4, 1083: 4, 110
7: 4, 1122: 4, 1152: 4, 1162: 4, 1197: 4, 1199: 4, 1295: 4, 1326: 4, 1329: 4, 1375: 4, 1396: 4, 1589: 4, 1757: 4, 3727: 4, 9
47: 4, 370: 3, 379: 3, 385: 3, 388: 3, 417: 3, 426: 3, 439: 3, 443: 3, 455: 3, 458: 3, 466: 3, 494: 3, 495: 3, 499: 3, 502:
3, 504: 3, 514: 3, 518: 3, 535: 3, 540: 3, 545: 3, 547: 3, 550: 3, 551: 3, 566: 3, 574: 3, 590: 3, 591: 3, 605: 3, 607: 3, 6
15: 3, 616: 3, 618: 3, 628: 3, 629: 3, 630: 3, 634: 3, 640: 3, 657: 3, 658: 3, 661: 3, 664: 3, 666: 3, 668: 3, 675: 3, 676:
3, 683: 3, 686: 3, 691: 3, 694: 3, 701: 3, 715: 3, 718: 3, 723: 3, 729: 3, 734: 3, 743: 3, 745: 3, 748: 3, 753: 3, 755: 3, 7
62: 3, 763: 3, 764: 3, 771: 3, 778: 3, 779: 3, 781: 3, 785: 3, 796: 3, 800: 3, 802: 3, 804: 3, 819: 3, 820: 3, 821: 3, 822:
3, 829: 3, 834: 3, 843: 3, 852: 3, 859: 3, 860: 3, 863: 3, 868: 3, 872: 3, 876: 3, 884: 3, 885: 3, 891: 3, 898: 3, 901: 3, 9
05: 3, 910: 3, 918: 3, 927: 3, 937: 3, 945: 3, 954: 3, 955: 3, 966: 3, 972: 3, 974: 3, 975: 3, 979: 3, 985: 3, 999: 3, 1006:
3, 1038: 3, 1039: 3, 1046: 3, 1049: 3, 1053: 3, 1079: 3, 1100: 3, 1115: 3, 1126: 3, 1159: 3, 1164: 3, 1169: 3, 1174: 3, 118
2: 3, 1191: 3, 1195: 3, 1217: 3, 1222: 3, 1227: 3, 1236: 3, 1245: 3, 1259: 3, 1281: 3, 1289: 3, 1298: 3, 1319: 3, 1323: 3, 1
339: 3, 1344: 3, 1345: 3, 1359: 3, 1377: 3, 1388: 3, 1390: 3, 1402: 3, 1419: 3, 1441: 3, 1442: 3, 1448: 3, 1466: 3, 1499: 3,
1540: 3, 1587: 3, 1630: 3, 1631: 3, 1636: 3, 1646: 3, 1692: 3, 1776: 3, 1777: 3, 1788: 3, 1793: 3, 1819: 3, 1837: 3, 1890:
3, 1894: 3, 2002: 3, 2018: 3, 2030: 3, 2058: 3, 2073: 3, 2160: 3, 2203: 3, 2241: 3, 2467: 3, 2529: 3, 423: 3, 3194: 3, 3335:
3, 3632: 3, 776: 3, 5010: 3, 915: 3, 397: 2, 447: 2, 463: 2, 473: 2, 477: 2, 496: 2, 506: 2, 510: 2, 511: 2, 523: 2, 525: 2,
526: 2, 528: 2, 531: 2, 544: 2, 553: 2, 565: 2, 569: 2, 581: 2, 585: 2, 587: 2, 598: 2, 599: 2, 602: 2, 604: 2, 606: 2, 623:
2, 626: 2, 627: 2, 636: 2, 649: 2, 653: 2, 679: 2, 687: 2, 698: 2, 703: 2, 708: 2, 714: 2, 719: 2, 720: 2, 722: 2, 724: 2, 7
26: 2, 728: 2, 730: 2, 733: 2, 735: 2, 739: 2, 744: 2, 746: 2, 757: 2, 759: 2, 766: 2, 769: 2, 782: 2, 783: 2, 784: 2, 788:
2, 792: 2, 798: 2, 799: 2, 801: 2, 807: 2, 815: 2, 818: 2, 824: 2, 833: 2, 839: 2, 847: 2, 849: 2, 850: 2, 851: 2, 853: 2, 8
54: 2, 856: 2, 857: 2, 861: 2, 866: 2, 875: 2, 883: 2, 888: 2, 890: 2, 897: 2, 904: 2, 907: 2, 919: 2, 921: 2, 923: 2, 925:
2, 930: 2, 941: 2, 946: 2, 949: 2, 951: 2, 952: 2, 958: 2, 961: 2, 962: 2, 968: 2, 976: 2, 992: 2, 993: 2, 994: 2, 1002: 2,
1003: 2, 1004: 2, 1005: 2, 1007: 2, 1010: 2, 1011: 2, 1013: 2, 1014: 2, 1018: 2, 1019: 2, 1022: 2, 1023: 2, 1026: 2, 1027:
2, 1028: 2, 1029: 2, 1037: 2, 1041: 2, 1043: 2, 1044: 2, 1051: 2, 1052: 2, 1054: 2, 1057: 2, 1061: 2, 1069: 2, 1070: 2, 107
8: 2, 1082: 2, 1085: 2, 1096: 2, 1098: 2, 1101: 2, 1105: 2, 1113: 2, 1116: 2, 1119: 2, 1127: 2, 1136: 2, 1140: 2, 1141: 2, 1
142: 2, 1144: 2, 1149: 2, 1151: 2, 1160: 2, 1166: 2, 1180: 2, 1184: 2, 1185: 2, 1186: 2, 1192: 2, 1196: 2, 1201: 2, 1204: 2,
1205: 2, 1213: 2, 1214: 2, 1215: 2, 1228: 2, 1229: 2, 1233: 2, 1240: 2, 1244: 2, 1248: 2, 1267: 2, 1271: 2, 1276: 2, 1282:
2, 1284: 2, 1287: 2, 1288: 2, 1290: 2, 1293: 2, 1301: 2, 1305: 2, 9498: 2, 1308: 2, 1309: 2, 1316: 2, 1317: 2, 1320: 2, 132
1: 2, 1324: 2, 1331: 2, 1335: 2, 1340: 2, 1346: 2, 1350: 2, 1355: 2, 1356: 2, 1367: 2, 1370: 2, 1373: 2, 1376: 2, 1378: 2, 1
395: 2, 1397: 2, 1401: 2, 1409: 2, 1422: 2, 1424: 2, 1431: 2, 1433: 2, 1450: 2, 1454: 2, 1458: 2, 1472: 2, 1482: 2, 1483: 2,
1486: 2, 1504: 2, 1508: 2, 1511: 2, 1517: 2, 1526: 2, 1541: 2, 1544: 2, 1547: 2, 1548: 2, 1550: 2, 1555: 2, 1562: 2, 1571:
2, 1578: 2, 1579: 2, 1580: 2, 1592: 2, 1593: 2, 1594: 2, 1599: 2, 1604: 2, 1606: 2, 1609: 2, 1611: 2, 1616: 2, 1618: 2, 164
7: 2, 1650: 2, 1655: 2, 1658: 2, 1672: 2, 1681: 2, 1700: 2, 1705: 2, 1710: 2, 1727: 2, 1731: 2, 1739: 2, 1768: 2, 1773: 2, 1
784: 2, 1804: 2, 1813: 2, 1818: 2, 1821: 2, 1839: 2, 1844: 2, 1859: 2, 1869: 2, 1879: 2, 1881: 2, 1884: 2, 1891: 2, 1910: 2,
1913: 2, 1932: 2, 1944: 2, 1956: 2, 1958: 2, 1980: 2, 2000: 2, 2003: 2, 2005: 2, 2012: 2, 2020: 2, 2028: 2, 2051: 2, 2052:
2, 2056: 2, 2060: 2, 2065: 2, 2080: 2, 2097: 2, 2107: 2, 2108: 2, 2116: 2, 2119: 2, 2123: 2, 2136: 2, 2149: 2, 2177: 2, 218
3: 2, 2199: 2, 2202: 2, 2223: 2, 2230: 2, 2244: 2, 2256: 2, 2261: 2, 2317: 2, 2336: 2, 2339: 2, 2340: 2, 2341: 2, 2369: 2, 2
377: 2, 2401: 2, 2410: 2, 2429: 2, 2433: 2, 2434: 2, 2448: 2, 2489: 2, 2499: 2, 2523: 2, 2542: 2, 2561: 2, 2566: 2, 2578: 2,
2617: 2, 2624: 2, 2633: 2, 2648: 2, 10885: 2, 2822: 2, 2842: 2, 2851: 2, 3074: 2, 3127: 2, 3200: 2, 3290: 2, 3326: 2, 3360:
2, 3396: 2, 3407: 2, 3435: 2, 3439: 2, 3447: 2, 3481: 2, 3532: 2, 3631: 2, 3728: 2, 3733: 2, 3751: 2, 3755: 2, 3826: 2, 388
7: 2, 654: 2, 12194: 2, 4022: 2, 4258: 2, 4268: 2, 4298: 2, 4319: 2, 721: 2, 4365: 2, 4551: 2, 4743: 2, 4871: 2, 813: 2, 488
2: 2, 826: 2, 4982: 2, 5069: 2, 5294: 2, 5311: 2, 5547: 2, 13773: 2, 5902: 2, 1068: 2, 6413: 2, 6888: 2, 1603: 2, 1263: 2, 7
844: 2, 2710: 2, 41009: 1, 8212: 1, 8387: 1, 16807: 1, 474: 1, 480: 1, 483: 1, 486: 1, 8280: 1, 8730: 1, 33307: 1, 16938: 1,
555: 1, 8749: 1, 570: 1, 573: 1, 575: 1, 600: 1, 613: 1, 641: 1, 644: 1, 646: 1, 1473: 1, 651: 1, 8846: 1, 660: 1, 663: 1, 6
65: 1, 671: 1, 674: 1, 678: 1, 690: 1, 696: 1, 697: 1, 700: 1, 704: 1, 710: 1, 712: 1, 713: 1, 717: 1, 33489: 1, 732: 1, 73
6: 1, 740: 1, 741: 1, 742: 1, 752: 1, 754: 1, 25332: 1, 758: 1, 768: 1, 773: 1, 8968: 1, 787: 1, 8995: 1, 808: 1, 17193: 1,
9005: 1, 816: 1, 823: 1, 825: 1, 9018: 1, 827: 1, 836: 1, 838: 1, 842: 1, 845: 1, 848: 1, 867: 1, 870: 1, 874: 1, 877: 1, 87
9: 1, 880: 1, 886: 1, 893: 1, 895: 1, 896: 1, 900: 1, 902: 1, 903: 1, 906: 1, 912: 1, 913: 1, 914: 1, 9107: 1, 917: 1, 920:
1, 926: 1, 931: 1, 934: 1, 935: 1, 936: 1, 938: 1, 939: 1, 940: 1, 942: 1, 943: 1, 944: 1, 9139: 1, 948: 1, 1524: 1, 963: 1,
969: 1, 971: 1, 978: 1, 983: 1, 986: 1, 987: 1, 988: 1, 990: 1, 991: 1, 995: 1, 996: 1, 997: 1, 9190: 1, 1000: 1, 1001: 1, 1
008: 1, 1015: 1, 1016: 1, 1017: 1, 1020: 1, 1024: 1, 1031: 1, 1034: 1, 1040: 1, 1042: 1, 1045: 1, 1048: 1, 1050: 1, 1055: 1,
1058: 1, 1060: 1, 1062: 1, 1063: 1, 1065: 1, 1066: 1, 1067: 1, 9260: 1, 1071: 1, 1072: 1, 1074: 1, 1075: 1, 1076: 1, 9269:
1, 9272: 1, 32948: 1, 1087: 1, 1088: 1, 1089: 1, 1090: 1, 1091: 1, 1093: 1, 1094: 1, 1095: 1, 1097: 1, 1099: 1, 1549: 1, 110
6: 1, 1108: 1, 1109: 1, 1110: 1, 1112: 1, 17498: 1, 1117: 1, 1120: 1, 1121: 1, 9315: 1, 1124: 1, 1125: 1, 1131: 1, 1133: 1,
1134: 1, 1135: 1, 1138: 1, 1143: 1, 1145: 1, 1146: 1, 1147: 1, 1148: 1, 1150: 1, 1153: 1, 1154: 1, 9348: 1, 1157: 1, 1161:
1, 1168: 1, 1170: 1, 1172: 1, 1173: 1, 1177: 1, 17563: 1, 1181: 1, 1188: 1, 24774: 1, 1194: 1, 17584: 1, 1203: 1, 1206: 1, 1
207: 1, 1209: 1, 1210: 1, 1211: 1, 1218: 1, 1219: 1, 9412: 1, 1221: 1, 42184: 1, 1225: 1, 1226: 1, 1230: 1, 1232: 1, 1237:
1, 1238: 1, 1241: 1, 1243: 1, 1246: 1, 1249: 1, 1251: 1, 1252: 1, 1253: 1, 1254: 1, 1255: 1, 1256: 1, 1257: 1, 1258: 1, 126
1: 1, 1262: 1, 9455: 1, 1265: 1, 1266: 1, 9460: 1, 1269: 1, 1270: 1, 1273: 1, 1275: 1, 1280: 1, 9477: 1, 1291: 1, 1296: 1, 1
297: 1, 1299: 1, 1300: 1, 1302: 1, 1303: 1, 1310: 1, 1311: 1, 1313: 1, 1314: 1, 1322: 1, 1325: 1, 1328: 1, 1333: 1, 1336: 1,
1338: 1, 1341: 1, 1348: 1, 1352: 1, 1353: 1, 1354: 1, 1360: 1, 1361: 1, 1364: 1, 1365: 1, 1366: 1, 1369: 1, 1371: 1, 1374:
1, 1379: 1, 1380: 1, 1381: 1, 1383: 1, 1384: 1, 25963: 1, 1389: 1, 1392: 1, 1394: 1, 1398: 1, 1400: 1, 1403: 1, 1404: 1, 141
0: 1, 1411: 1, 9604: 1, 1413: 1, 1416: 1, 1420: 1, 1425: 1, 1427: 1, 1428: 1, 1429: 1, 1430: 1, 1434: 1, 1436: 1, 1437: 1, 1
438: 1, 9631: 1, 1443: 1, 1444: 1, 1449: 1, 1452: 1, 1453: 1, 1456: 1, 17841: 1, 66995: 1, 1460: 1, 1462: 1, 50620: 1, 1469:
1, 17857: 1, 1479: 1, 1480: 1, 1481: 1, 1484: 1, 1487: 1, 26065: 1, 1491: 1, 1493: 1, 1494: 1, 1498: 1, 4346: 1, 1507: 1, 15
09: 1, 1512: 1, 1513: 1, 1514: 1, 1519: 1, 1523: 1, 9716: 1, 1527: 1, 1533: 1, 1534: 1, 1536: 1, 1542: 1, 1545: 1, 9741: 1,
1551: 1, 1553: 1, 1554: 1, 1556: 1, 9749: 1, 1560: 1, 1563: 1, 1565: 1, 9759: 1, 1568: 1, 1569: 1, 1570: 1, 1572: 1, 1574:
1, 1576: 1, 1577: 1, 1581: 1, 1584: 1, 16648: 1, 1588: 1, 1596: 1, 1597: 1, 1598: 1, 8459: 1, 1612: 1, 1614: 1, 1619: 1, 162
1: 1, 1623: 1, 1624: 1, 1625: 1, 1626: 1, 1627: 1, 1632: 1, 1633: 1, 9830: 1, 1640: 1, 1641: 1, 1644: 1, 1645: 1, 1648: 1, 1
651: 1, 1659: 1, 1660: 1, 1666: 1, 1667: 1, 1670: 1, 18055: 1, 1675: 1, 1677: 1, 1682: 1, 1684: 1, 1685: 1, 1686: 1, 1690:
1, 1694: 1, 1695: 1, 1696: 1, 1702: 1, 1703: 1, 1704: 1, 1706: 1, 1708: 1, 1709: 1, 1711: 1, 1712: 1, 1713: 1, 16671: 1, 172
8: 1, 1729: 1, 1730: 1, 1732: 1, 8241: 1, 1734: 1, 1735: 1, 1736: 1, 1737: 1, 1740: 1, 1741: 1, 1742: 1, 1743: 1, 1745: 1, 1
746: 1, 1747: 1, 18132: 1, 1753: 1, 1756: 1, 1758: 1, 1759: 1, 1760: 1, 1761: 1, 1762: 1, 9957: 1, 1766: 1, 1769: 1, 1770:
```

```
1, 1778: 1, 1781: 1, 1782: 1, 1783: 1, 1785: 1, 1786: 1, 1790: 1, 1791: 1, 1797: 1, 42758: 1, 1800: 1, 1801: 1, 1802: 1, 180
3: 1, 10001: 1, 9859: 1, 1816: 1, 1817: 1, 1827: 1, 1829: 1, 1830: 1, 1832: 1, 1833: 1, 1838: 1, 1841: 1, 18226: 1, 1845: 1,
18230: 1, 1847: 1, 1848: 1, 1850: 1, 1851: 1, 1852: 1, 1853: 1, 1854: 1, 1856: 1, 1861: 1, 1862: 1, 1863: 1, 1865: 1, 1866:
1, 1868: 1, 1872: 1, 1873: 1, 1874: 1, 1875: 1, 1877: 1, 1878: 1, 1882: 1, 1883: 1, 1886: 1, 1889: 1, 1893: 1, 1896: 1, 189
7: 1, 1903: 1, 1905: 1, 1908: 1, 1912: 1, 1915: 1, 1916: 1, 1917: 1, 1920: 1, 1922: 1, 1923: 1, 1925: 1, 1927: 1, 1928: 1, 1
929: 1, 1934: 1, 1936: 1, 1938: 1, 1940: 1, 1942: 1, 1943: 1, 1948: 1, 1949: 1, 1950: 1, 1951: 1, 1954: 1, 1962: 1, 1963: 1,
1966: 1, 1972: 1, 1975: 1, 18360: 1, 1977: 1, 1978: 1, 24889: 1, 1983: 1, 1985: 1, 1988: 1, 1991: 1, 1992: 1, 1995: 1, 1998:
1, 2006: 1, 2017: 1, 2021: 1, 2022: 1, 2023: 1, 2033: 1, 2034: 1, 18419: 1, 2036: 1, 2037: 1, 2043: 1, 2044: 1, 10239: 1, 11
264: 1, 10242: 1, 2057: 1, 2067: 1, 9902: 1, 2071: 1, 2077: 1, 2079: 1, 2083: 1, 2086: 1, 46769: 1, 2092: 1, 2093: 1, 10290:
1, 67636: 1, 2105: 1, 2109: 1, 2114: 1, 2115: 1, 2117: 1, 2124: 1, 2127: 1, 2128: 1, 2130: 1, 10323: 1, 2137: 1, 2138: 1, 21
39: 1, 2140: 1, 2141: 1, 2143: 1, 2144: 1, 2146: 1, 2151: 1, 2154: 1, 2155: 1, 2157: 1, 10355: 1, 2173: 1, 2174: 1, 2176: 1,
2179: 1, 2181: 1, 2182: 1, 2185: 1, 2193: 1, 2197: 1, 2205: 1, 2214: 1, 2215: 1, 2216: 1, 2218: 1, 2219: 1, 2221: 1, 2224:
1, 2226: 1, 2233: 1, 2234: 1, 2235: 1, 2236: 1, 2245: 1, 9932: 1, 2253: 1, 2254: 1, 2263: 1, 10457: 1, 2270: 1, 2277: 1, 228
2: 1, 2286: 1, 8573: 1, 2289: 1, 2296: 1, 2300: 1, 2301: 1, 2302: 1, 2303: 1, 2305: 1, 2310: 1, 18697: 1, 2314: 1, 2322: 1,
2327: 1, 2331: 1, 2343: 1, 2344: 1, 26538: 1, 26928: 1, 2355: 1, 2360: 1, 2362: 1, 2368: 1, 2373: 1, 43997: 1, 2379: 1, 238
2: 1, 2386: 1, 2387: 1, 2389: 1, 2390: 1, 2391: 1, 2393: 1, 2394: 1, 2396: 1, 10589: 1, 2398: 1, 2399: 1, 2411: 1, 2412: 1,
2414: 1, 2415: 1, 2418: 1, 2419: 1, 2420: 1, 10614: 1, 2425: 1, 2427: 1, 2432: 1, 1771: 1, 2436: 1, 2437: 1, 2438: 1, 2441:
1, 2442: 1, 2446: 1, 2450: 1, 2451: 1, 2455: 1, 2458: 1, 2459: 1, 2460: 1, 2471: 1, 18859: 1, 2481: 1, 2491: 1, 2493: 1, 249
6: 1, 2500: 1, 18885: 1, 2503: 1, 2515: 1, 2519: 1, 2520: 1, 2526: 1, 2527: 1, 2528: 1, 2530: 1, 2535: 1, 2540: 1, 18361: 1,
2548: 1, 2549: 1, 2550: 1, 2552: 1, 2554: 1, 2555: 1, 2557: 1, 2563: 1, 2570: 1, 2575: 1, 2580: 1, 2581: 1, 2582: 1, 2584:
1, 2587: 1, 2593: 1, 2595: 1, 2597: 1, 2598: 1, 2599: 1, 2600: 1, 2602: 1, 2606: 1, 2607: 1, 2608: 1, 2611: 1, 2612: 1, 261
3: 1, 2621: 1, 2625: 1, 2629: 1, 2634: 1, 2640: 1, 38670: 1, 2651: 1, 2653: 1, 2654: 1, 2655: 1, 2656: 1, 2660: 1, 2661: 1,
2663: 1, 2664: 1, 2666: 1, 2667: 1, 2670: 1, 2677: 1, 2691: 1, 2692: 1, 2694: 1, 2697: 1, 2698: 1, 2701: 1, 2703: 1, 2707:
1, 10902: 1, 2713: 1, 2715: 1, 2721: 1, 10011: 1, 2724: 1, 2725: 1, 2729: 1, 2740: 1, 2755: 1, 2761: 1, 2762: 1, 2772: 1, 27
74: 1, 2775: 1, 2777: 1, 2779: 1, 2782: 1, 10978: 1, 2790: 1, 2792: 1, 2793: 1, 2794: 1, 2799: 1, 1457: 1, 2803: 1, 2809: 1,
2812: 1, 2816: 1, 2818: 1, 2823: 1, 2824: 1, 2826: 1, 2829: 1, 11024: 1, 8664: 1, 2837: 1, 19223: 1, 2856: 1, 8668: 1, 1459:
1, 2863: 1, 2865: 1, 2866: 1, 2871: 1, 11067: 1, 1846: 1, 2891: 1, 2898: 1, 2901: 1, 19287: 1, 2905: 1, 1765: 1, 2916: 1, 29
19: 1, 2923: 1, 2930: 1, 2935: 1, 2940: 1, 11134: 1, 2957: 1, 2958: 1, 2961: 1, 2963: 1, 2964: 1, 2971: 1, 2972: 1, 2976: 1,
2984: 1, 2988: 1, 2989: 1, 11182: 1, 2991: 1, 2994: 1, 2995: 1, 3001: 1, 3002: 1, 3007: 1, 3017: 1, 3020: 1, 3026: 1, 3028:
1, 3029: 1, 3037: 1, 3044: 1, 3050: 1, 3054: 1, 3055: 1, 3058: 1, 3060: 1, 3062: 1, 3064: 1, 3065: 1, 11260: 1, 19453: 1, 30
72: 1, 41474: 1, 3088: 1, 3090: 1, 3091: 1, 3096: 1, 3099: 1, 3102: 1, 3103: 1, 3108: 1, 3110: 1, 3111: 1, 19497: 1, 3115:
1, 24654: 1, 3118: 1, 3119: 1, 3121: 1, 1468: 1, 3135: 1, 3138: 1, 3149: 1, 3152: 1, 3154: 1, 3155: 1, 3159: 1, 3160: 1, 316
6: 1, 8368: 1, 3176: 1, 3190: 1, 3191: 1, 11385: 1, 3195: 1, 3198: 1, 33302: 1, 3216: 1, 3220: 1, 35995: 1, 3228: 1, 538: 1,
3230: 1, 36000: 1, 3237: 1, 3240: 1, 11433: 1, 3243: 1, 3244: 1, 3246: 1, 11440: 1, 3249: 1, 3250: 1, 16748: 1, 3258: 1, 326
0: 1, 3261: 1, 3262: 1, 3265: 1, 3270: 1, 3280: 1, 3282: 1, 3285: 1, 3289: 1, 36059: 1, 8293: 1, 3313: 1, 3315: 1, 554: 1, 3
328: 1, 3329: 1, 3331: 1, 8748: 1, 3341: 1, 3351: 1, 3367: 1, 11560: 1, 3371: 1, 3374: 1, 3376: 1, 8755: 1, 11574: 1, 3385:
1, 3295: 1, 3388: 1, 3390: 1, 3392: 1, 3398: 1, 3399: 1, 3400: 1, 3408: 1, 3411: 1, 3415: 1, 3416: 1, 11610: 1, 11611: 1, 34
22: 1, 3442: 1, 3443: 1, 3444: 1, 3449: 1, 3456: 1, 3463: 1, 3466: 1, 3467: 1, 8770: 1, 3470: 1, 3475: 1, 3476: 1, 3482: 1,
3486: 1, 3487: 1, 3490: 1, 3493: 1, 3496: 1, 3497: 1, 19884: 1, 3504: 1, 3510: 1, 150967: 1, 3515: 1, 3527: 1, 3529: 1, 1173
```

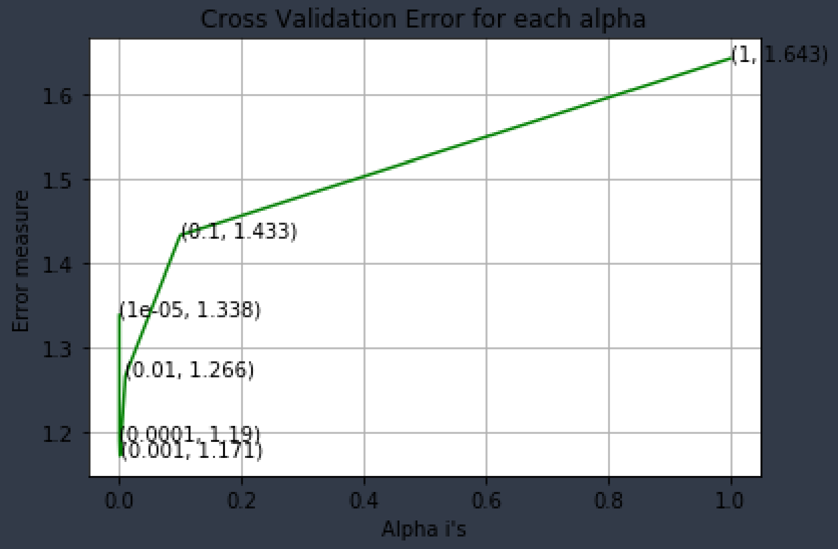
```
1: 1, 3545: 1, 3546: 1, 3548: 1, 3552: 1, 3554: 1, 3558: 1, 11760: 1, 3569: 1, 3572: 1, 3576: 1, 3577: 1, 3581: 1, 3582: 1,
3586: 1, 3591: 1, 3592: 1, 3596: 1, 3606: 1, 3608: 1, 3611: 1, 3615: 1, 3618: 1, 3619: 1, 3623: 1, 3625: 1, 3633: 1, 11827:
1, 11828: 1, 3337: 1, 3640: 1, 28230: 1, 11852: 1, 3671: 1, 3678: 1, 3681: 1, 3684: 1, 3698: 1, 3702: 1, 3703: 1, 3704: 1, 3
706: 1, 3709: 1, 3711: 1, 3712: 1, 8307: 1, 3717: 1, 3718: 1, 11911: 1, 3720: 1, 11921: 1, 3735: 1, 8308: 1, 3752: 1, 3754:
1, 3758: 1, 3759: 1, 3760: 1, 3768: 1, 3773: 1, 3777: 1, 10187: 1, 3782: 1, 3784: 1, 3787: 1, 3793: 1, 3808: 1, 12028: 1, 38
38: 1, 12033: 1, 3842: 1, 3846: 1, 3859: 1, 12058: 1, 8209: 1, 3870: 1, 3873: 1, 12069: 1, 3893: 1, 12087: 1, 3899: 1, 3901:
1, 3903: 1, 3915: 1, 3935: 1, 3936: 1, 3937: 1, 3939: 1, 3940: 1, 3941: 1, 3944: 1, 3947: 1, 3958: 1, 3959: 1, 3961: 1, 398
0: 1, 3981: 1, 3991: 1, 14319: 1, 3998: 1, 4001: 1, 2861: 1, 4007: 1, 4008: 1, 4009: 1, 4014: 1, 12210: 1, 4020: 1, 4031: 1,
4032: 1, 4035: 1, 4037: 1, 12233: 1, 4044: 1, 12238: 1, 4065: 1, 4068: 1, 4075: 1, 4076: 1, 4083: 1, 12280: 1, 4090: 1, 410
5: 1, 2050: 1, 20501: 1, 4118: 1, 1501: 1, 4136: 1, 4138: 1, 4148: 1, 4151: 1, 4153: 1, 12353: 1, 4164: 1, 4166: 1, 4169: 1,
4178: 1, 4180: 1, 4186: 1, 20571: 1, 4191: 1, 4193: 1, 4212: 1, 12408: 1, 4219: 1, 4226: 1, 4234: 1, 4236: 1, 4245: 1, 4248:
1, 4254: 1, 4256: 1, 4257: 1, 20643: 1, 4273: 1, 4275: 1, 4280: 1, 20667: 1, 4284: 1, 12478: 1, 4302: 1, 4303: 1, 4309: 1, 4
314: 1, 12512: 1, 6739: 1, 12516: 1, 4325: 1, 4333: 1, 4335: 1, 12538: 1, 4350: 1, 12545: 1, 8485: 1, 4357: 1, 727: 1, 4364:
1, 4366: 1, 4372: 1, 12566: 1, 4390: 1, 4394: 1, 12598: 1, 4429: 1, 4432: 1, 12627: 1, 4446: 1, 4448: 1, 4450: 1, 4462: 1, 4
471: 1, 12674: 1, 41707: 1, 4491: 1, 4492: 1, 4498: 1, 4509: 1, 17136: 1, 4517: 1, 4521: 1, 4523: 1, 4527: 1, 12732: 1, 454
1: 1, 8949: 1, 4545: 1, 4552: 1, 4554: 1, 4559: 1, 4561: 1, 4562: 1, 4590: 1, 4605: 1, 4610: 1, 4615: 1, 4617: 1, 12829: 1,
4646: 1, 4653: 1, 4672: 1, 4673: 1, 4681: 1, 4705: 1, 4715: 1, 4717: 1, 4725: 1, 15808: 1, 4740: 1, 4745: 1, 4749: 1, 4758:
1, 12953: 1, 4762: 1, 10352: 1, 4803: 1, 12998: 1, 4807: 1, 4813: 1, 4816: 1, 4829: 1, 4845: 1, 1798: 1, 4853: 1, 809: 1, 48
57: 1, 4868: 1, 4881: 1, 4886: 1, 4889: 1, 4896: 1, 4897: 1, 8816: 1, 13099: 1, 4929: 1, 4932: 1, 66358: 1, 13126: 1, 4946:
1, 4950: 1, 4952: 1, 4962: 1, 9019: 1, 4974: 1, 4979: 1, 4990: 1, 37767: 1, 9144: 1, 5011: 1, 5029: 1, 5033: 1, 5047: 1, 132
48: 1, 5058: 1, 5059: 1, 5066: 1, 5067: 1, 33613: 1, 5084: 1, 5092: 1, 5094: 1, 5114: 1, 5135: 1, 13328: 1, 13144: 1, 5139:
1, 9051: 1, 5169: 1, 5174: 1, 29756: 1, 46148: 1, 5213: 1, 5237: 1, 5240: 1, 5630: 1, 13469: 1, 13470: 1, 5335: 1, 9083: 1,
5349: 1, 5350: 1, 5389: 1, 21776: 1, 2265: 1, 13594: 1, 21794: 1, 5414: 1, 5420: 1, 5423: 1, 13618: 1, 5434: 1, 5451: 1, 545
2: 1, 5467: 1, 5468: 1, 5480: 1, 5490: 1, 5505: 1, 5511: 1, 13707: 1, 5521: 1, 5526: 1, 5529: 1, 5536: 1, 13747: 1, 13755:
1, 5589: 1, 5595: 1, 5600: 1, 21987: 1, 5617: 1, 5619: 1, 62970: 1, 67019: 1, 5641: 1, 5643: 1, 5651: 1, 54815: 1, 5682: 1,
5689: 1, 5692: 1, 5697: 1, 13894: 1, 54860: 1, 8919: 1, 5713: 1, 5714: 1, 5719: 1, 5720: 1, 5778: 1, 5783: 1, 5787: 1, 2217
8: 1, 50120: 1, 1285: 1, 5833: 1, 5874: 1, 5882: 1, 5883: 1, 5886: 1, 22276: 1, 14086: 1, 22287: 1, 5909: 1, 5919: 1, 14116:
1, 2801: 1, 22344: 1, 5973: 1, 5981: 1, 5984: 1, 5992: 1, 5995: 1, 6003: 1, 6006: 1, 9193: 1, 6010: 1, 22402: 1, 11927: 1, 6
032: 1, 6040: 1, 6084: 1, 6097: 1, 6102: 1, 6104: 1, 6113: 1, 6115: 1, 41981: 1, 6147: 1, 6177: 1, 6185: 1, 6187: 1, 6203:
1, 6206: 1, 14399: 1, 6209: 1, 6214: 1, 6222: 1, 120944: 1, 6271: 1, 6296: 1, 6303: 1, 6322: 1, 6324: 1, 6330: 1, 6335: 1, 6
360: 1, 6366: 1, 14559: 1, 6372: 1, 6383: 1, 6392: 1, 6398: 1, 6399: 1, 6402: 1, 14604: 1, 6451: 1, 14644: 1, 1077: 1, 6466:
1, 6473: 1, 6481: 1, 39275: 1, 6527: 1, 63880: 1, 6553: 1, 14747: 1, 6557: 1, 6559: 1, 14765: 1, 6583: 1, 6585: 1, 6597: 1,
6599: 1, 14800: 1, 6612: 1, 6618: 1, 6638: 1, 1585: 1, 12034: 1, 6678: 1, 6693: 1, 6729: 1, 1123: 1, 9772: 1, 6753: 1, 6768:
1, 24776: 1, 6782: 1, 10688: 1, 2501: 1, 6821: 1, 6843: 1, 6855: 1, 15065: 1, 3877: 1, 6903: 1, 17536: 1, 6930: 1, 1156: 1,
6943: 1, 6952: 1, 6962: 1, 6967: 1, 6980: 1, 16873: 1, 7013: 1, 15207: 1, 15210: 1, 1171: 1, 7032: 1, 7035: 1, 7046: 1, 706
0: 1, 7074: 1, 1179: 1, 7083: 1, 7095: 1, 7098: 1, 7106: 1, 7108: 1, 7116: 1, 7117: 1, 7121: 1, 7132: 1, 1189: 1, 7139: 1, 1
5338: 1, 7158: 1, 7160: 1, 7172: 1, 7181: 1, 7184: 1, 1200: 1, 40004: 1, 7249: 1, 1412: 1, 31831: 1, 15449: 1, 7260: 1, 1547
6: 1, 32790: 1, 7298: 1, 15495: 1, 7324: 1, 15532: 1, 23743: 1, 81096: 1, 7399: 1, 15609: 1, 7419: 1, 23810: 1, 7443: 1, 238
45: 1, 7464: 1, 7493: 1, 7501: 1, 7503: 1, 7518: 1, 7532: 1, 7533: 1, 7559: 1, 7574: 1, 7576: 1, 15787: 1, 7596: 1, 1268: 1,
32192: 1, 9462: 1, 7634: 1, 7638: 1, 17713: 1, 15858: 1, 7667: 1, 8712: 1, 7692: 1, 15886: 1, 17669: 1, 7729: 1, 7750: 1, 77
64: 1, 7769: 1, 7770: 1, 7775: 1, 7806: 1, 1306: 1, 7840: 1, 7864: 1, 10083: 1, 7963: 1, 16166: 1, 24359: 1, 7987: 1, 7990:
1, 7993: 1, 7994: 1, 9527: 1, 8022: 1, 8042: 1, 16252: 1, 8063: 1, 8095: 1, 8096: 1, 8105: 1, 2723: 1, 8455: 1, 9553: 1, 817
0: 1, 24566: 1})
```

In [50]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t 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",cv=None)
    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, labe
ls=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",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.3381902080683417
For values of alpha =  0.0001 The log loss is: 1.189774336644601
For values of alpha =  0.001 The log loss is: 1.1711849306609765
For values of alpha =  0.01 The log loss is: 1.2661660207388017
For values of alpha =  0.1 The log loss is: 1.4329881488304357
For values of alpha =  1 The log loss is: 1.6430932501445323
```



```
For values of best alpha =  0.001 The train log loss is: 0.7152773566845694
For values of best alpha =  0.001 The cross validation log loss is: 1.1711849306609765
For values of best alpha =  0.001 The test log loss is: 1.1193142751961653
```

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

Ans. Yes, it seems like!

In [51]:
```
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 [52]:
```
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 dat
a")
```

```
97.061 % of word of test data appeared in train data
97.854 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [53]:
```
#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",cv =None)
    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 ea
ch 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 [54]:
```
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 [55]:
```
# 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 quer
y point")
```

Stacking the three types of features

In [56]:
```
# 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_fe
ature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
re_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one
hotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCod
ing)).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)).toc
sr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_varia
tion_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variatio
n_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feat
ure_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_re
sponseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_respo
nseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCod
ing))
```

In [57]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55825)
```

```
(number of data points * number of features) in test data =  (665, 55825)
(number of data points * number of features) in cross validation data = (532, 55825)
```

In [58]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

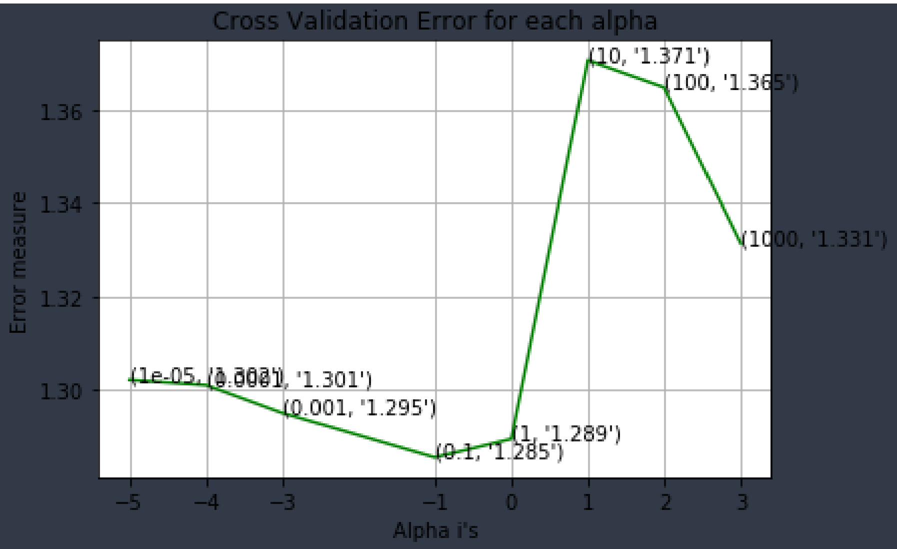
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [59]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=
3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/na
ive-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",cv=None)
    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 esti
mates
    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",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.3019887897106295
for alpha = 0.0001
Log Loss : 1.300875080188898
for alpha = 0.001
Log Loss : 1.2948544096648285
for alpha = 0.1
Log Loss : 1.2852735223933165
for alpha = 1
Log Loss : 1.2892465620145752
for alpha = 10
Log Loss : 1.3708098192870757
for alpha = 100
Log Loss : 1.365013032962014
for alpha = 1000
Log Loss : 1.3314332639172644
```



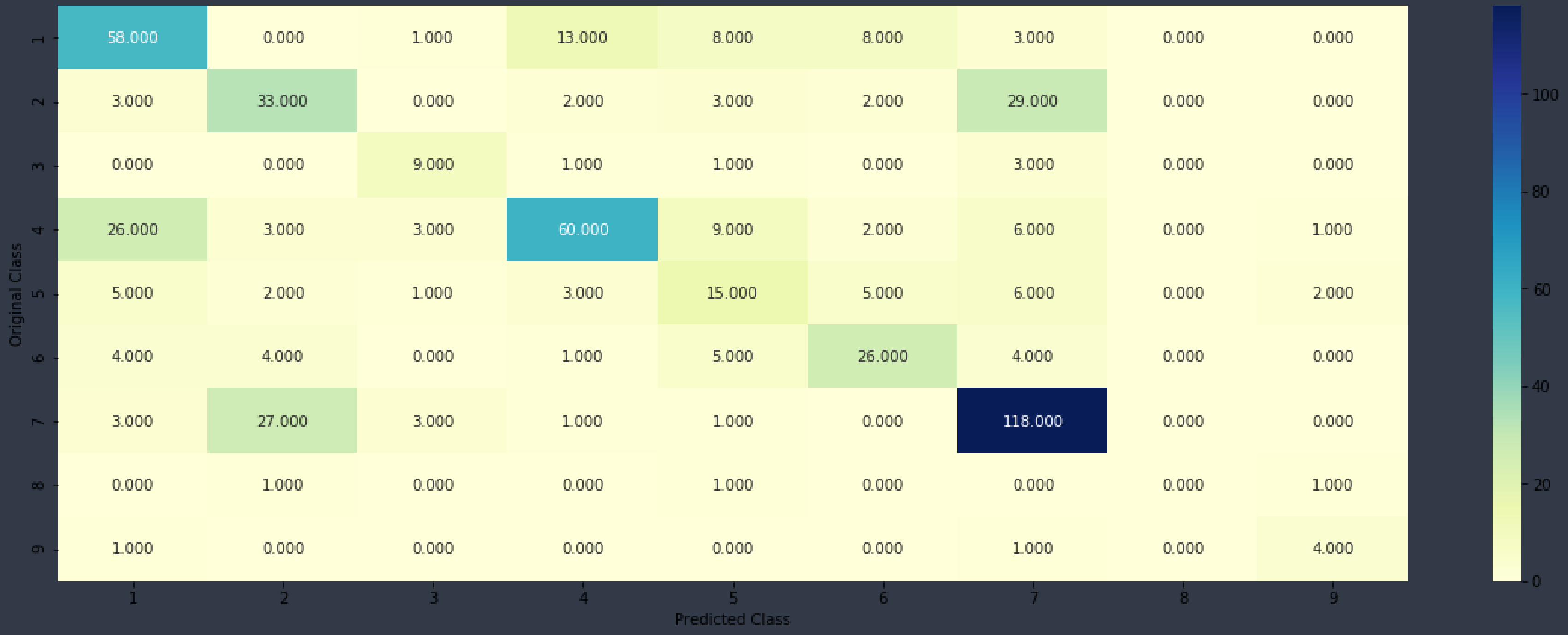
```
For values of best alpha =  0.1 The train log loss is: 0.9083908465682289
For values of best alpha =  0.1 The cross validation log loss is: 1.2852735223933165
For values of best alpha =  0.1 The test log loss is: 1.2197307184179849
```

4.1.1.2. Testing the model with best hyper paramters

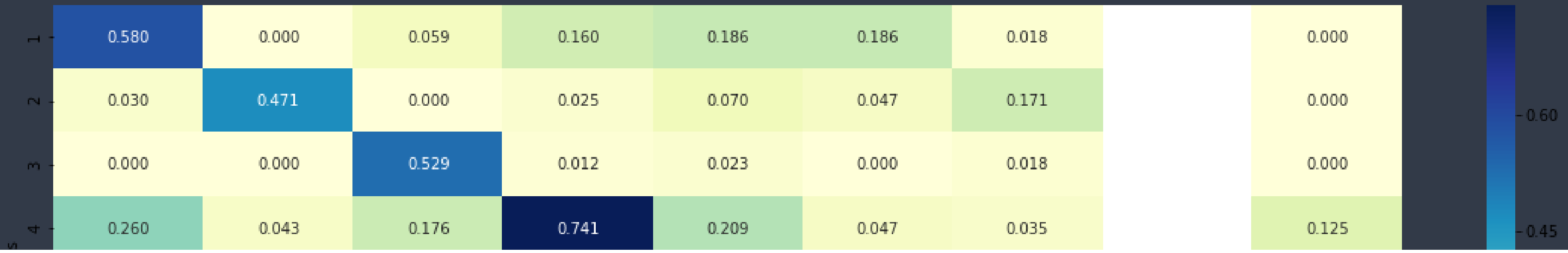
In [60]:

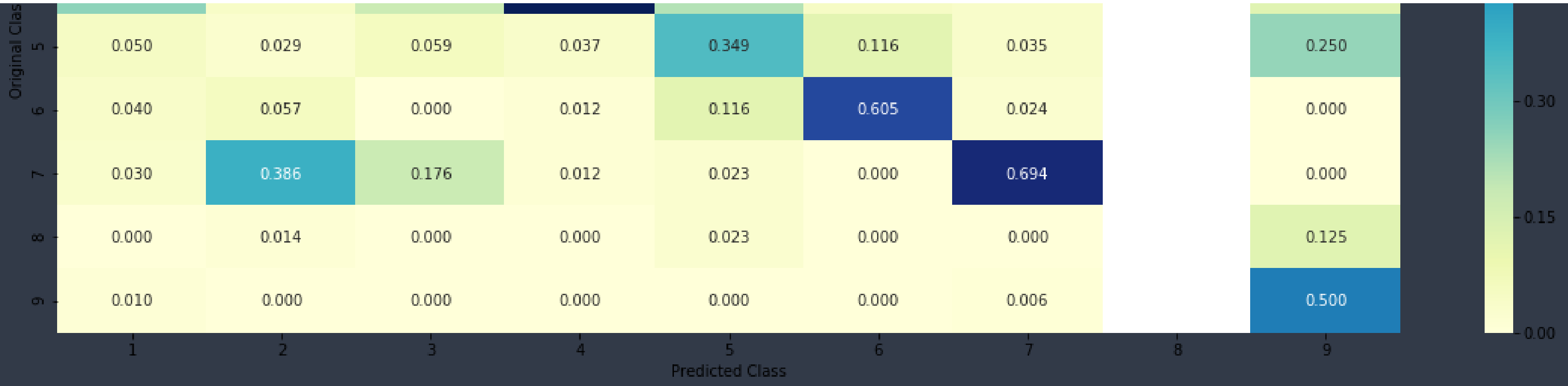
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/mo
dules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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



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

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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 estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.2852735223933165
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```

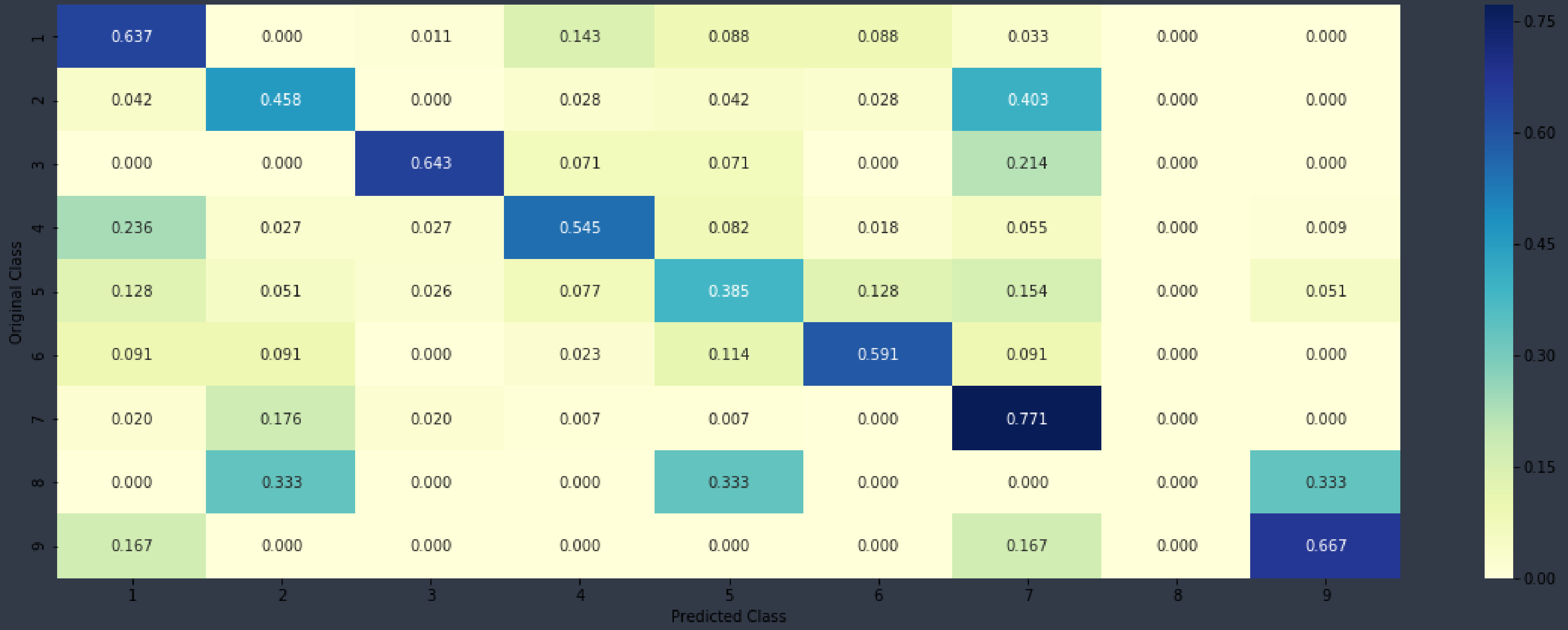


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.1.1.3. Feature Importance, Correctly classified point

In [61]:
```
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_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5555 0.0793 0.0149 0.1164 0.0383 0.0389 0.1487 0.0046 0.0034]]
Actual Class : 1
--------------------------------------------------
10 Text feature [affect] present in test data point [True]
11 Text feature [function] present in test data point [True]
12 Text feature [one] present in test data point [True]
13 Text feature [dna] present in test data point [True]
14 Text feature [type] present in test data point [True]
15 Text feature [protein] present in test data point [True]
16 Text feature [two] present in test data point [True]
17 Text feature [region] present in test data point [True]
19 Text feature [wild] present in test data point [True]
21 Text feature [containing] present in test data point [True]
22 Text feature [sequence] present in test data point [True]
23 Text feature [binding] present in test data point [True]
24 Text feature [possible] present in test data point [True]
25 Text feature [large] present in test data point [True]
26 Text feature [indicate] present in test data point [True]
27 Text feature [loss] present in test data point [True]

28 Text feature [four] present in test data point [True]
```

```
28 Text feature [four] present in test data point [True]
29 Text feature [present] present in test data point [True]
31 Text feature [gene] present in test data point [True]
32 Text feature [six] present in test data point [True]
34 Text feature [therefore] present in test data point [True]
38 Text feature [specific] present in test data point [True]
39 Text feature [corresponding] present in test data point [True]
40 Text feature [functions] present in test data point [True]
41 Text feature [used] present in test data point [True]
43 Text feature [five] present in test data point [True]
45 Text feature [identified] present in test data point [True]
46 Text feature [involved] present in test data point [True]
47 Text feature [three] present in test data point [True]
48 Text feature [results] present in test data point [True]
51 Text feature [deletion] present in test data point [True]
52 Text feature [change] present in test data point [True]
53 Text feature [data] present in test data point [True]
54 Text feature [define] present in test data point [True]
55 Text feature [form] present in test data point [True]
57 Text feature [using] present in test data point [True]
58 Text feature [structure] present in test data point [True]
59 Text feature [effect] present in test data point [True]
60 Text feature [conserved] present in test data point [True]
61 Text feature [surface] present in test data point [True]
63 Text feature [specifically] present in test data point [True]
65 Text feature [table] present in test data point [True]
66 Text feature [also] present in test data point [True]
67 Text feature [control] present in test data point [True]
68 Text feature [following] present in test data point [True]
69 Text feature [contains] present in test data point [True]
70 Text feature [result] present in test data point [True]
71 Text feature [ability] present in test data point [True]
72 Text feature [determined] present in test data point [True]
75 Text feature [located] present in test data point [True]
76 Text feature [defined] present in test data point [True]
77 Text feature [performed] present in test data point [True]
78 Text feature [terminal] present in test data point [True]
79 Text feature [transcriptional] present in test data point [True]
80 Text feature [reveal] present in test data point [True]

82 Text feature [additional] present in test data point [True]
```

```
82 Text feature [additional] present in test data point [True]
84 Text feature [indicated] present in test data point [True]
85 Text feature [fig] present in test data point [True]
86 Text feature [changes] present in test data point [True]
87 Text feature [expected] present in test data point [True]
88 Text feature [identify] present in test data point [True]
89 Text feature [discussion] present in test data point [True]
90 Text feature [acids] present in test data point [True]
91 Text feature [possibility] present in test data point [True]
93 Text feature [either] present in test data point [True]
94 Text feature [length] present in test data point [True]
95 Text feature [important] present in test data point [True]
96 Text feature [individual] present in test data point [True]
97 Text feature [addition] present in test data point [True]
98 Text feature [together] present in test data point [True]
99 Text feature [efficiency] present in test data point [True]
Out of the top  100  features  71 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [62]:

```python
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_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.251  0.0905 0.015  0.1522 0.0386 0.0391 0.4057 0.0045 0.0034]]
Actual Class : 4
```

```
---------------------------------------------------
16 Text feature [presence] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [downstream] present in test data point [True]
19 Text feature [independent] present in test data point [True]
20 Text feature [inhibitor] present in test data point [True]
21 Text feature [well] present in test data point [True]
23 Text feature [expressing] present in test data point [True]
24 Text feature [cell] present in test data point [True]
25 Text feature [showed] present in test data point [True]
30 Text feature [contrast] present in test data point [True]
31 Text feature [recently] present in test data point [True]
32 Text feature [previously] present in test data point [True]
33 Text feature [compared] present in test data point [True]
34 Text feature [cells] present in test data point [True]
35 Text feature [found] present in test data point [True]
36 Text feature [shown] present in test data point [True]
37 Text feature [obtained] present in test data point [True]
38 Text feature [enhanced] present in test data point [True]
39 Text feature [also] present in test data point [True]
40 Text feature [however] present in test data point [True]
41 Text feature [suggest] present in test data point [True]
42 Text feature [observed] present in test data point [True]
43 Text feature [10] present in test data point [True]
44 Text feature [activation] present in test data point [True]
45 Text feature [higher] present in test data point [True]
46 Text feature [1a] present in test data point [True]
47 Text feature [treated] present in test data point [True]
48 Text feature [similar] present in test data point [True]
49 Text feature [factor] present in test data point [True]
50 Text feature [mutations] present in test data point [True]
51 Text feature [inhibition] present in test data point [True]
52 Text feature [growth] present in test data point [True]
53 Text feature [may] present in test data point [True]
54 Text feature [described] present in test data point [True]
55 Text feature [interestingly] present in test data point [True]
56 Text feature [respectively] present in test data point [True]
57 Text feature [addition] present in test data point [True]
58 Text feature [without] present in test data point [True]
```

```
59 Text feature [molecular] present in test data point [True]
60 Text feature [new] present in test data point [True]
61 Text feature [including] present in test data point [True]
62 Text feature [studies] present in test data point [True]
63 Text feature [identified] present in test data point [True]
64 Text feature [hours] present in test data point [True]
66 Text feature [approximately] present in test data point [True]
67 Text feature [using] present in test data point [True]
68 Text feature [total] present in test data point [True]
70 Text feature [increased] present in test data point [True]
71 Text feature [figure] present in test data point [True]
72 Text feature [various] present in test data point [True]
73 Text feature [reported] present in test data point [True]
74 Text feature [followed] present in test data point [True]
77 Text feature [3a] present in test data point [True]
78 Text feature [occur] present in test data point [True]
79 Text feature [report] present in test data point [True]
80 Text feature [furthermore] present in test data point [True]
81 Text feature [although] present in test data point [True]
82 Text feature [different] present in test data point [True]
83 Text feature [suggesting] present in test data point [True]
84 Text feature [proliferation] present in test data point [True]
86 Text feature [leading] present in test data point [True]
88 Text feature [whereas] present in test data point [True]
89 Text feature [1b] present in test data point [True]
90 Text feature [recent] present in test data point [True]
91 Text feature [measured] present in test data point [True]
93 Text feature [confirm] present in test data point [True]
94 Text feature [serum] present in test data point [True]
95 Text feature [demonstrated] present in test data point [True]
96 Text feature [three] present in test data point [True]
97 Text feature [active] present in test data point [True]
98 Text feature [within] present in test data point [True]
99 Text feature [inhibitors] present in test data point [True]
Out of the top  100  features  72 are present in query point
```

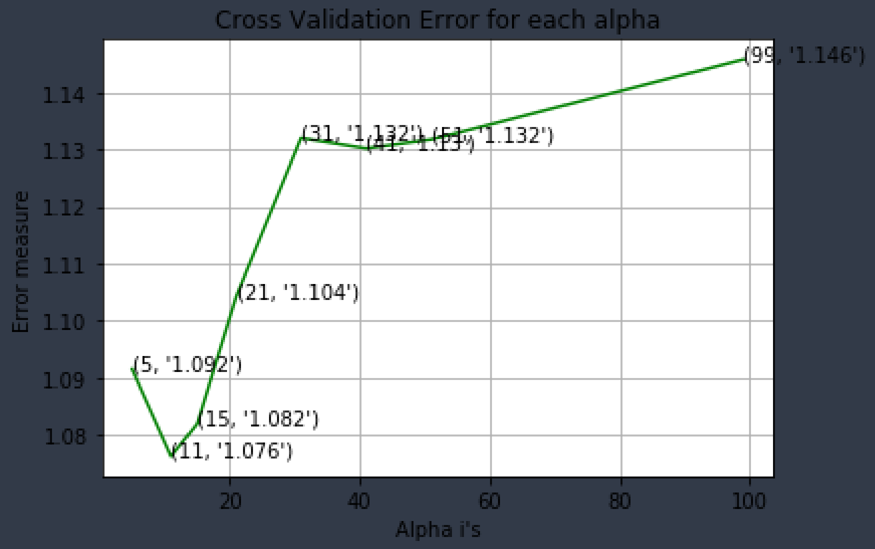
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [63]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/gen
erated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
 p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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



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



alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    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",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0915141863557163
for alpha = 11
Log Loss : 1.0762154185069404
for alpha = 15
Log Loss : 1.0819586099996104
for alpha = 21
Log Loss : 1.10423728629933
for alpha = 31
Log Loss : 1.131985738538
for alpha = 41
Log Loss : 1.1301598960132477
for alpha = 51
Log Loss : 1.1317179373827864

for alpha = 99
```

```
Log Loss : 1.145802056830009
```



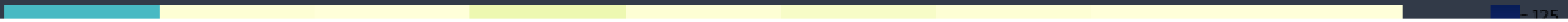
```
For values of best alpha =  11 The train log loss is: 0.6314169517736006
For values of best alpha =  11 The cross validation log loss is: 1.0762154185069404
For values of best alpha =  11 The test log loss is: 1.0163893373522823
```

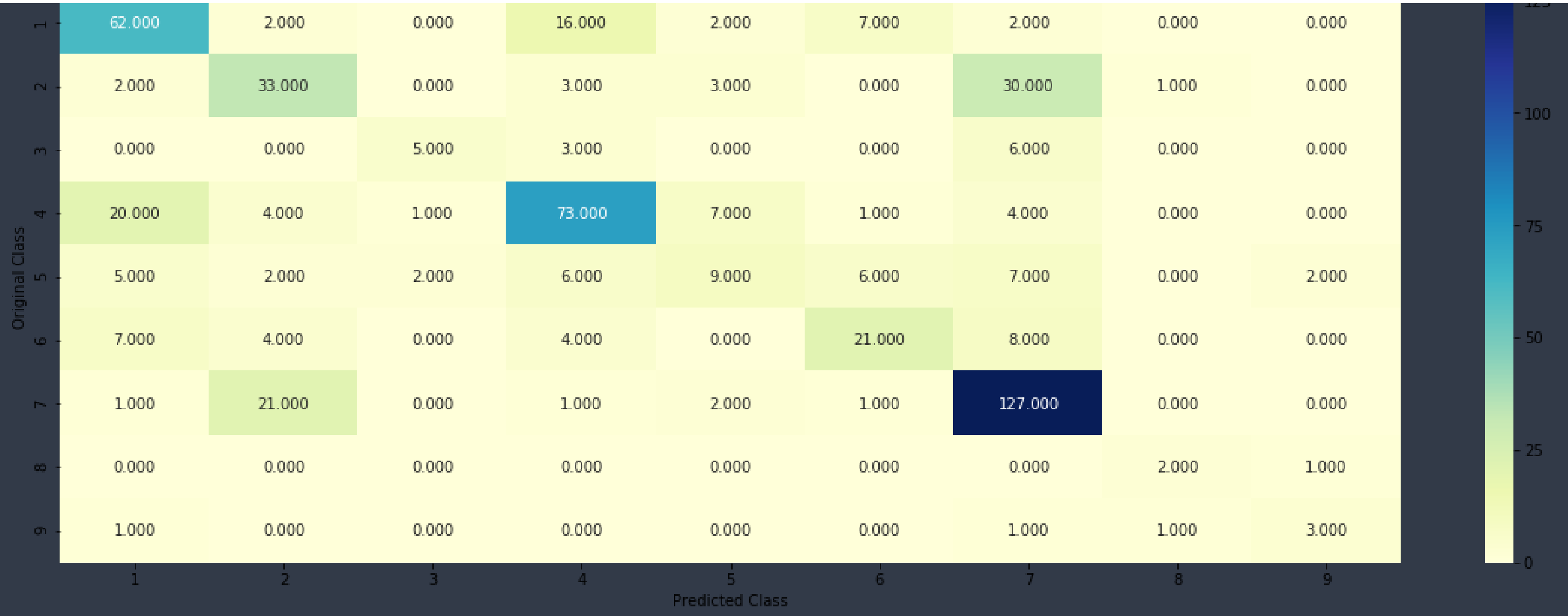
4.2.2. Testing the model with best hyper paramters

In [64]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/gen
erated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
 p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-
nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding,
cv_y, clf)
```

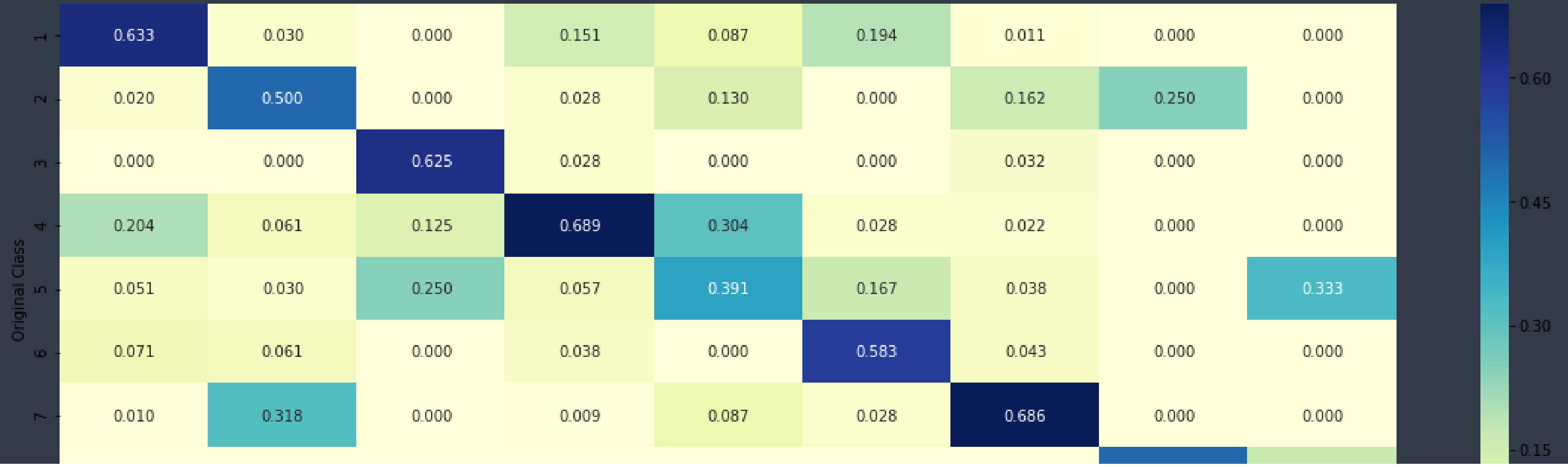
```
Log loss : 1.0762154185069404
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



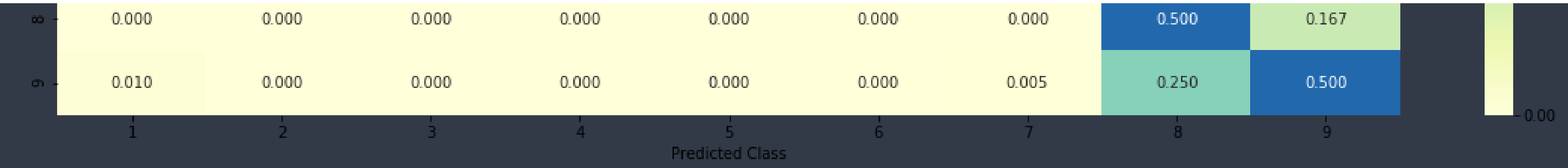


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	62.000	2.000	0.000	16.000	2.000	7.000	2.000	0.000	0.000
2	2.000	33.000	0.000	3.000	3.000	0.000	30.000	1.000	0.000
3	0.000	0.000	5.000	3.000	0.000	0.000	6.000	0.000	0.000
4	20.000	4.000	1.000	73.000	7.000	1.000	4.000	0.000	0.000
5	5.000	2.000	2.000	6.000	9.000	6.000	7.000	0.000	2.000
6	7.000	4.000	0.000	4.000	0.000	21.000	8.000	0.000	0.000
7	1.000	21.000	0.000	1.000	2.000	1.000	127.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	3.000

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

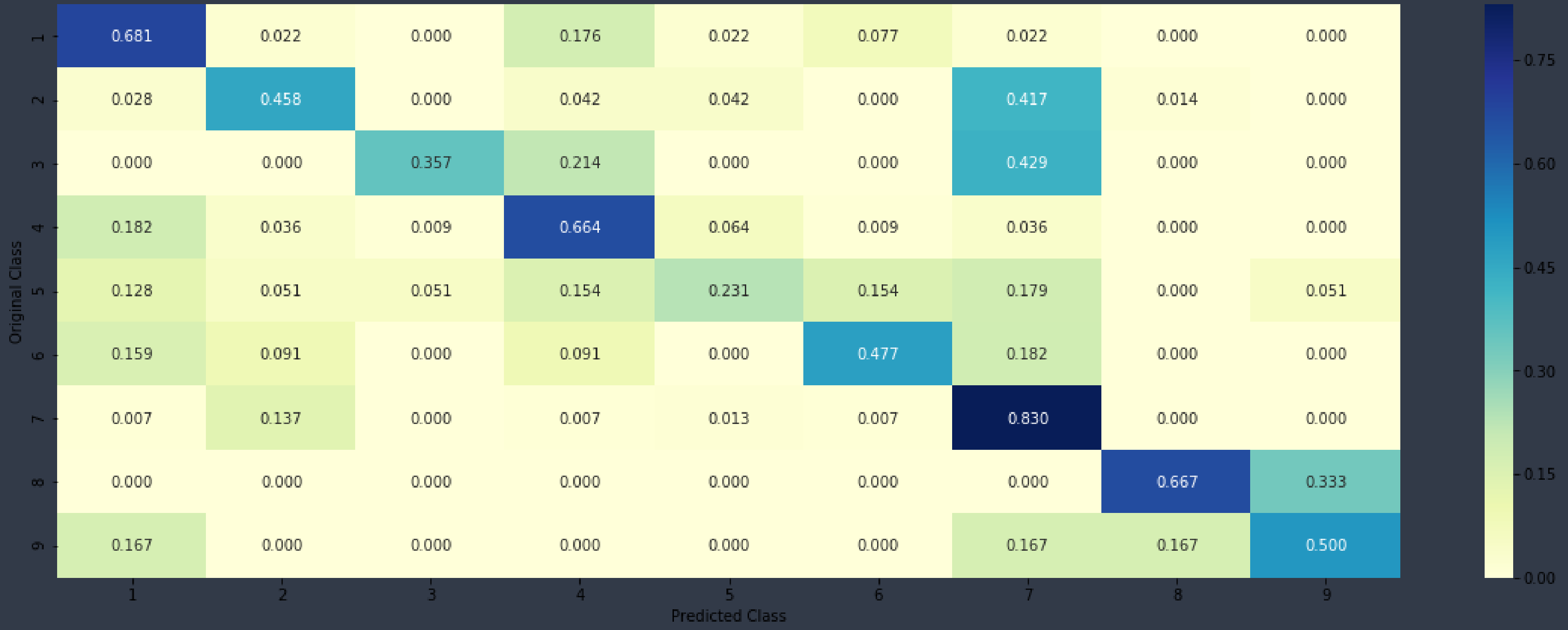


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.633	0.030	0.000	0.151	0.087	0.194	0.011	0.000	0.000
2	0.020	0.500	0.000	0.028	0.130	0.000	0.162	0.250	0.000
3	0.000	0.000	0.625	0.028	0.000	0.000	0.032	0.000	0.000
4	0.204	0.061	0.125	0.689	0.304	0.028	0.022	0.000	0.000
5	0.051	0.030	0.250	0.057	0.391	0.167	0.038	0.000	0.333
6	0.071	0.061	0.000	0.038	0.000	0.583	0.043	0.000	0.000
7	0.010	0.318	0.000	0.009	0.087	0.028	0.686	0.000	0.000



	1	2	3	4	5	6	7	8	9
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.500	0.167
9	0.010	0.000	0.000	0.000	0.000	0.000	0.005	0.250	0.500

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



	1	2	3	4	5	6	7	8	9
1	0.681	0.022	0.000	0.176	0.022	0.077	0.022	0.000	0.000
2	0.028	0.458	0.000	0.042	0.042	0.000	0.417	0.014	0.000
3	0.000	0.000	0.357	0.214	0.000	0.000	0.429	0.000	0.000
4	0.182	0.036	0.009	0.664	0.064	0.009	0.036	0.000	0.000
5	0.128	0.051	0.051	0.154	0.231	0.154	0.179	0.000	0.051
6	0.159	0.091	0.000	0.091	0.000	0.477	0.182	0.000	0.000
7	0.007	0.137	0.000	0.007	0.013	0.007	0.830	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.667	0.333
9	0.167	0.000	0.000	0.000	0.000	0.000	0.167	0.167	0.500

4.2.3.Sample Query point -1

In [65]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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 classe
s",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 1
The  11  nearest neighbours of the test points belongs to classes [1 1 1 4 1 1 3 1 1 5 1]
Fequency of nearest points : Counter({1: 8, 3: 1, 4: 1, 5: 1})
```

4.2.4. Sample Query Point-2

In [66]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv = None)
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100
```

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

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [4 4 4 4 1 4 4 4 1 3 2]
Fequency of nearest points : Counter({4: 7, 1: 2, 2: 1, 3: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

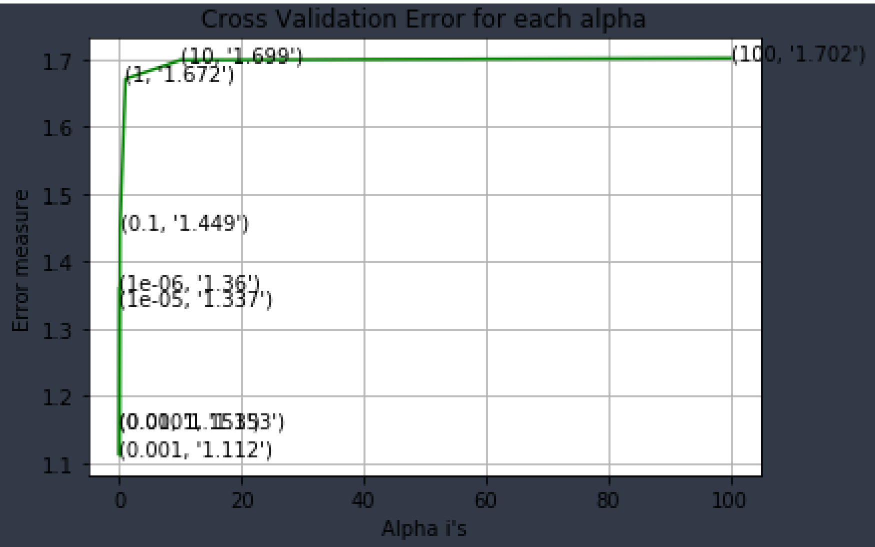
In [67]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
```

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

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



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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
```

```
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    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",cv=None)
```

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

```
for alpha = 1e-06
Log Loss : 1.359708524387313
for alpha = 1e-05
Log Loss : 1.3369311091684417
for alpha = 0.0001
Log Loss : 1.1534922241687542
for alpha = 0.001
Log Loss : 1.1115439140669414
for alpha = 0.01
Log Loss : 1.1533336843091526
for alpha = 0.1
Log Loss : 1.4494447193180873
for alpha = 1
Log Loss : 1.6717483455468505
for alpha = 10
Log Loss : 1.6992865507091743
for alpha = 100
Log Loss : 1.7021617847310466
```

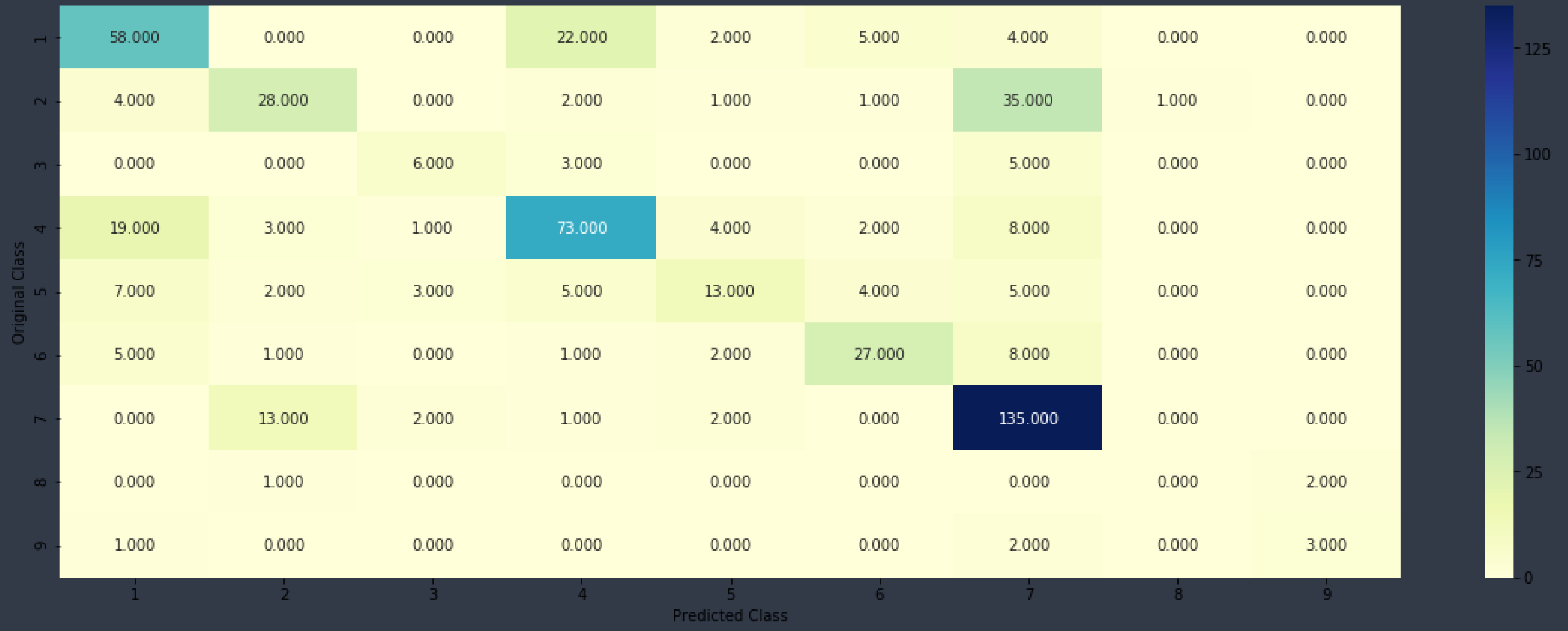


```
For values of best alpha =  0.001 The train log loss is: 0.5803577697459831
For values of best alpha =  0.001 The cross validation log loss is: 1.1115439140669414
For values of best alpha =  0.001 The test log loss is: 1.0322990425247833
```

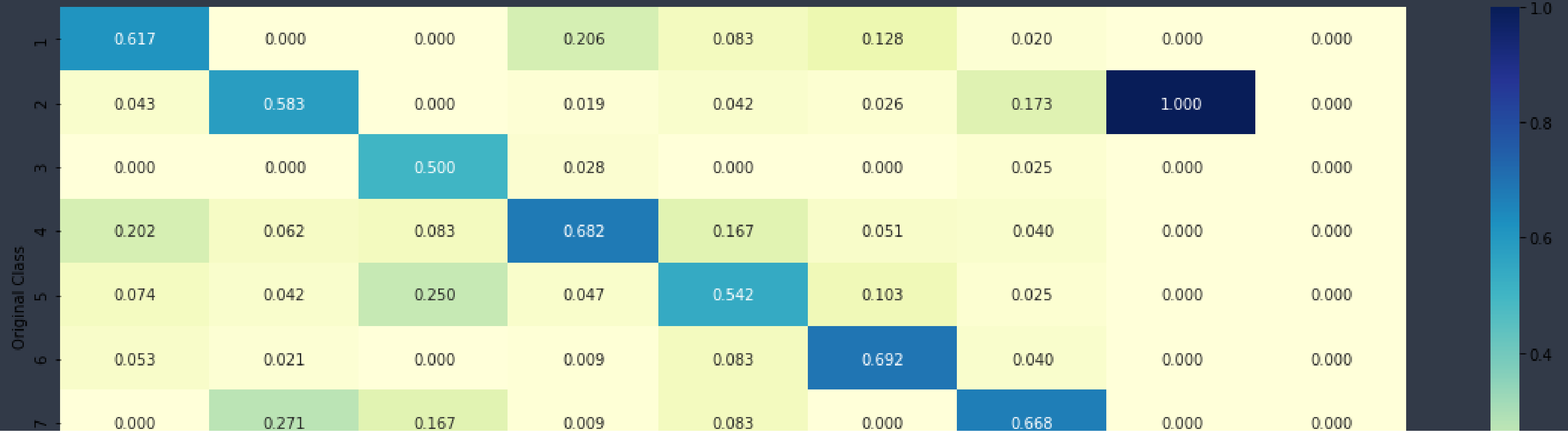
4.3.1.2. Testing the model with best hyper paramters

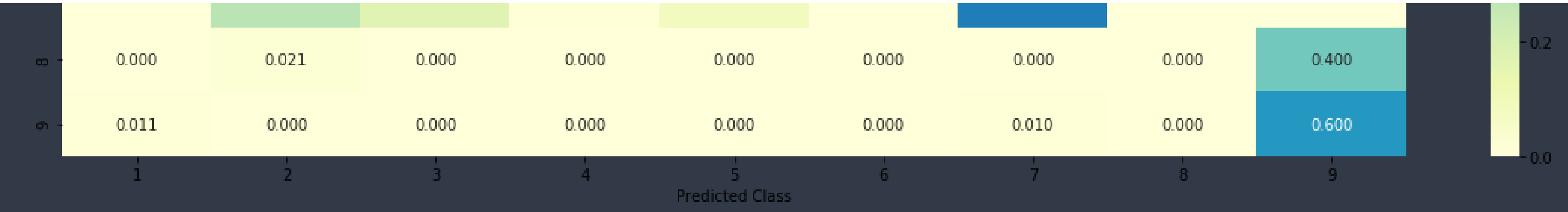
In [68]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/ge
ometric-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 : 1.1115439140669414
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```

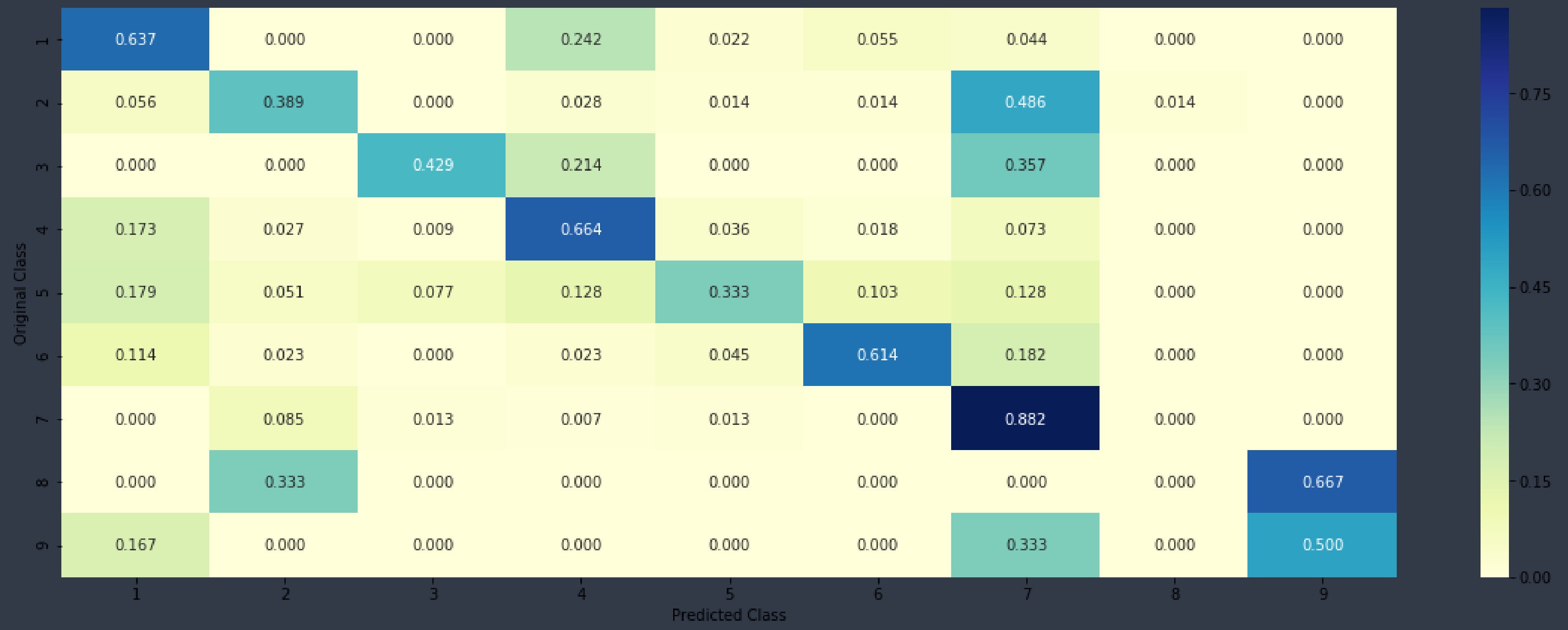


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.1.3. Feature Importance

```
In [69]: 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 [70]:
```
# 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_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6475 0.0866 0.0086 0.1298 0.0303 0.0181 0.069  0.0059 0.0041]]
Actual Class : 1
--------------------------------------------------
159 Text feature [ortholog] present in test data point [True]
175 Text feature [processing] present in test data point [True]
197 Text feature [archaebacterial] present in test data point [True]
218 Text feature [tgs] present in test data point [True]
233 Text feature [paralog] present in test data point [True]
319 Text feature [tnrc6b] present in test data point [True]
322 Text feature [deletion] present in test data point [True]
338 Text feature [hooks] present in test data point [True]
342 Text feature [piwi] present in test data point [True]
369 Text feature [ago1] present in test data point [True]
376 Text feature [ago] present in test data point [True]
384 Text feature [hook] present in test data point [True]
388 Text feature [silencing] present in test data point [True]
494 Text feature [derepresses] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [71]:
```
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_onehotCodi
ng[test_point_index]),4))
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2881 0.0839 0.0086 0.3845 0.0304 0.0177 0.1707 0.0099 0.0062]]
Actual Class : 4
--------------------------------------------------
170 Text feature [suppressor] present in test data point [True]
309 Text feature [trevigen] present in test data point [True]
314 Text feature [degradation] present in test data point [True]
346 Text feature [neighbouring] present in test data point [True]
439 Text feature [microscopy] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

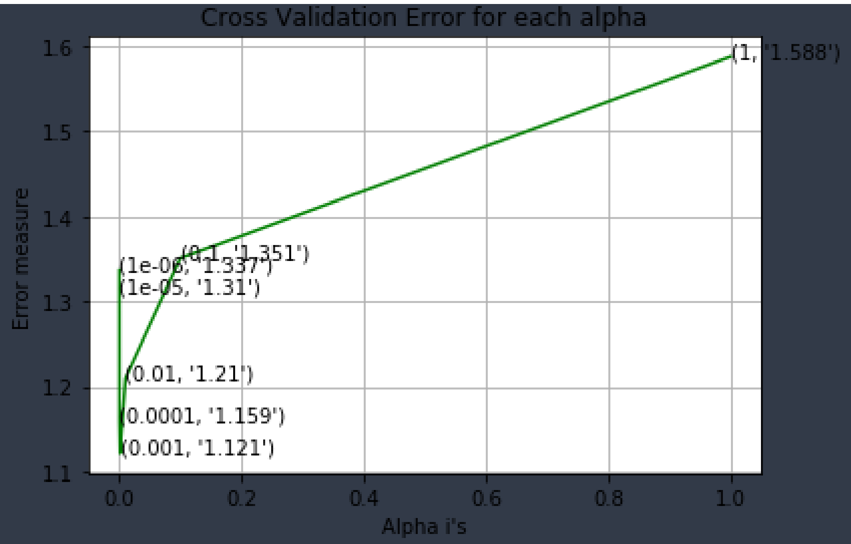
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [72]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t Descent.
# predict(X)    Predict class labels for samples in X.

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




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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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",cv = None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.3369093929150098
for alpha = 1e-05
Log Loss : 1.310120958016182
for alpha = 0.0001
Log Loss : 1.158916218951122
for alpha = 0.001
Log Loss : 1.1213418687589178
for alpha = 0.01
Log Loss : 1.2098805187133874
for alpha = 0.1
Log Loss : 1.3511788593887577
for alpha = 1
Log Loss : 1.5878648635479533
```

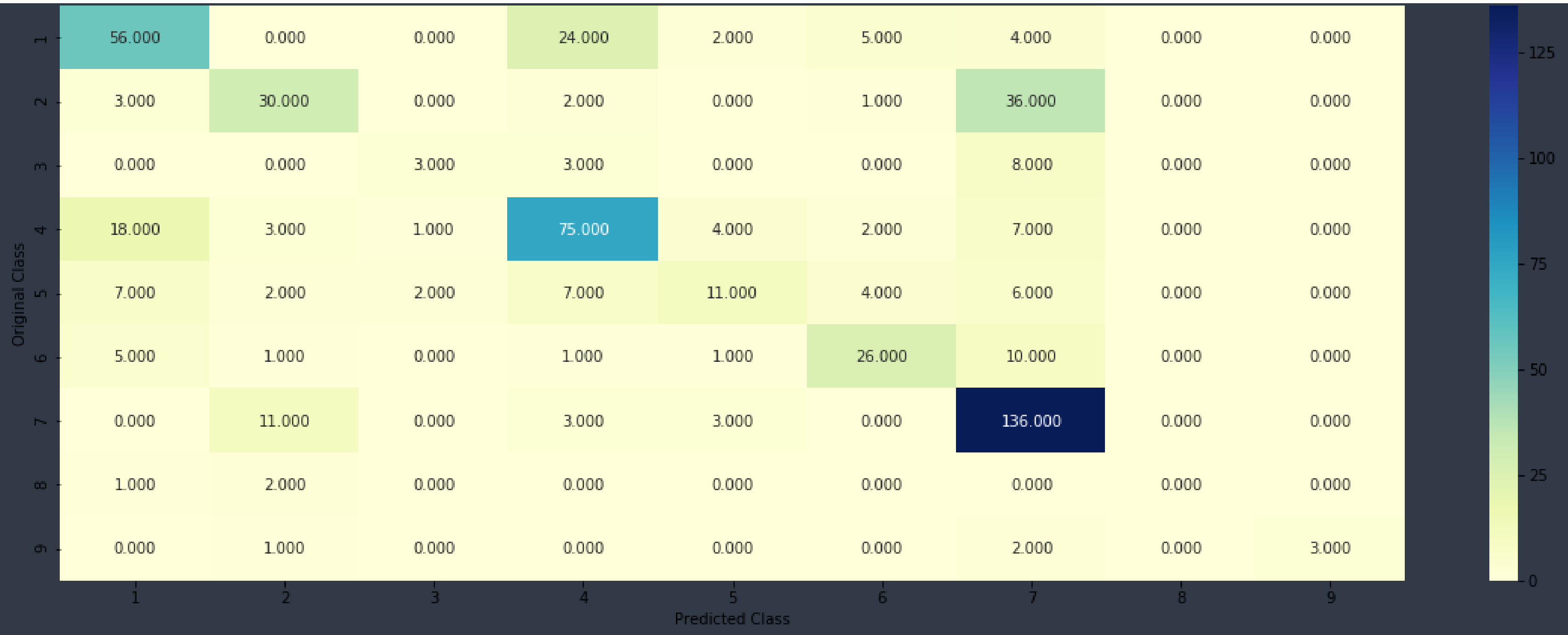


```
For values of best alpha =  0.001 The train log loss is: 0.5805831049211583
For values of best alpha =  0.001 The cross validation log loss is: 1.1213418687589178
For values of best alpha =  0.001 The test log loss is: 1.0494926777133984
```

4.3.2.2. Testing model with best hyper parameters

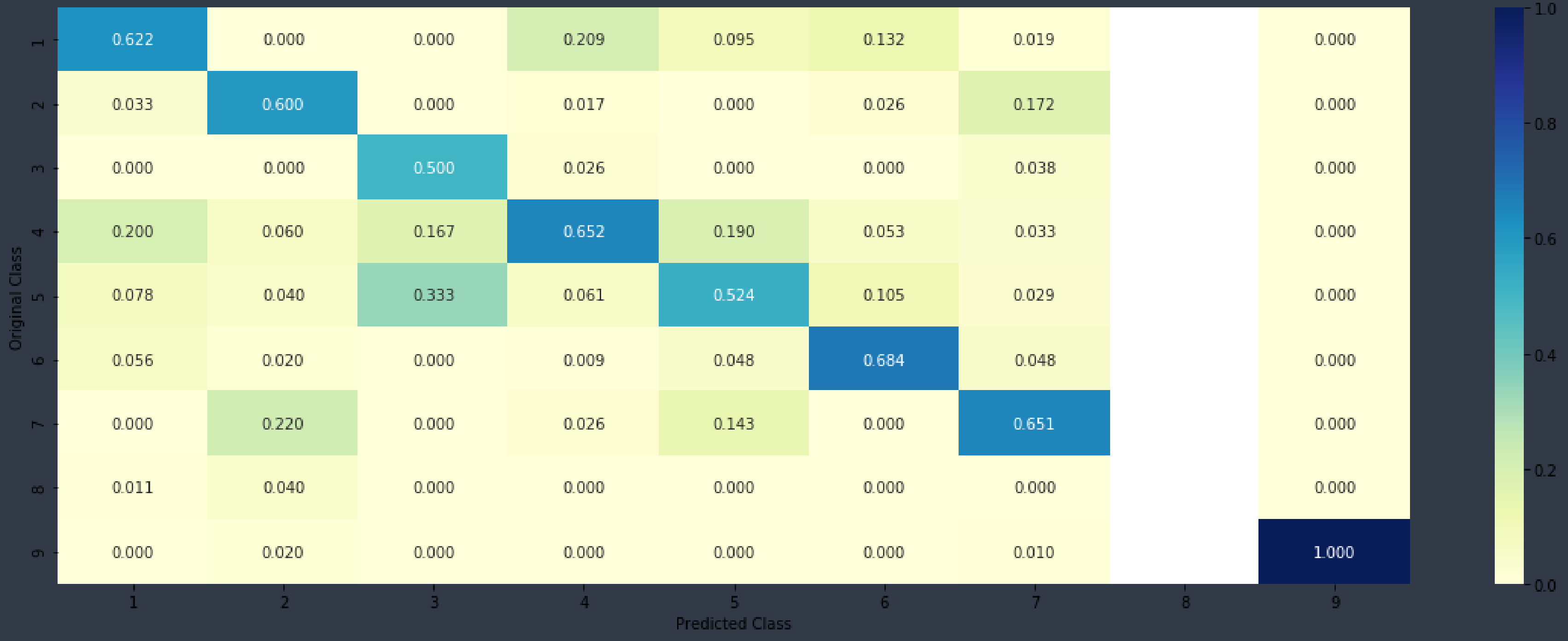
In [73]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t 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 : 1.1213418687589178
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

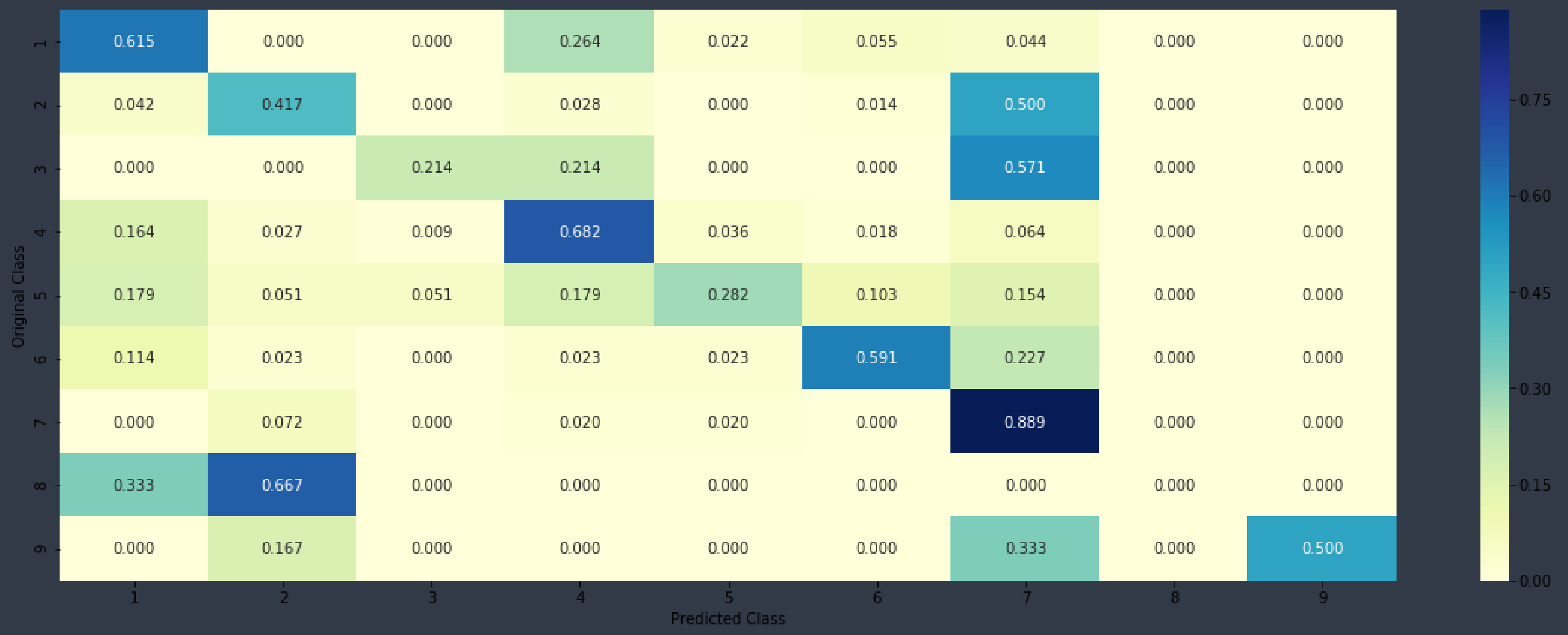


	1	2	3	4	5	6	7	8	9
1	56.000	0.000	0.000	24.000	2.000	5.000	4.000	0.000	0.000
2	3.000	30.000	0.000	2.000	0.000	1.000	36.000	0.000	0.000
3	0.000	0.000	3.000	3.000	0.000	0.000	8.000	0.000	0.000
4	18.000	3.000	1.000	75.000	4.000	2.000	7.000	0.000	0.000
5	7.000	2.000	2.000	7.000	11.000	4.000	6.000	0.000	0.000
6	5.000	1.000	0.000	1.000	1.000	26.000	10.000	0.000	0.000
7	0.000	11.000	0.000	3.000	3.000	0.000	136.000	0.000	0.000
8	1.000	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	1.000	0.000	0.000	0.000	0.000	2.000	0.000	3.000

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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [74]:

```
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_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6483 0.0888 0.0089 0.1281 0.0314 0.0194 0.066  0.0059 0.0032]]
Actual Class : 1
--------------------------------------------------
181 Text feature [ortholog] present in test data point [True]
215 Text feature [processing] present in test data point [True]
219 Text feature [archaebacterial] present in test data point [True]
240 Text feature [paralog] present in test data point [True]
242 Text feature [tgs] present in test data point [True]
332 Text feature [tnrc6b] present in test data point [True]
345 Text feature [hooks] present in test data point [True]
349 Text feature [piwi] present in test data point [True]
376 Text feature [ago1] present in test data point [True]
388 Text feature [ago] present in test data point [True]
392 Text feature [hook] present in test data point [True]
397 Text feature [deletion] present in test data point [True]
409 Text feature [silencing] present in test data point [True]
Out of the top  500  features  13 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [75]:

```
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_onehotCodi
ng[test_point_index]),4))
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2805 0.0812 0.008  0.4054 0.0285 0.017  0.1668 0.0095 0.0033]]
Actual Class : 4
--------------------------------------------------
216 Text feature [suppressor] present in test data point [True]
321 Text feature [neighbouring] present in test data point [True]
368 Text feature [trevigen] present in test data point [True]
422 Text feature [degradation] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.4. Linear Support Vector Machines

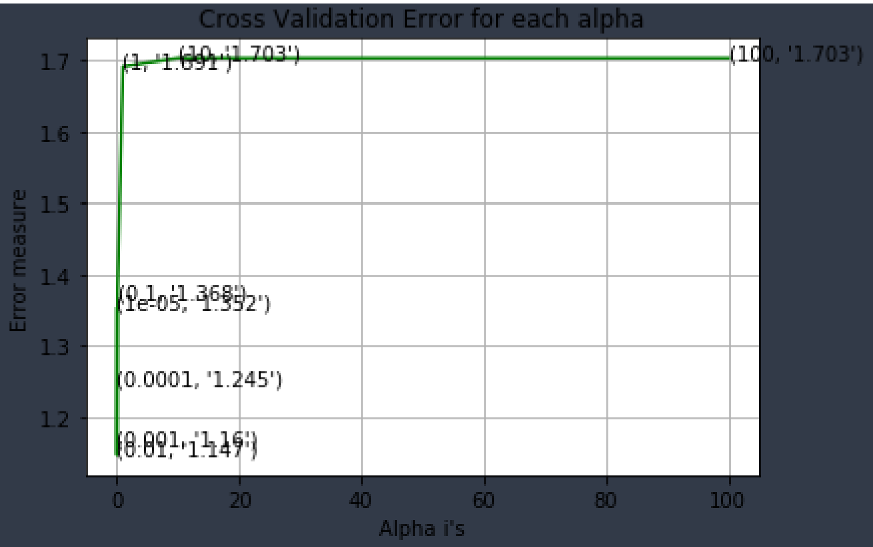
4.4.1. Hyper paramter tuning

In [76]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.o
rg/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabilit
y=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/ma
thematical-derivation-copy-8/
# --------------------------------




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

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

```
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', r
andom_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv= None)
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-
15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.3524569724749946
for C = 0.0001
Log Loss : 1.2450746542152122
for C = 0.001
Log Loss : 1.1600525511390811
for C = 0.01
Log Loss : 1.146523550332717
for C = 0.1
Log Loss : 1.367641778353358
for C = 1
Log Loss : 1.691373569950243
for C = 10
Log Loss : 1.7027183144245908
for C = 100
Log Loss : 1.7026663780978135
```



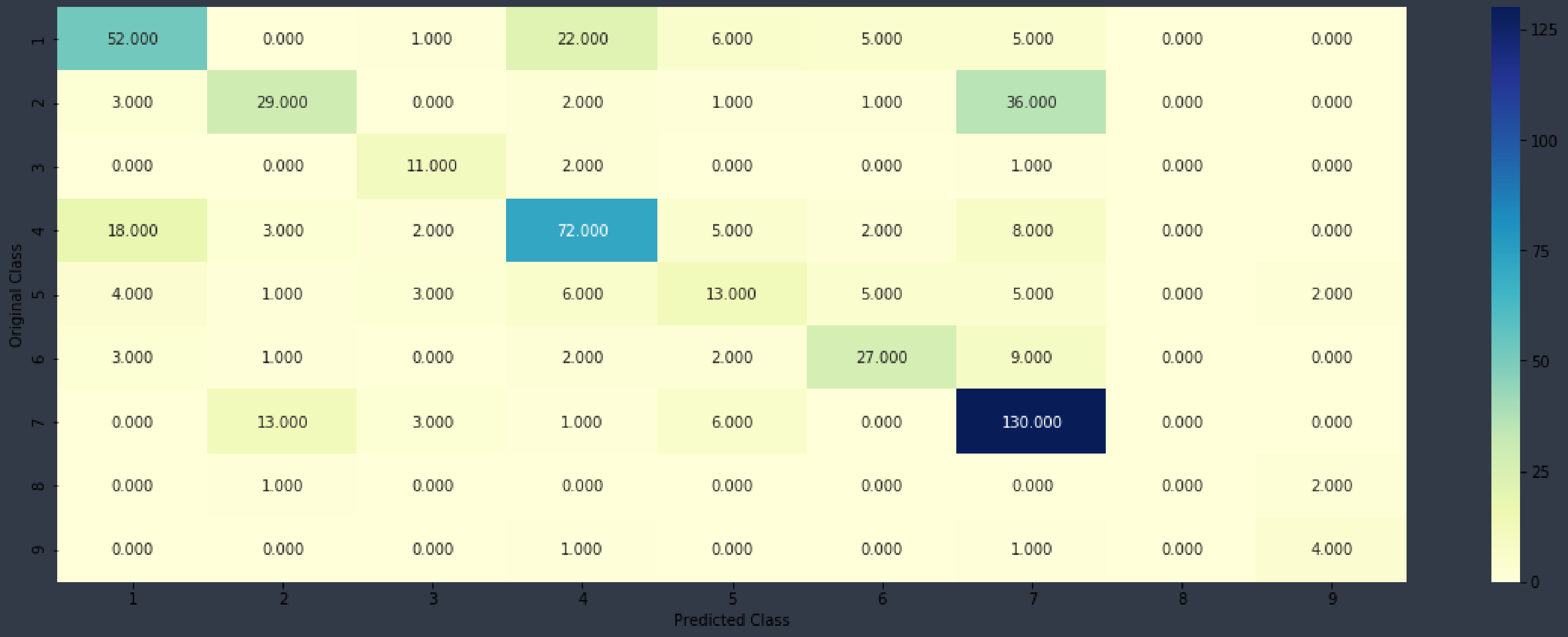
```
For values of best alpha =  0.01 The train log loss is: 0.7712014933162398
For values of best alpha =  0.01 The cross validation log loss is: 1.146523550332717
For values of best alpha =  0.01 The test log loss is: 1.0992427668689382
```

4.4.2. Testing model with best hyper parameters

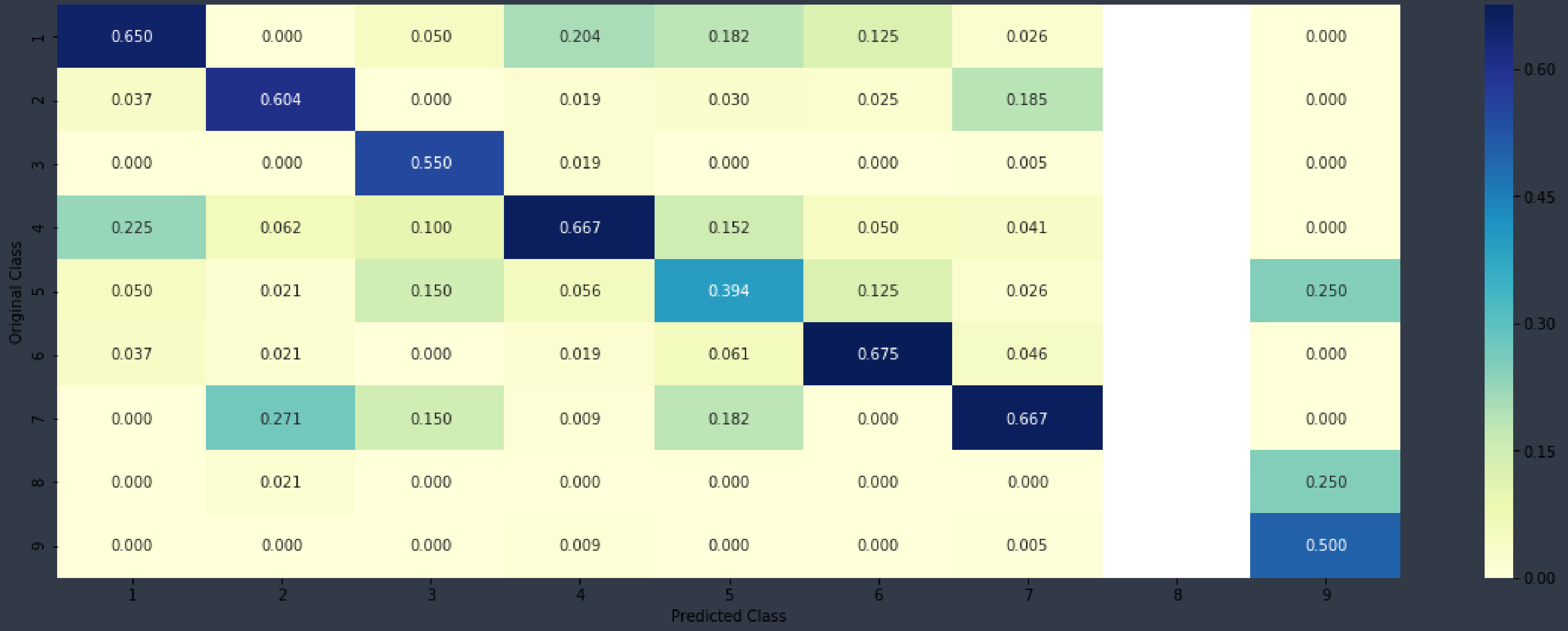
In [77]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.o
rg/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabilit
y=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/ma
thematical-derivation-copy-8/
# --------------------------------


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

```
Log loss : 1.146523550332717
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```

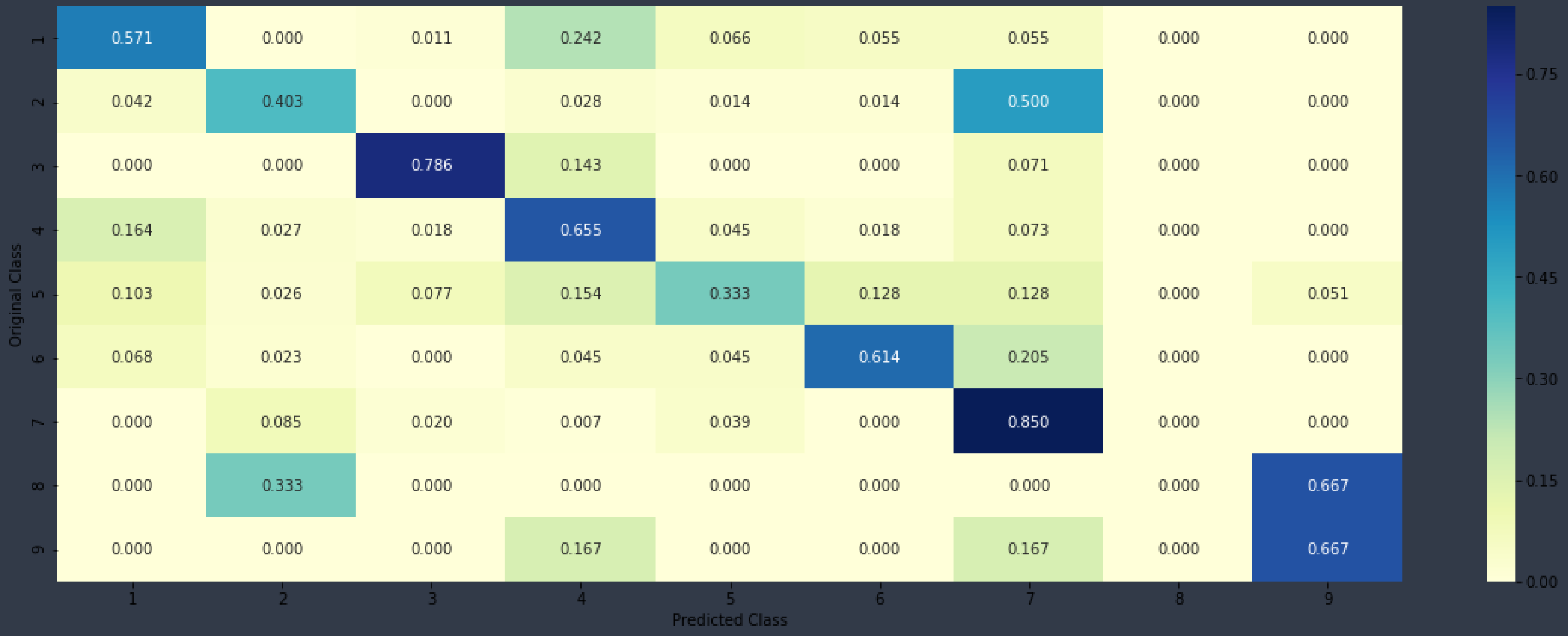


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.650	0.000	0.050	0.204	0.182	0.125	0.026		0.000
2	0.037	0.604	0.000	0.019	0.030	0.025	0.185		0.000
3	0.000	0.000	0.550	0.019	0.000	0.000	0.005		0.000
4	0.225	0.062	0.100	0.667	0.152	0.050	0.041		0.000
5	0.050	0.021	0.150	0.056	0.394	0.125	0.026		0.250
6	0.037	0.021	0.000	0.019	0.061	0.675	0.046		0.000
7	0.000	0.271	0.150	0.009	0.182	0.000	0.667		0.000
8	0.000	0.021	0.000	0.000	0.000	0.000	0.000		0.250
9	0.000	0.000	0.000	0.009	0.000	0.000	0.005		0.500

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



	1	2	3	4	5	6	7	8	9
1	0.571	0.000	0.011	0.242	0.066	0.055	0.055	0.000	0.000
2	0.042	0.403	0.000	0.028	0.014	0.014	0.500	0.000	0.000
3	0.000	0.000	0.786	0.143	0.000	0.000	0.071	0.000	0.000
4	0.164	0.027	0.018	0.655	0.045	0.018	0.073	0.000	0.000
5	0.103	0.026	0.077	0.154	0.333	0.128	0.128	0.000	0.051
6	0.068	0.023	0.000	0.045	0.045	0.614	0.205	0.000	0.000
7	0.000	0.085	0.020	0.007	0.039	0.000	0.850	0.000	0.000
8	0.000	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.667
9	0.000	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.667

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [78]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42
)
```

```python
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_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.599  0.0907 0.0115 0.1321 0.0388 0.0265 0.0919 0.0053 0.0043]]
Actual Class : 1
--------------------------------------------------
12 Text feature [archaebacterial] present in test data point [True]
39 Text feature [ortholog] present in test data point [True]
40 Text feature [tgs] present in test data point [True]
53 Text feature [tnrc6b] present in test data point [True]
60 Text feature [piwi] present in test data point [True]
67 Text feature [processing] present in test data point [True]
69 Text feature [ago] present in test data point [True]
75 Text feature [paralog] present in test data point [True]
79 Text feature [ago1] present in test data point [True]
106 Text feature [hooks] present in test data point [True]
111 Text feature [derepresses] present in test data point [True]
118 Text feature [silencing] present in test data point [True]
144 Text feature [a260] present in test data point [True]
149 Text feature [hook] present in test data point [True]
155 Text feature [a280] present in test data point [True]
191 Text feature [awg] present in test data point [True]
195 Text feature [trnas] present in test data point [True]
199 Text feature [reestablish] present in test data point [True]
```

```
224 Text feature [derepression] present in test data point [True]
292 Text feature [deletion] present in test data point [True]
343 Text feature [mimicry] present in test data point [True]
Out of the top  500  features  21 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [79]:
```python
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_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2255 0.1016 0.0168 0.2968 0.052  0.0347 0.2574 0.0075 0.0077]]
Actual Class : 4
--------------------------------------------------
89 Text feature [microscopy] present in test data point [True]
94 Text feature [suppressor] present in test data point [True]
118 Text feature [trevigen] present in test data point [True]
420 Text feature [degradation] present in test data point [True]
442 Text feature [phosphatases] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [80]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
one, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
 verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
om_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv =None)
        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_erro
r_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', ma
x_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",cv =None)
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 i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

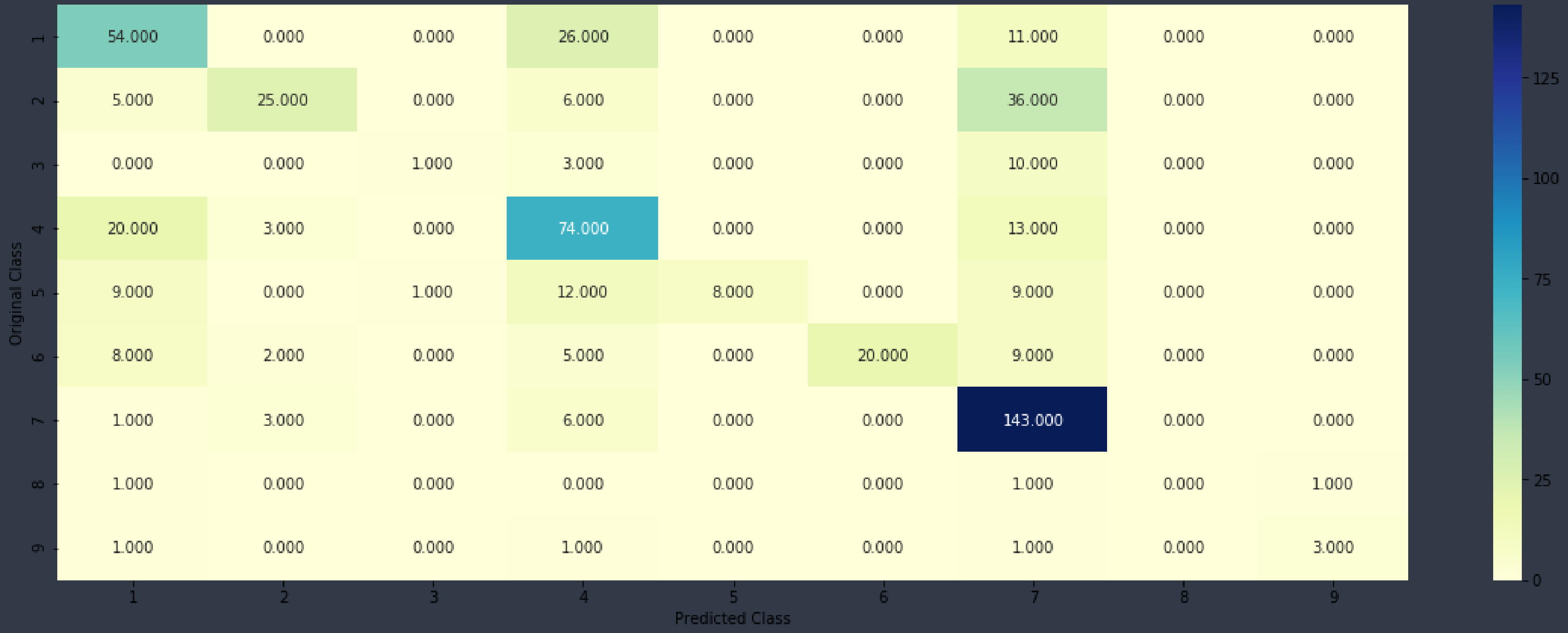
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2688416724170735
for n_estimators = 100 and max depth =  10
```

```
Log Loss : 1.2087727655909308
for n_estimators = 200 and max depth =  5
Log Loss : 1.258595206386008
for n_estimators = 200 and max depth =  10
Log Loss : 1.1950033841946577
for n_estimators = 500 and max depth =  5
Log Loss : 1.243850557415404
for n_estimators = 500 and max depth =  10
Log Loss : 1.1866806202017914
for n_estimators = 1000 and max depth =  5
Log Loss : 1.246225714225061
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1845336170641814
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2457997810987553
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1850105655127776
For values of best estimator =  1000 The train log loss is: 0.7158043669972709
For values of best estimator =  1000 The cross validation log loss is: 1.1845336170641814
For values of best estimator =  1000 The test log loss is: 1.1604813292237788
```

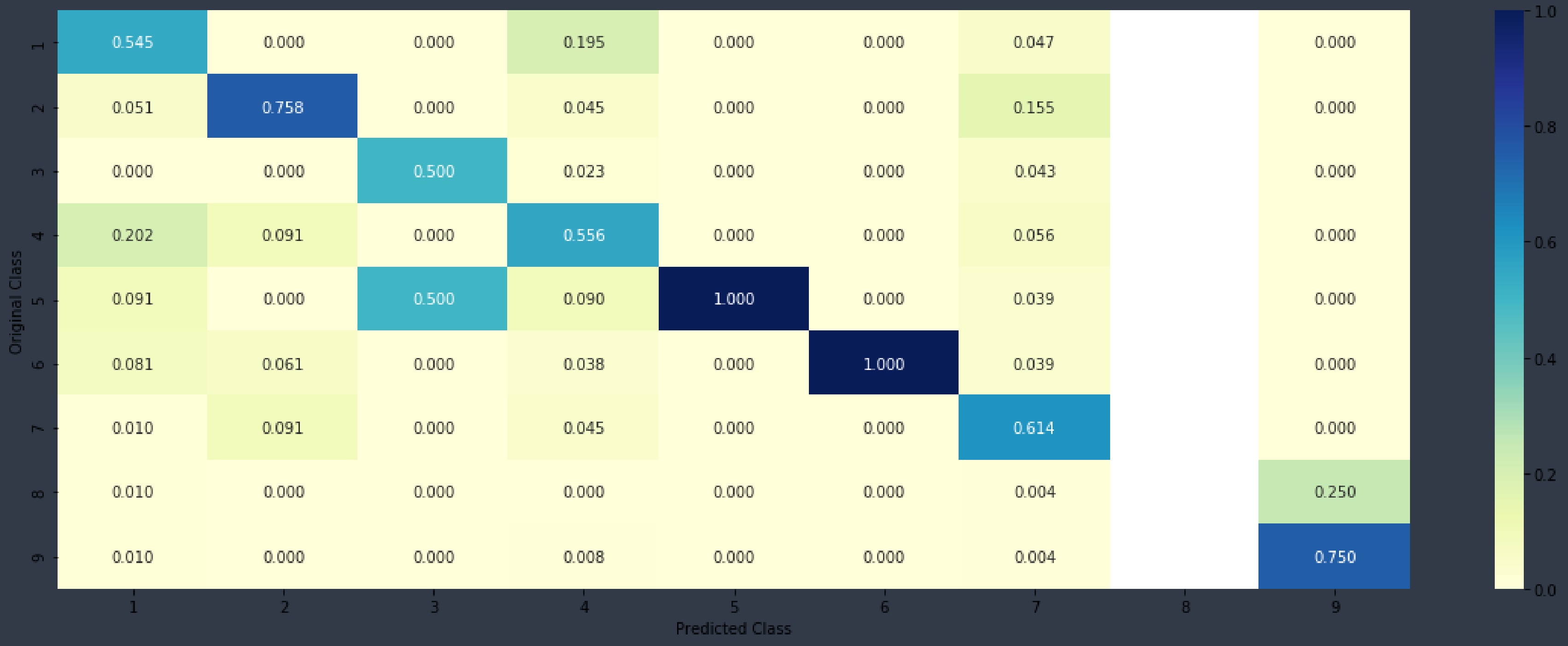
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [81]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_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 : 1.1845336170641814
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```

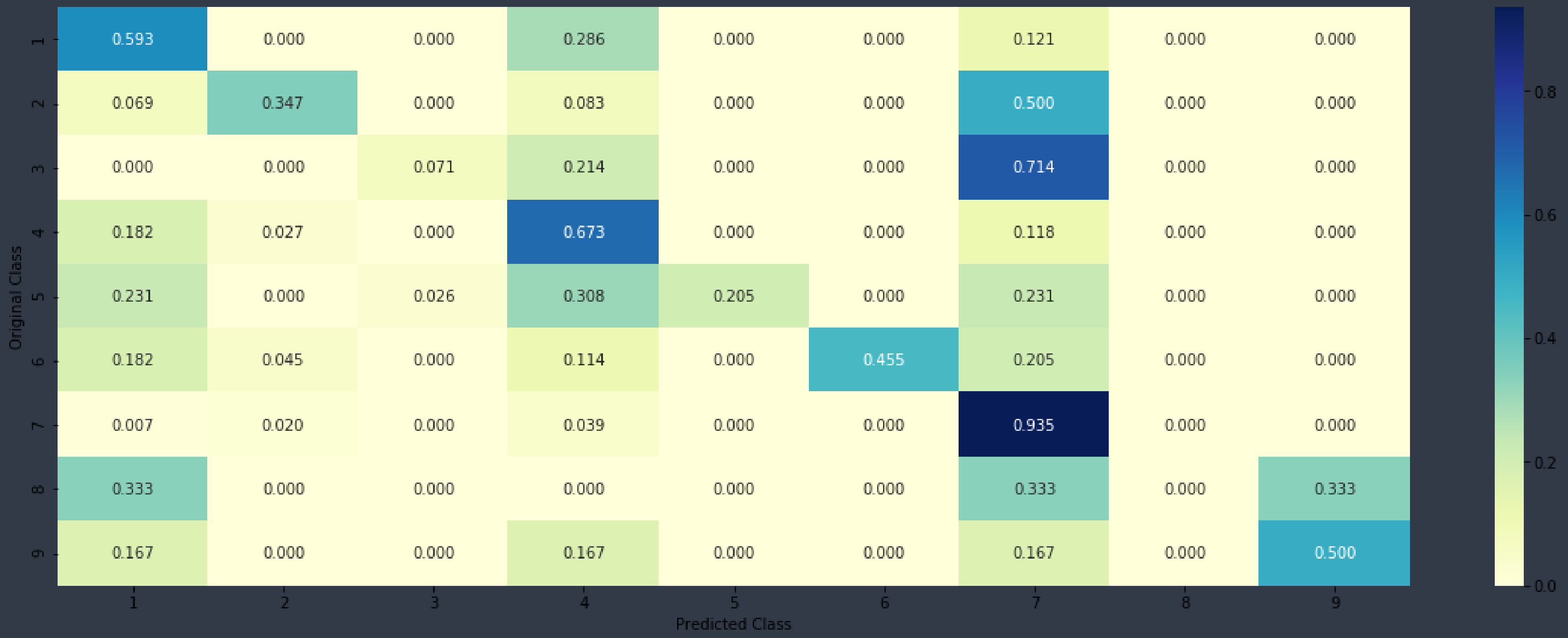


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.545	0.000	0.000	0.195	0.000	0.000	0.047		0.000
2	0.051	0.758	0.000	0.045	0.000	0.000	0.155		0.000
3	0.000	0.000	0.500	0.023	0.000	0.000	0.043		0.000
4	0.202	0.091	0.000	0.556	0.000	0.000	0.056		0.000
5	0.091	0.000	0.500	0.090	1.000	0.000	0.039		0.000
6	0.081	0.061	0.000	0.038	0.000	1.000	0.039		0.000
7	0.010	0.091	0.000	0.045	0.000	0.000	0.614		0.000
8	0.010	0.000	0.000	0.000	0.000	0.000	0.004		0.250
9	0.010	0.000	0.000	0.008	0.000	0.000	0.004		0.750

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



	1	2	3	4	5	6	7	8	9
1	0.593	0.000	0.000	0.286	0.000	0.000	0.121	0.000	0.000
2	0.069	0.347	0.000	0.083	0.000	0.000	0.500	0.000	0.000
3	0.000	0.000	0.071	0.214	0.000	0.000	0.714	0.000	0.000
4	0.182	0.027	0.000	0.673	0.000	0.000	0.118	0.000	0.000
5	0.231	0.000	0.026	0.308	0.205	0.000	0.231	0.000	0.000
6	0.182	0.045	0.000	0.114	0.000	0.455	0.205	0.000	0.000
7	0.007	0.020	0.000	0.039	0.000	0.000	0.935	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.333
9	0.167	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.500

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [82]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
```

```python
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_onehotCodi
ng[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4913 0.0605 0.0194 0.2511 0.058  0.0528 0.0509 0.0075 0.0084]]
Actual Class : 1
--------------------------------------------------
2 Text feature [tyrosine] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
27 Text feature [function] present in test data point [True]
30 Text feature [cells] present in test data point [True]
35 Text feature [functional] present in test data point [True]
36 Text feature [loss] present in test data point [True]
37 Text feature [yeast] present in test data point [True]
47 Text feature [cell] present in test data point [True]
64 Text feature [lines] present in test data point [True]
66 Text feature [nuclear] present in test data point [True]
68 Text feature [transformation] present in test data point [True]
72 Text feature [inhibition] present in test data point [True]
79 Text feature [expression] present in test data point [True]
84 Text feature [presence] present in test data point [True]
```

```
86 Text feature [dna] present in test data point [True]
99 Text feature [independent] present in test data point [True]
Out of the top  100  features  17 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [83]:

```
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_onehotCodi
ng[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1812 0.0986 0.024  0.1854 0.0545 0.0474 0.3924 0.0076 0.0089]]
Actuall Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
7 Text feature [suppressor] present in test data point [True]
8 Text feature [inhibitors] present in test data point [True]
9 Text feature [activated] present in test data point [True]
10 Text feature [missense] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
15 Text feature [signaling] present in test data point [True]
17 Text feature [oncogenic] present in test data point [True]
```

```
20 Text feature [receptor] present in test data point [True]
24 Text feature [growth] present in test data point [True]
27 Text feature [function] present in test data point [True]
28 Text feature [downstream] present in test data point [True]
29 Text feature [transforming] present in test data point [True]
30 Text feature [cells] present in test data point [True]
35 Text feature [functional] present in test data point [True]
36 Text feature [loss] present in test data point [True]
38 Text feature [constitutively] present in test data point [True]
41 Text feature [ras] present in test data point [True]
43 Text feature [response] present in test data point [True]
47 Text feature [cell] present in test data point [True]
49 Text feature [phospho] present in test data point [True]
51 Text feature [extracellular] present in test data point [True]
53 Text feature [proliferation] present in test data point [True]
54 Text feature [stability] present in test data point [True]
61 Text feature [expressing] present in test data point [True]
63 Text feature [lung] present in test data point [True]
64 Text feature [lines] present in test data point [True]
66 Text feature [nuclear] present in test data point [True]
72 Text feature [inhibition] present in test data point [True]
73 Text feature [treated] present in test data point [True]
77 Text feature [inhibited] present in test data point [True]
79 Text feature [expression] present in test data point [True]
80 Text feature [serum] present in test data point [True]
82 Text feature [defective] present in test data point [True]
83 Text feature [receptors] present in test data point [True]
84 Text feature [presence] present in test data point [True]
86 Text feature [dna] present in test data point [True]
99 Text feature [independent] present in test data point [True]
Out of the top  100  features  42 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [84]:

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

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

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

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



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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
om_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_erro
r_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', ma
x_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2262522292101115
for n_estimators = 10 and max depth =  3
Log Loss : 1.8880784512904003
for n_estimators = 10 and max depth =  5
Log Loss : 1.5714491153626275
for n_estimators = 10 and max depth =  10
Log Loss : 1.7326629999496141
for n_estimators = 50 and max depth =  2
```

```
Log Loss : 1.8127536591736855
for n_estimators = 50 and max depth =  3
Log Loss : 1.5681415678395936
for n_estimators = 50 and max depth =  5
Log Loss : 1.4261497291469316
for n_estimators = 50 and max depth =  10
Log Loss : 1.794551887023849
for n_estimators = 100 and max depth =  2
Log Loss : 1.6507421975398098
for n_estimators = 100 and max depth =  3
Log Loss : 1.6100597127674834
for n_estimators = 100 and max depth =  5
Log Loss : 1.3501173032947442
for n_estimators = 100 and max depth =  10
Log Loss : 1.829196441600451
for n_estimators = 200 and max depth =  2
Log Loss : 1.741510653850449
for n_estimators = 200 and max depth =  3
Log Loss : 1.6070235968945639
for n_estimators = 200 and max depth =  5
Log Loss : 1.4136503322445506
for n_estimators = 200 and max depth =  10
Log Loss : 1.8542733543796914
for n_estimators = 500 and max depth =  2
Log Loss : 1.8261744088809844
for n_estimators = 500 and max depth =  3
Log Loss : 1.6261711915680421
for n_estimators = 500 and max depth =  5
Log Loss : 1.432686149945551
for n_estimators = 500 and max depth =  10
Log Loss : 1.8922637436582896
for n_estimators = 1000 and max depth =  2
Log Loss : 1.8175144635668152
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6717267309493578
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4236694372360745
for n_estimators = 1000 and max depth =  10
Log Loss : 1.830948162650952
```

```
For values of best alpha =  100 The train log loss is: 0.05455095461234915
For values of best alpha =  100 The cross validation log loss is: 1.3501173032947442
For values of best alpha =  100 The test log loss is: 1.3050860874683652
```

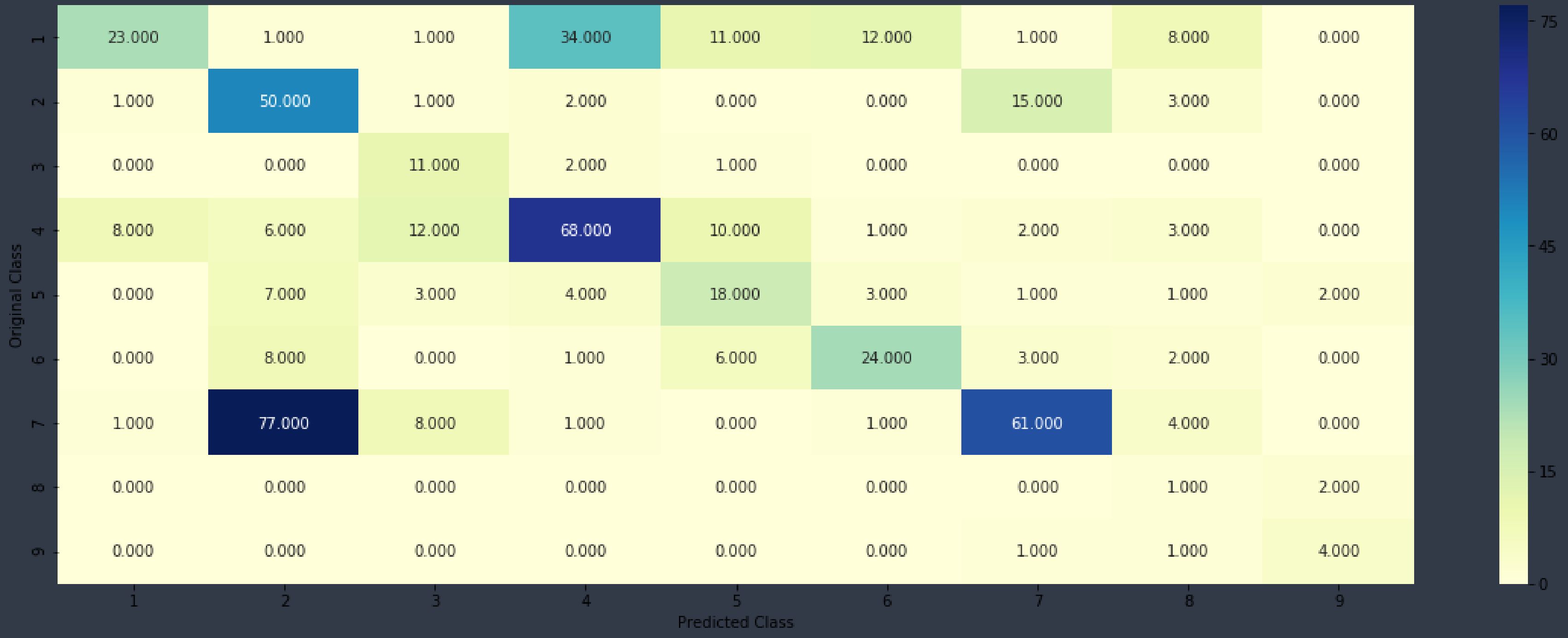
4.5.4. Testing model with best hyper parameters (Response Coding)

In [85]:

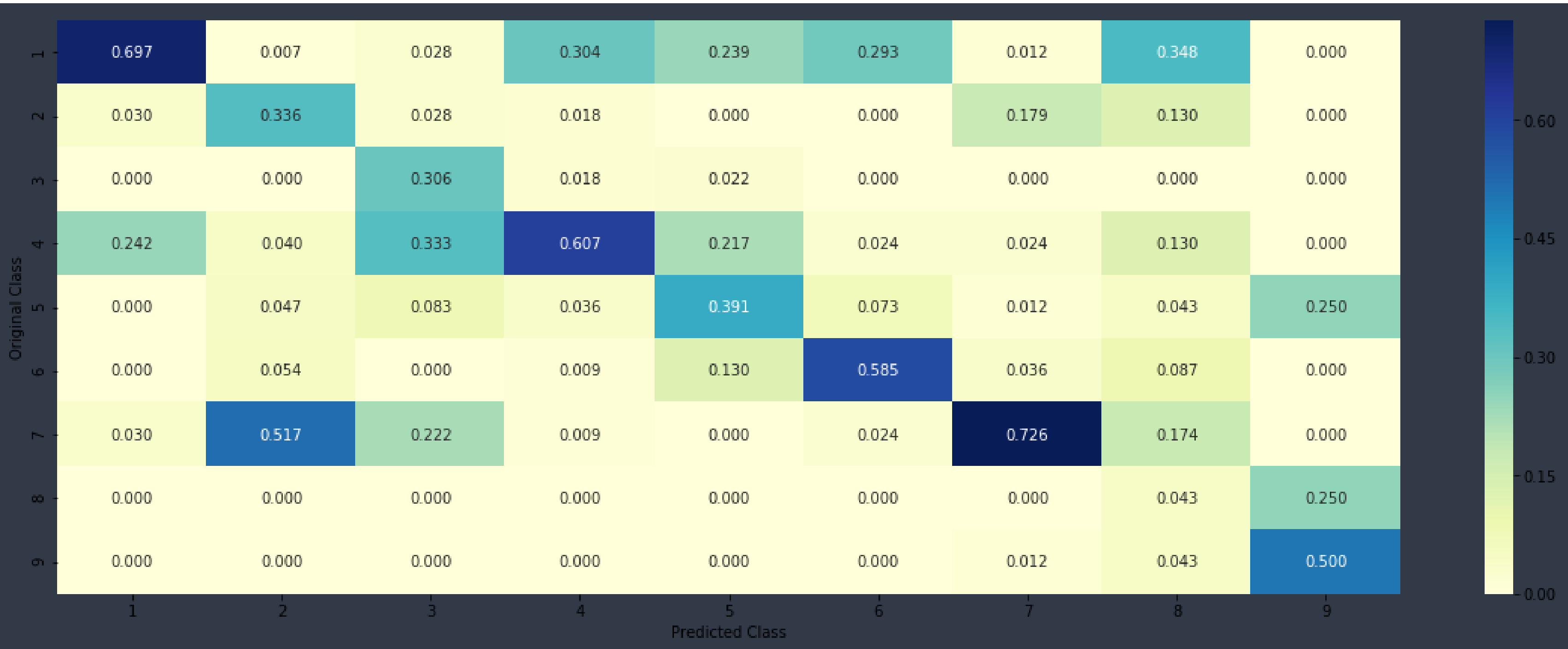
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=N
one, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=
None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.3501173032947444
Number of mis-classified points : 0.5112781954887218
-------------------- Confusion matrix --------------------
```

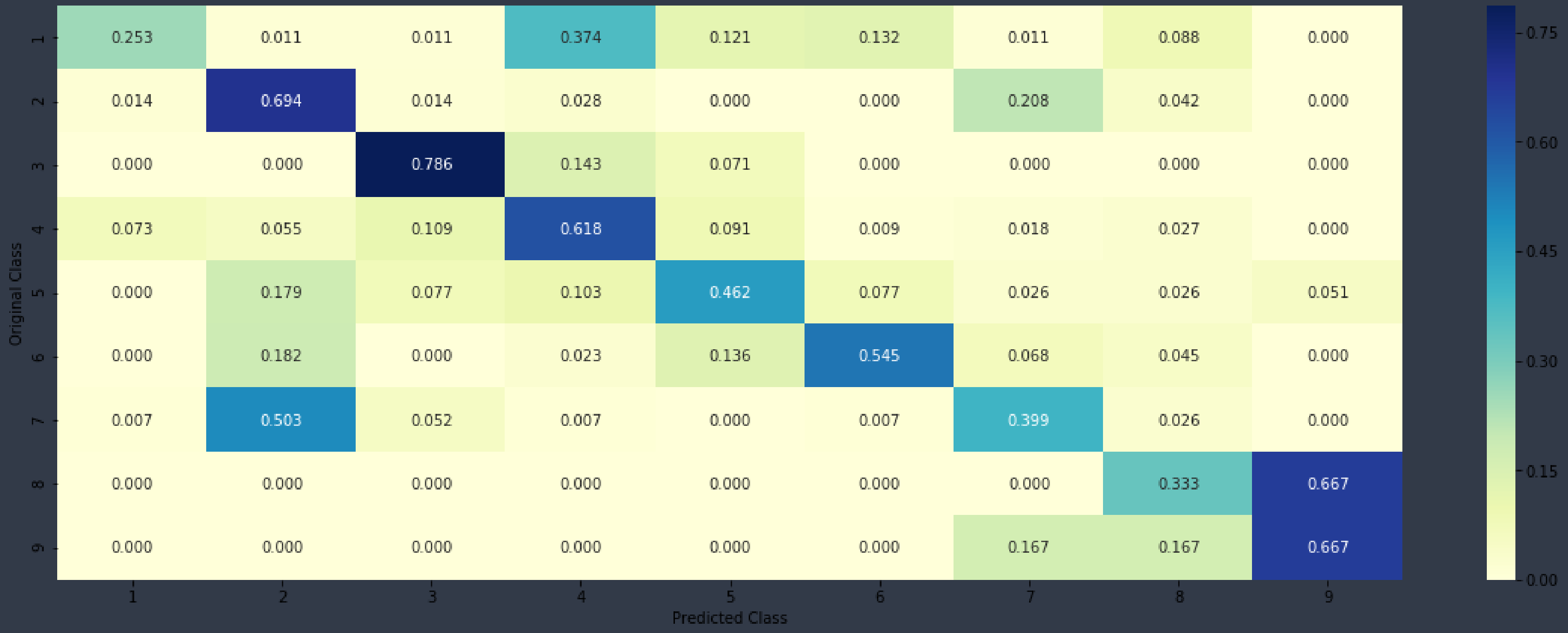


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.697	0.007	0.028	0.304	0.239	0.293	0.012	0.348	0.000
2	0.030	0.336	0.028	0.018	0.000	0.000	0.179	0.130	0.000
3	0.000	0.000	0.306	0.018	0.022	0.000	0.000	0.000	0.000
4	0.242	0.040	0.333	0.607	0.217	0.024	0.024	0.130	0.000
5	0.000	0.047	0.083	0.036	0.391	0.073	0.012	0.043	0.250
6	0.000	0.054	0.000	0.009	0.130	0.585	0.036	0.087	0.000
7	0.030	0.517	0.222	0.009	0.000	0.024	0.726	0.174	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.043	0.250
9	0.000	0.000	0.000	0.000	0.000	0.000	0.012	0.043	0.500

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



	1	2	3	4	5	6	7	8	9
1	0.253	0.011	0.011	0.374	0.121	0.132	0.011	0.088	0.000
2	0.014	0.694	0.014	0.028	0.000	0.000	0.208	0.042	0.000
3	0.000	0.000	0.786	0.143	0.071	0.000	0.000	0.000	0.000
4	0.073	0.055	0.109	0.618	0.091	0.009	0.018	0.027	0.000
5	0.000	0.179	0.077	0.103	0.462	0.077	0.026	0.026	0.051
6	0.000	0.182	0.000	0.023	0.136	0.545	0.068	0.045	0.000
7	0.007	0.503	0.052	0.007	0.000	0.007	0.399	0.026	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667
9	0.000	0.000	0.000	0.000	0.000	0.000	0.167	0.167	0.667

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [86]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', ma
x_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)
```

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2888 0.0717 0.0637 0.1074 0.0381 0.0703 0.0143 0.2146 0.1309]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [87]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCo
ding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2016 0.0225 0.1241 0.4567 0.0408 0.0716 0.0069 0.0343 0.0414]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [88]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sk
learn.linear_model.SGDClassifier.html
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=T
rue, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='opti
mal', 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 Gradien
t Descent.
# predict(X)    Predict class labels for samples in X.

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


# read more about support vector machines with linear kernals here http://scikit-learn.o
rg/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabilit
y=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/ma
thematical-derivation-copy-8/
# --------------------------------



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

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

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).
# ----------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/ra
ndom-forest-and-their-construction-2/
# ----------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', ran
dom_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid",cv=None)

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', rando
m_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid",cv=None)


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)

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_pro
ba(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_oneh
otCoding))))
```

```
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_classifie
r=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_lo
ss(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
```

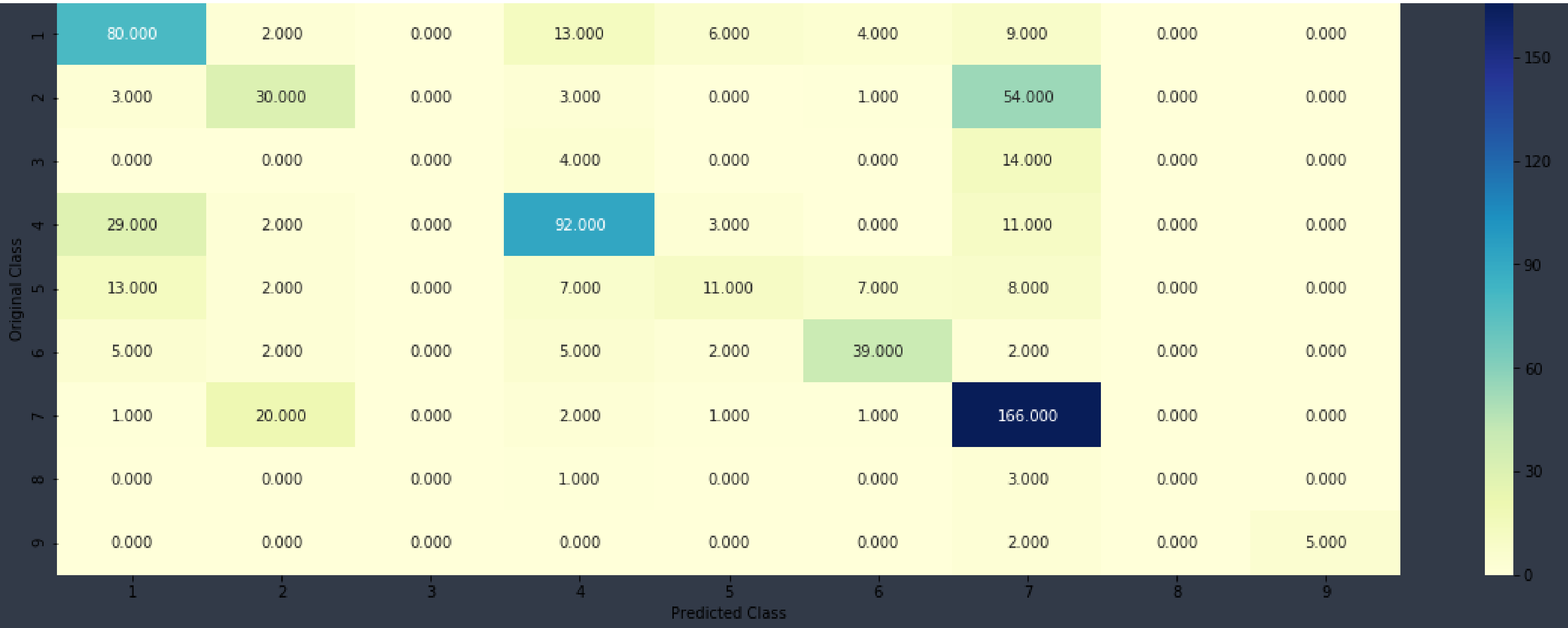
```
Logistic Regression :  Log Loss: 1.10
Support vector machines : Log Loss: 1.69
Naive Bayes : Log Loss: 1.29
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.042
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.534
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.139
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.241
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.552
```

4.7.2 testing the model with the best hyper parameters

In [89]:
```
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)
```

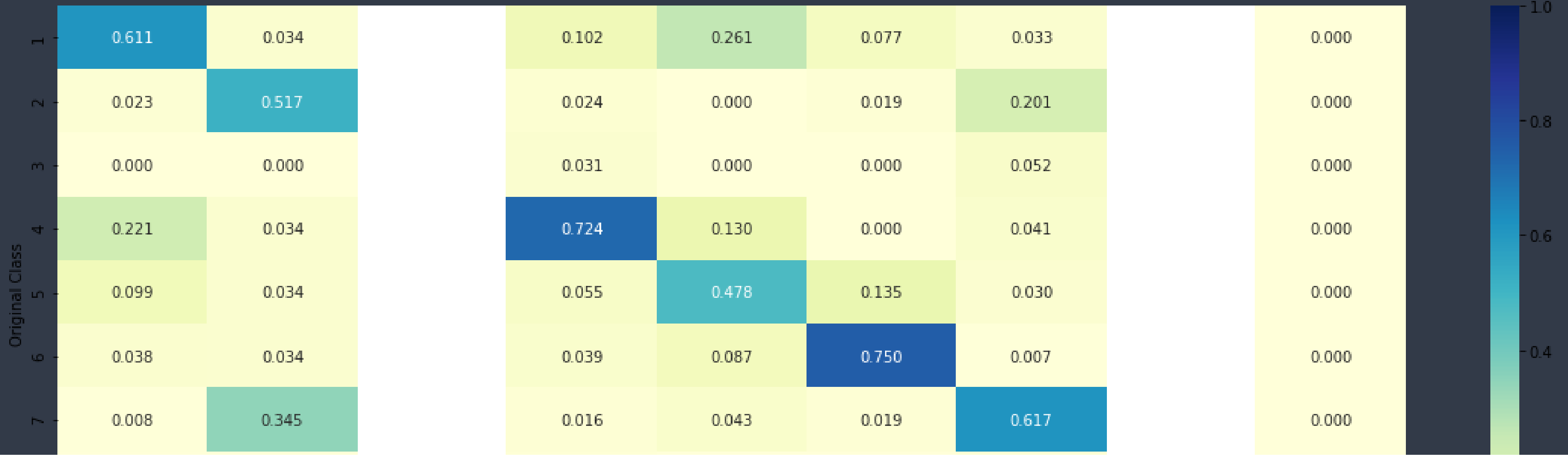
```
log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
print("Log loss (train) on the stacking classifier :",log_error)

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_onehotCo
ding)- 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 : 0.6511543219100392
Log loss (CV) on the stacking classifier : 1.138621819206958
Log loss (test) on the stacking classifier : 1.1158439148954673
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```

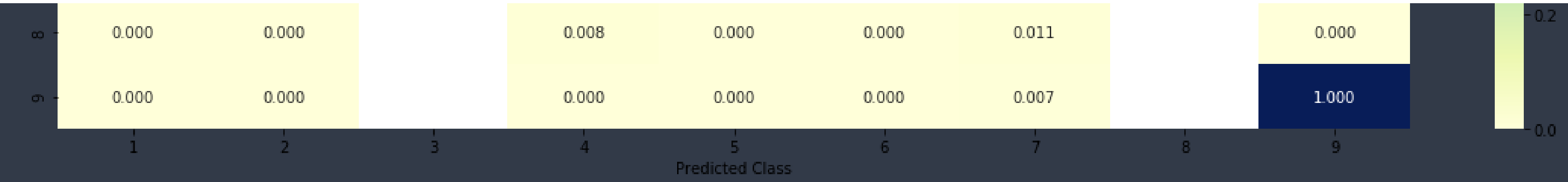


	1	2	3	4	5	6	7	8	9
1	80.000	2.000	0.000	13.000	6.000	4.000	9.000	0.000	0.000
2	3.000	30.000	0.000	3.000	0.000	1.000	54.000	0.000	0.000
3	0.000	0.000	0.000	4.000	0.000	0.000	14.000	0.000	0.000
4	29.000	2.000	0.000	92.000	3.000	0.000	11.000	0.000	0.000
5	13.000	2.000	0.000	7.000	11.000	7.000	8.000	0.000	0.000
6	5.000	2.000	0.000	5.000	2.000	39.000	2.000	0.000	0.000
7	1.000	20.000	0.000	2.000	1.000	1.000	166.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	3.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	5.000

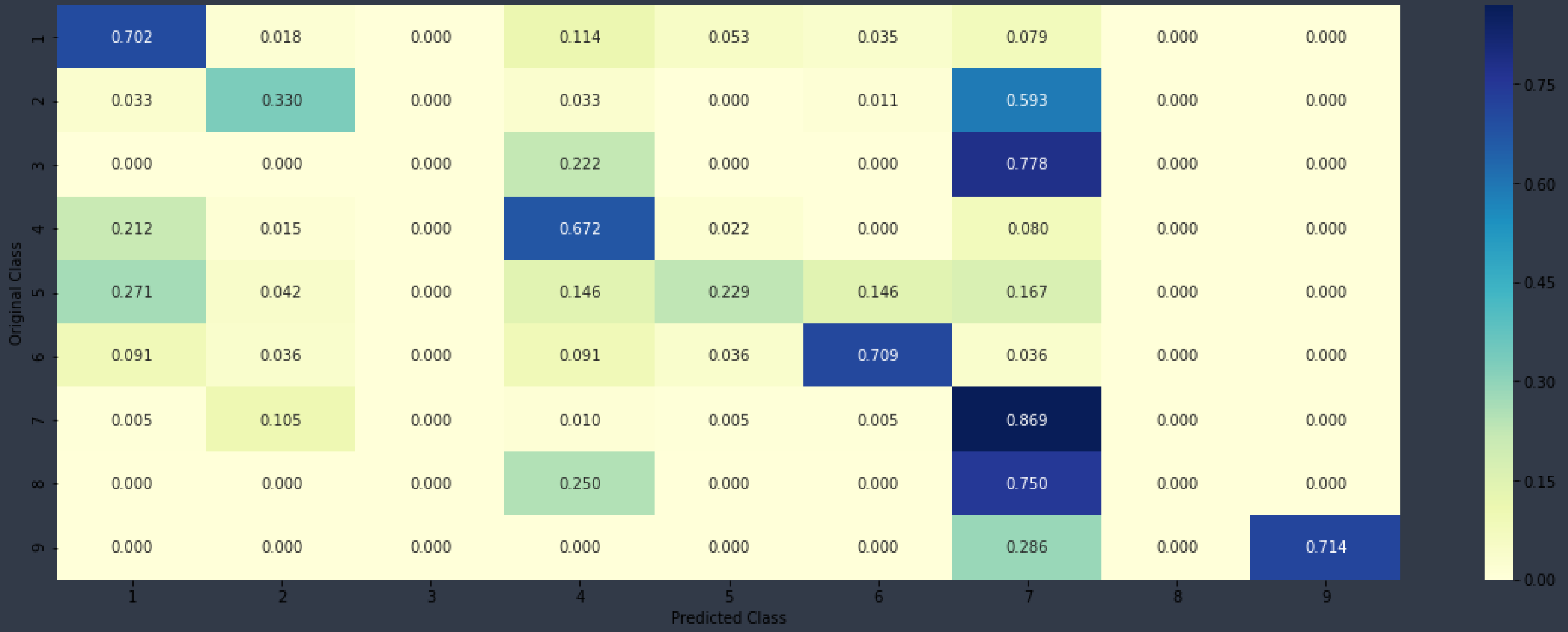
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



1	0.611	0.034		0.102	0.261	0.077	0.033		0.000
2	0.023	0.517		0.024	0.000	0.019	0.201		0.000
3	0.000	0.000		0.031	0.000	0.000	0.052		0.000
4	0.221	0.034		0.724	0.130	0.000	0.041		0.000
5	0.099	0.034		0.055	0.478	0.135	0.030		0.000
6	0.038	0.034		0.039	0.087	0.750	0.007		0.000
7	0.008	0.345		0.016	0.043	0.019	0.617		0.000



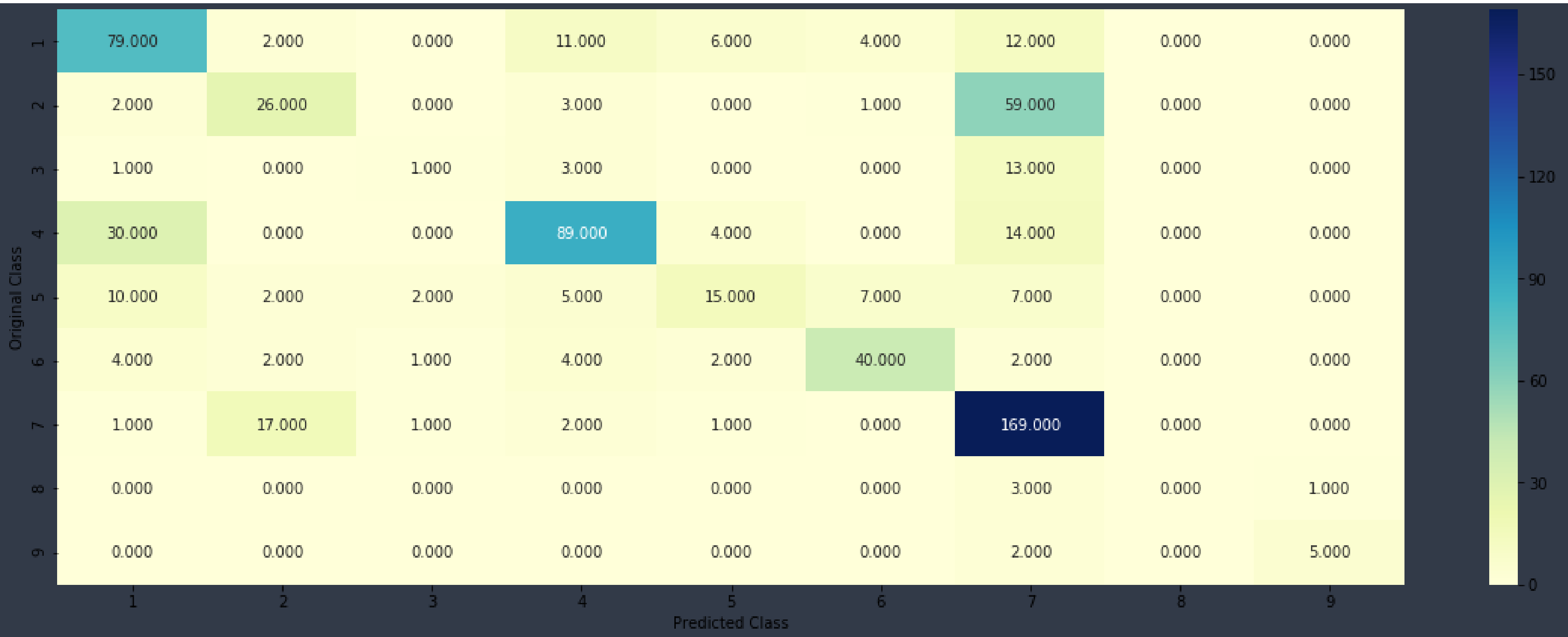
```
-------------------- Recall matrix (Row sum=1) --------------------
```



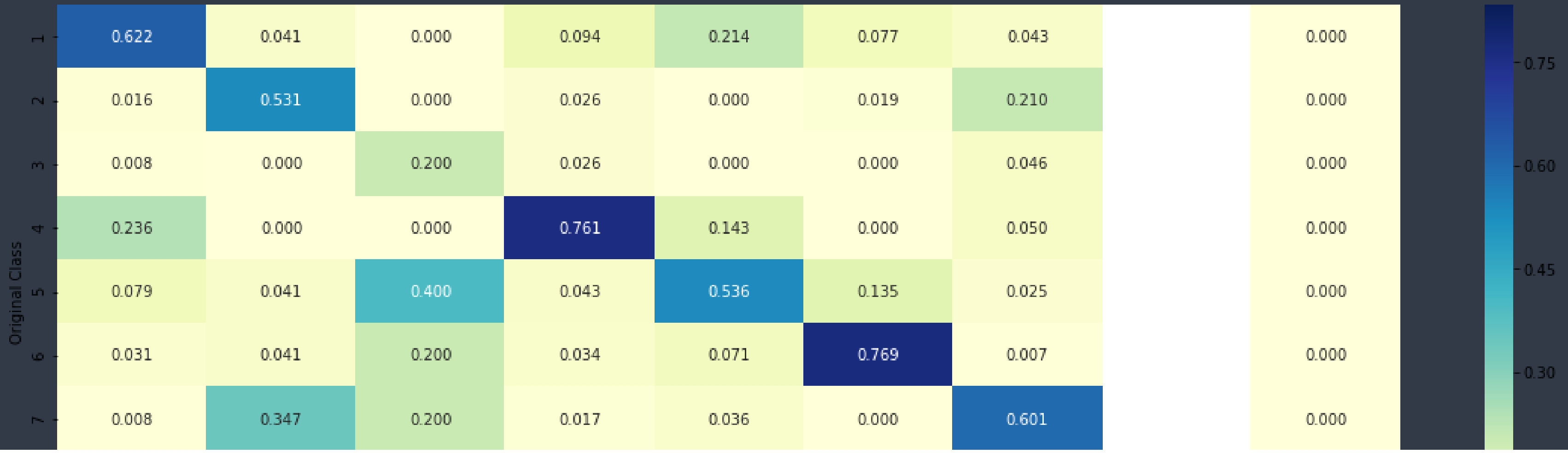
4.7.3 Maximum Voting classifier

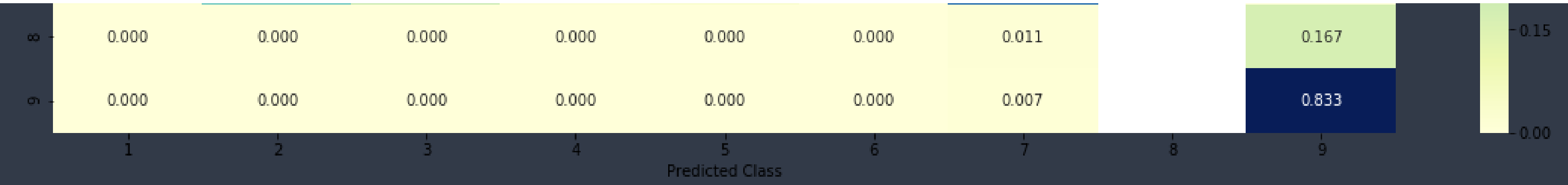
```
In [90]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifie
r.html

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(t
est_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCo
ding)- 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 : 0.9154669881995511
Log loss (CV) on the VotingClassifier : 1.216326492097411
Log loss (test) on the VotingClassifier : 1.1894053126330026
Number of missclassified point : 0.362406015037594
-------------------- Confusion matrix --------------------
```

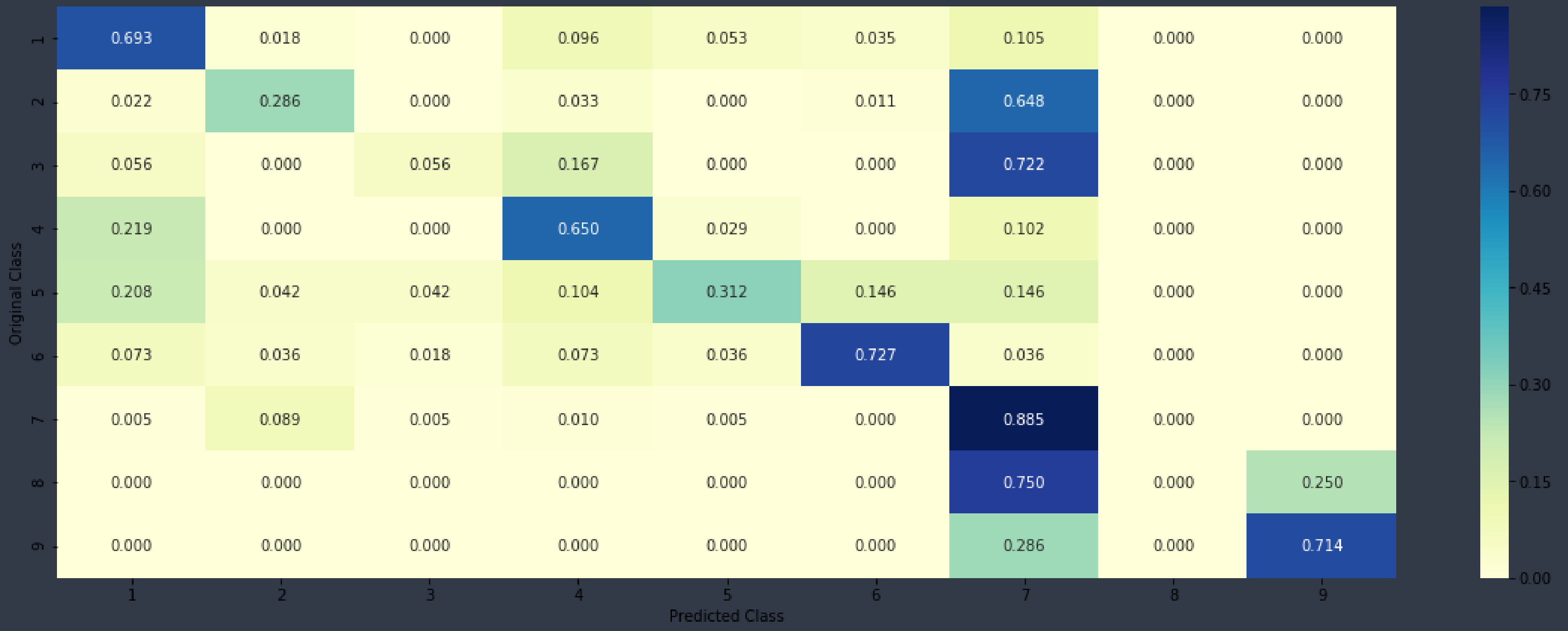


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the

same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

1. Using TF-IDF features

In [92]:
```
# building a tfidf-CountVectorizer with all the words that occured minimum 3 times in tr
ain data
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = tfidf_text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= tfidf_text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numbe
r 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 o
ccured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

```
In [93]: # 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 = tfidf_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 = tfidf_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 [94]: train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_fe
         ature_onehotCoding))
         test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
         re_onehotCoding))
         cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one
         hotCoding))

         train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCod
         ing)).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)).toc
```

```
sr()
cv_y = np.array(list(cv_df['Class']))
```

In [95]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3180)
(number of data points * number of features) in test data =  (665, 3180)
(number of data points * number of features) in cross validation data = (532, 3180)
```

Base line models

In [120]:
```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

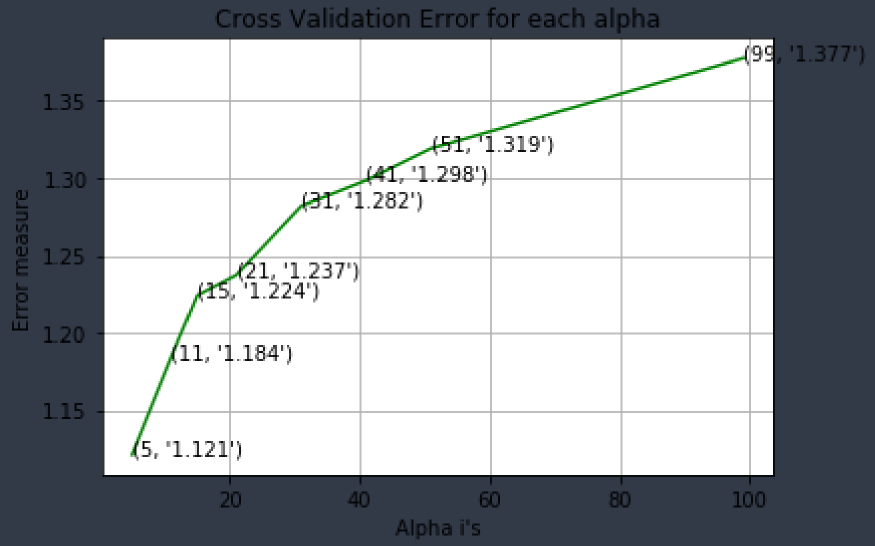
KNN

In [122]:
```
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_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
    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 esti
mates
    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_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.121096758757845
for alpha = 11
Log Loss : 1.1841272563519538
for alpha = 15
Log Loss : 1.2241019311062764
for alpha = 21
Log Loss : 1.237253833990515
for alpha = 31
Log Loss : 1.2818356626838434
for alpha = 41
Log Loss : 1.29846607972916
for alpha = 51
Log Loss : 1.3190793528025297
for alpha = 99
Log Loss : 1.3774120339020592
```

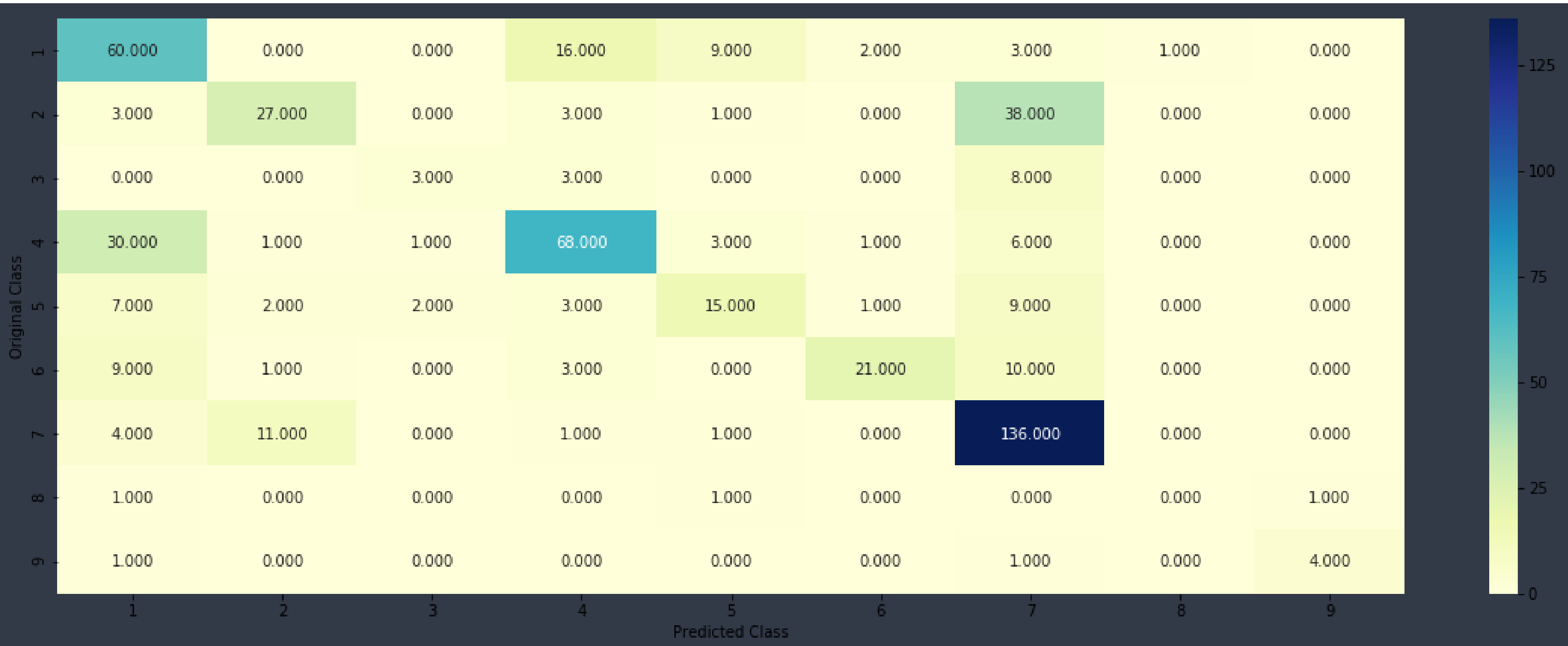


```
For values of best alpha =  5 The train log loss is: 0.8837266026345719
For values of best alpha =  5 The cross validation log loss is: 1.121096758757845
For values of best alpha =  5 The test log loss is: 1.0846021648704152
```

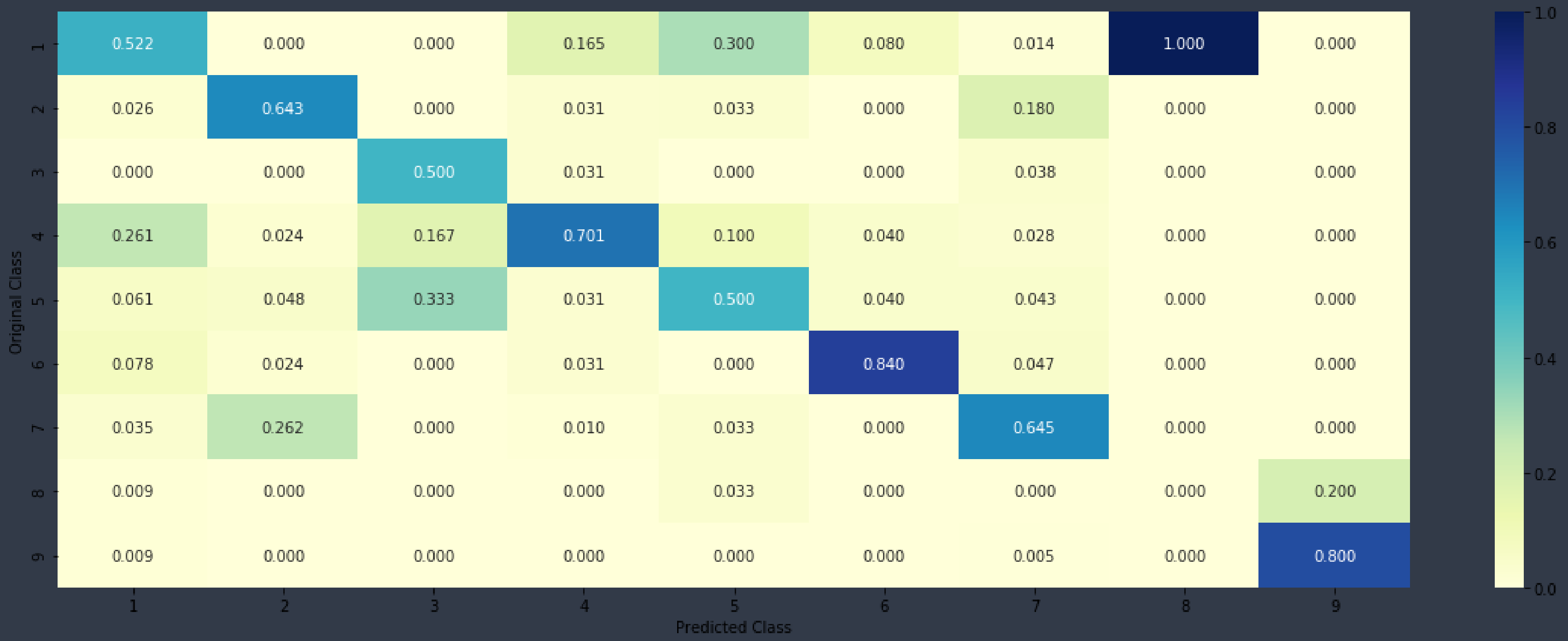
Testing on te best Hyperparameter

In [124]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y
, clf)
```

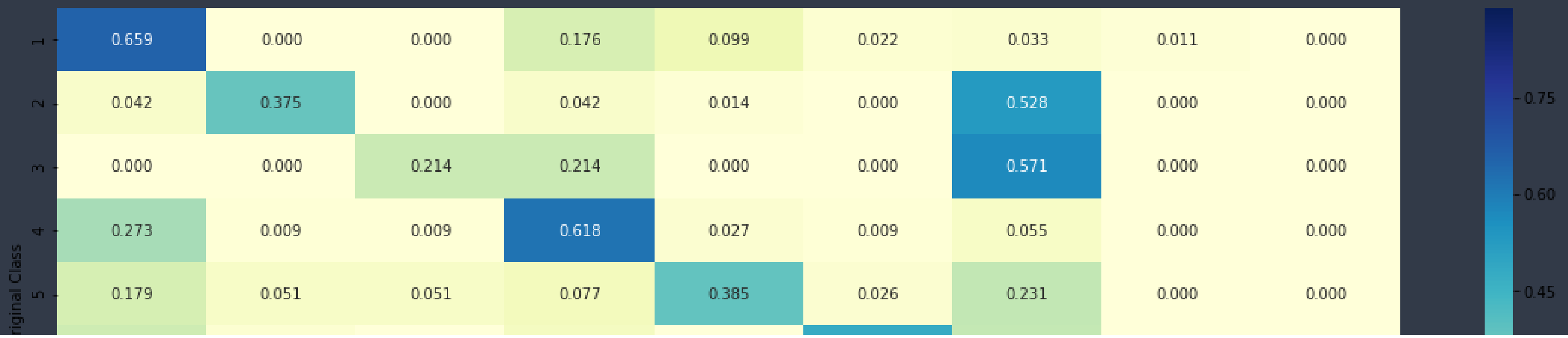
```
Log loss : 1.121096758757845
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

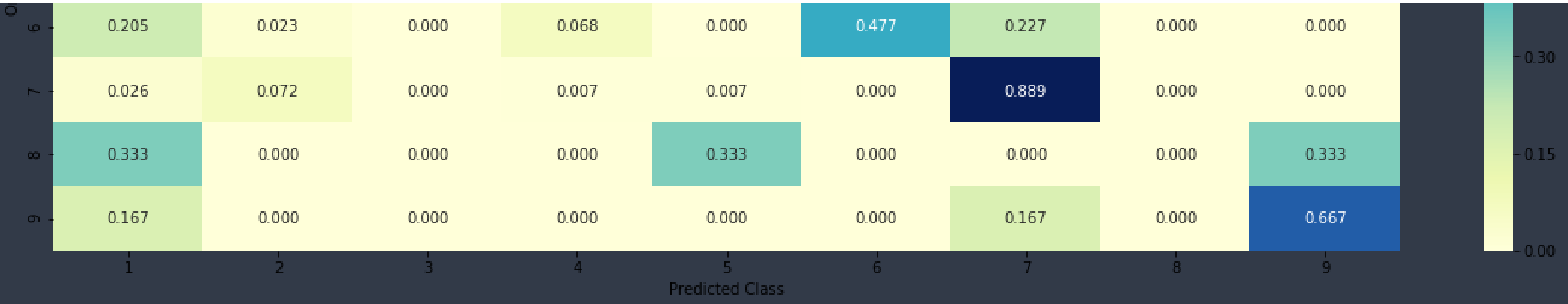


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





Sample query point : 1

In [125]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv = None)
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 12

predicted_cls = sig_clf.predict(test_x_onehotCoding[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_onehotCoding[test_point_index].reshape(1, -1), alpha[b
est_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test
 points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

Naive Bayes

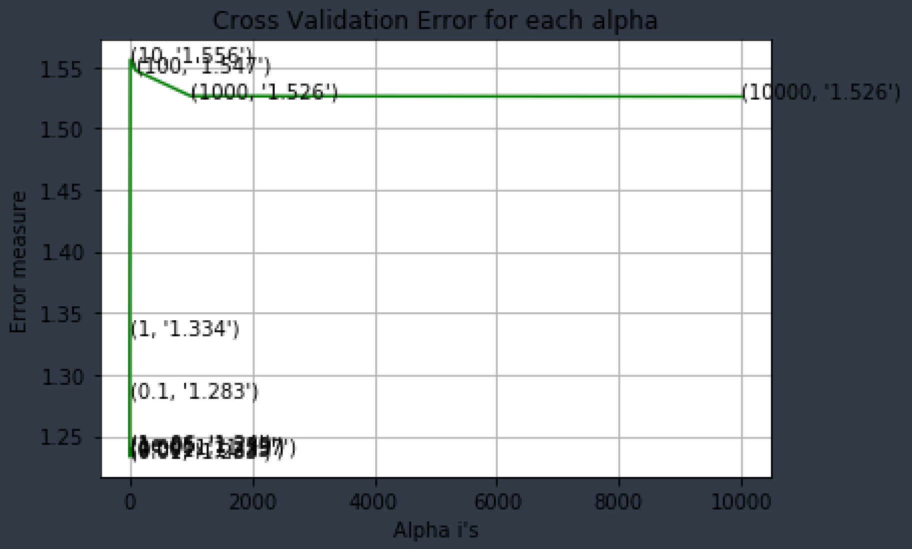
In [127]:

```
alpha = [10**i for i in range(-6,5) ]
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",cv=5)
    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 esti
mates
    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 = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.2395266864541712
for alpha = 1e-05
Log Loss : 1.2391177285616148
for alpha = 0.0001
Log Loss : 1.23705695574075
for alpha = 0.001
Log Loss : 1.2351322057219916
for alpha = 0.01
Log Loss : 1.2331628169706605
for alpha = 0.1
Log Loss : 1.2826340573844566
for alpha = 1
Log Loss : 1.33388184634762
for alpha = 10
Log Loss : 1.5555782261278808
for alpha = 100
Log Loss : 1.5470773319894693
for alpha = 1000

Log Loss : 1.526283548098692
```

```
for alpha = 10000
Log Loss : 1.5258339075182996
```



```
For values of best alpha =  0.01 The train log loss is: 0.46355254883333985
For values of best alpha =  0.01 The cross validation log loss is: 1.2331628169706605
For values of best alpha =  0.01 The test log loss is: 1.183924618014031
```

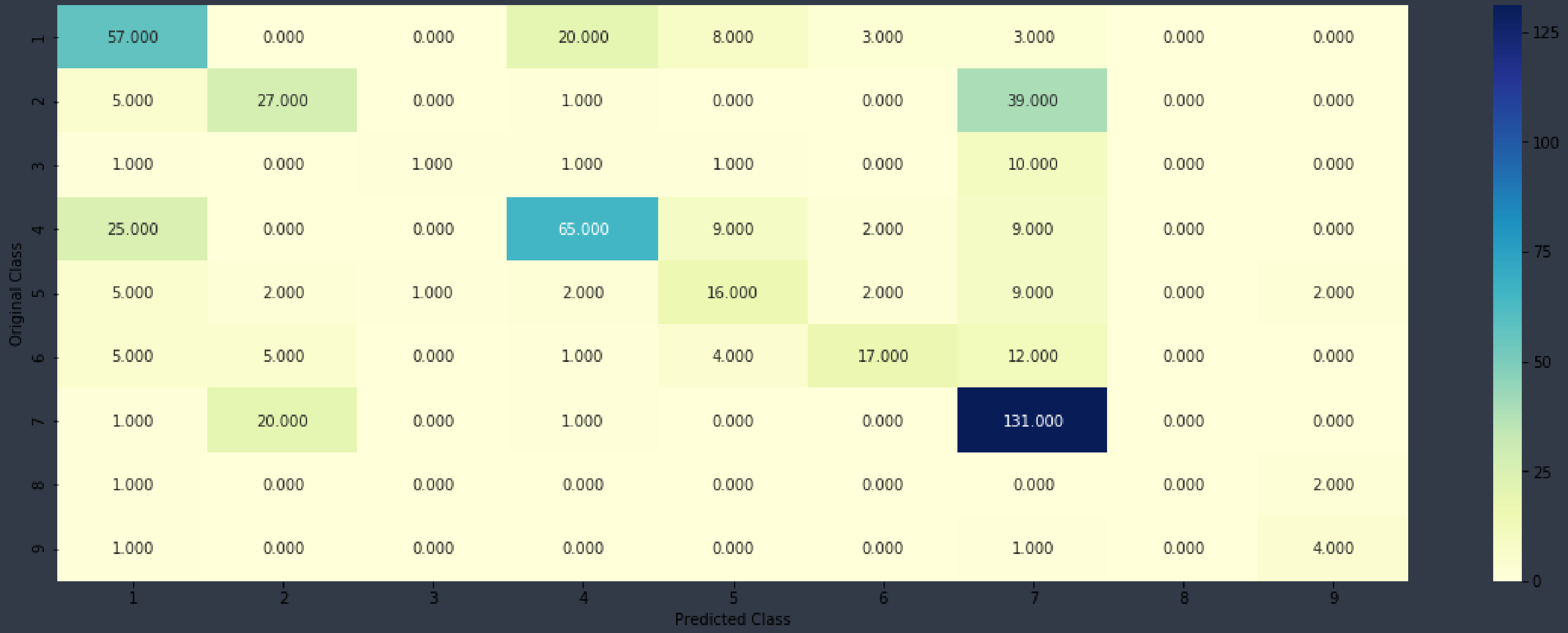
Testing on the best Hyperparameter

In [128]:

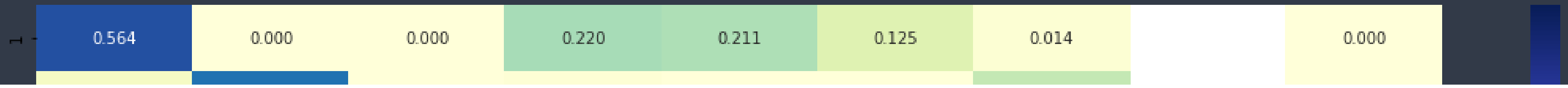
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
```

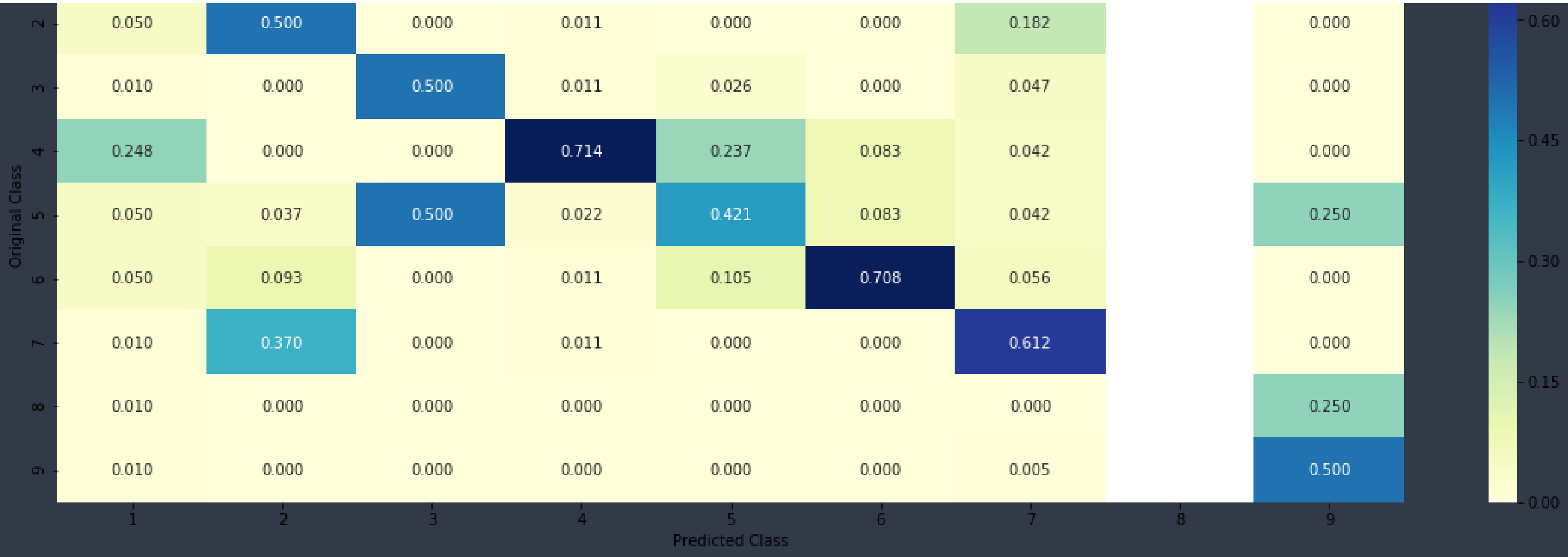
```
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.2383525941580682
Number of missclassified point : 0.40225563909774437
-------------------- Confusion matrix --------------------
```

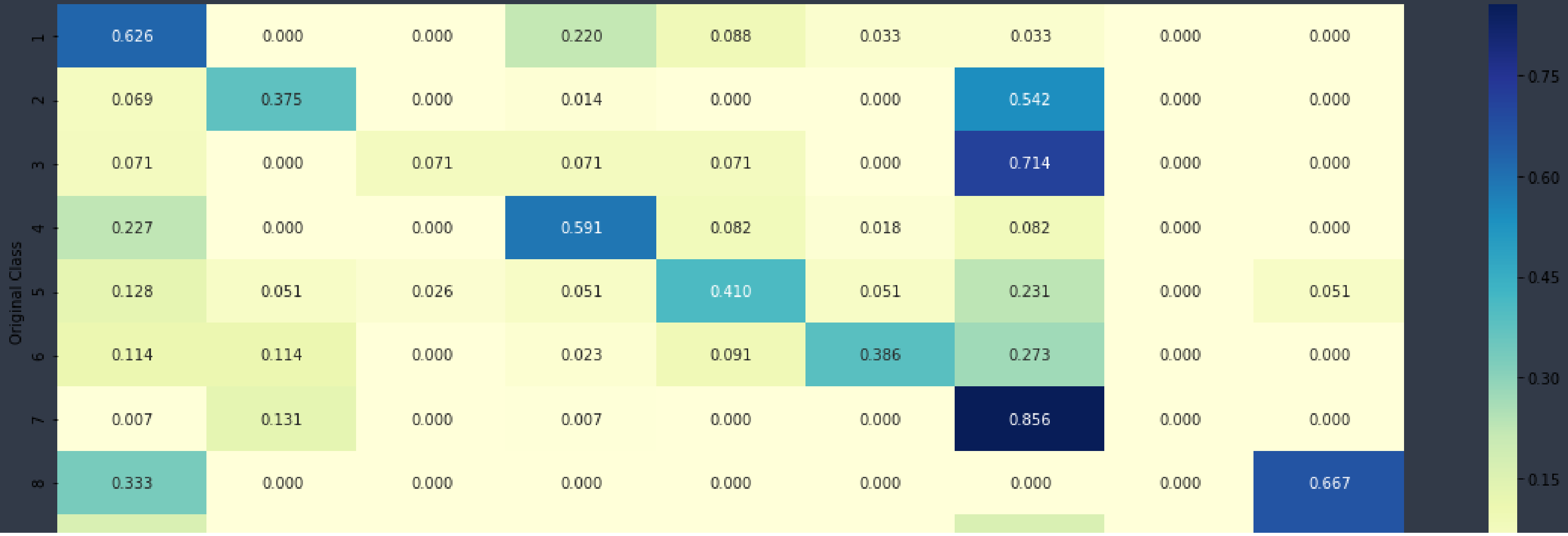


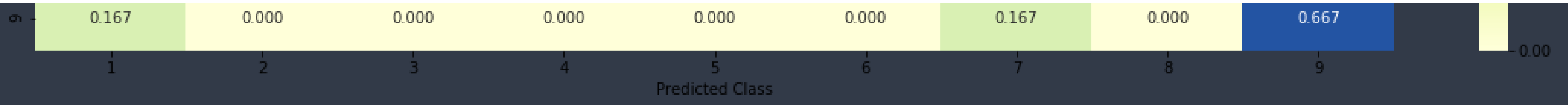
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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





Important features of predicted points

In [129]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5939 0.0429 0.0144 0.1861 0.036  0.0344 0.0852 0.0045 0.0026]]
Actual Class : 1
--------------------------------------------------
7 Text feature [one] present in test data point [True]
10 Text feature [results] present in test data point [True]
11 Text feature [function] present in test data point [True]
12 Text feature [loss] present in test data point [True]
13 Text feature [protein] present in test data point [True]
14 Text feature [two] present in test data point [True]
15 Text feature [therefore] present in test data point [True]
16 Text feature [role] present in test data point [True]
17 Text feature [table] present in test data point [True]
18 Text feature [type] present in test data point [True]
19 Text feature [region] present in test data point [True]
20 Text feature [large] present in test data point [True]
```

```
21 Text feature [also] present in test data point [True]
22 Text feature [however] present in test data point [True]
23 Text feature [functions] present in test data point [True]
25 Text feature [using] present in test data point [True]
26 Text feature [possible] present in test data point [True]
27 Text feature [used] present in test data point [True]
28 Text feature [affect] present in test data point [True]
29 Text feature [gene] present in test data point [True]
30 Text feature [either] present in test data point [True]
31 Text feature [determined] present in test data point [True]
32 Text feature [discussion] present in test data point [True]
33 Text feature [25] present in test data point [True]
35 Text feature [specific] present in test data point [True]
37 Text feature [well] present in test data point [True]
38 Text feature [wild] present in test data point [True]
39 Text feature [binding] present in test data point [True]
40 Text feature [control] present in test data point [True]
41 Text feature [may] present in test data point [True]
44 Text feature [following] present in test data point [True]
45 Text feature [three] present in test data point [True]
46 Text feature [whether] present in test data point [True]
47 Text feature [dna] present in test data point [True]
48 Text feature [containing] present in test data point [True]
49 Text feature [four] present in test data point [True]
50 Text feature [data] present in test data point [True]
52 Text feature [human] present in test data point [True]
54 Text feature [present] present in test data point [True]
55 Text feature [deletion] present in test data point [True]
56 Text feature [expression] present in test data point [True]
57 Text feature [defined] present in test data point [True]
59 Text feature [performed] present in test data point [True]
60 Text feature [important] present in test data point [True]
61 Text feature [shown] present in test data point [True]
62 Text feature [result] present in test data point [True]
63 Text feature [addition] present in test data point [True]
64 Text feature [effect] present in test data point [True]
65 Text feature [although] present in test data point [True]
67 Text feature [several] present in test data point [True]
68 Text feature [10] present in test data point [True]
```

```
69 Text feature [similar] present in test data point [True]
70 Text feature [suggest] present in test data point [True]
71 Text feature [30] present in test data point [True]
73 Text feature [identify] present in test data point [True]
75 Text feature [within] present in test data point [True]
76 Text feature [observed] present in test data point [True]
77 Text feature [fig] present in test data point [True]
78 Text feature [mutations] present in test data point [True]
79 Text feature [sufficient] present in test data point [True]
80 Text feature [studies] present in test data point [True]
81 Text feature [indicate] present in test data point [True]
82 Text feature [example] present in test data point [True]
83 Text feature [corresponding] present in test data point [True]
84 Text feature [mutation] present in test data point [True]
85 Text feature [together] present in test data point [True]
86 Text feature [37] present in test data point [True]
87 Text feature [first] present in test data point [True]
88 Text feature [15] present in test data point [True]
92 Text feature [essential] present in test data point [True]
93 Text feature [whereas] present in test data point [True]
95 Text feature [single] present in test data point [True]
98 Text feature [involved] present in test data point [True]
99 Text feature [identified] present in test data point [True]
Out of the top  100  features  74 are present in query point
```

Logistic Regression with Class Balancing

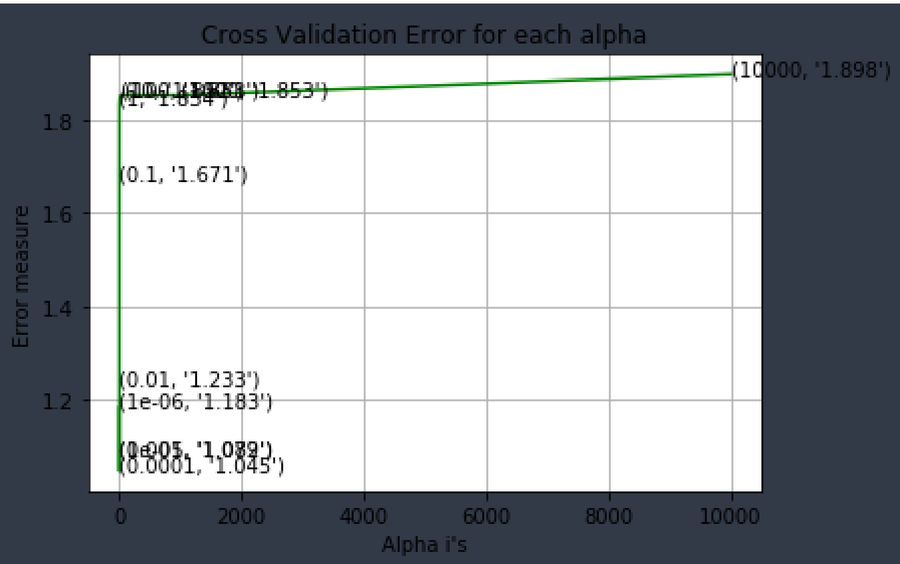
Hyperparameter Tuning

In [132]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=42)
```

```
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    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 esti
mates
    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",cv=5)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1833372846150654
for alpha = 1e-05
Log Loss : 1.081871226987801
for alpha = 0.0001
Log Loss : 1.0454118043451743
for alpha = 0.001
Log Loss : 1.0792975068099502
for alpha = 0.01
Log Loss : 1.2325664282406343
for alpha = 0.1
Log Loss : 1.6713610997408757
for alpha = 1
Log Loss : 1.8338267436569837
for alpha = 10
Log Loss : 1.851506095921084
for alpha = 100
Log Loss : 1.8534581553710916
for alpha = 1000
Log Loss : 1.8531024977108297
for alpha = 10000
Log Loss : 1.8975771018736085
```

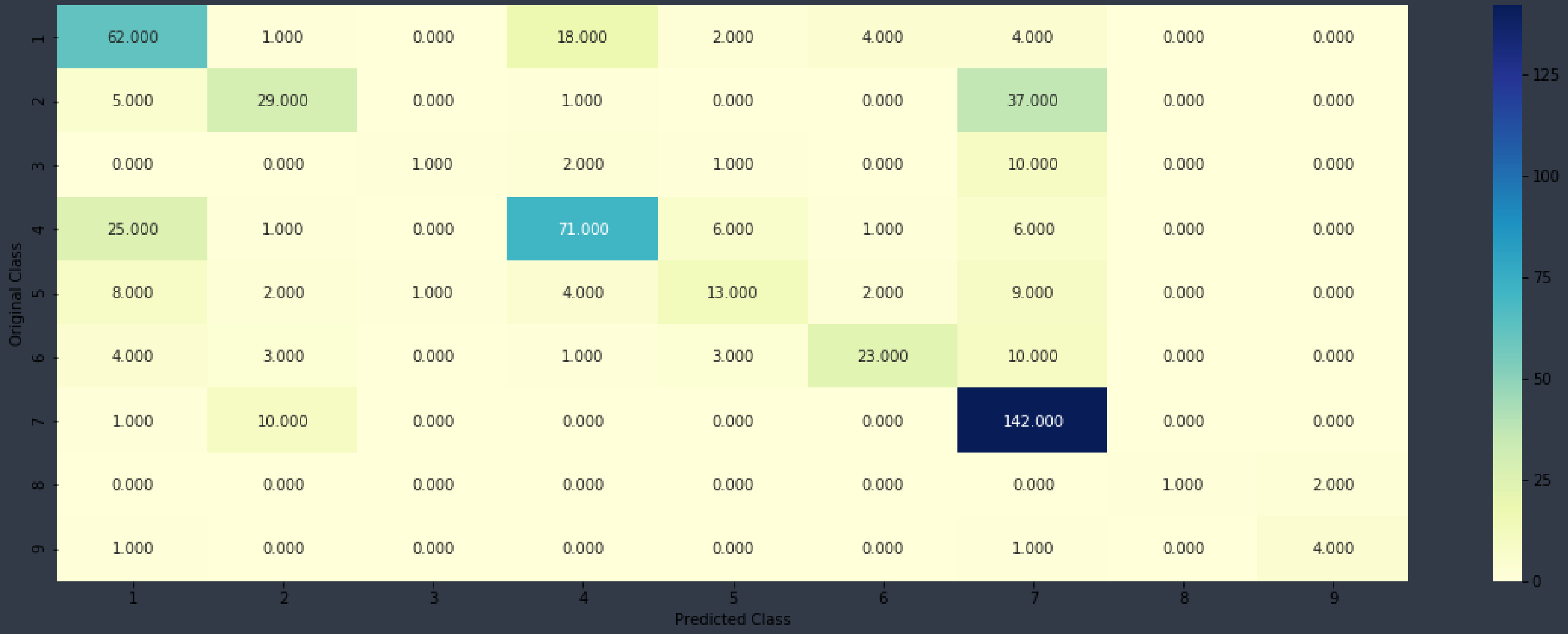


```
For values of best alpha =  0.0001 The train log loss is: 0.40241869814200093
For values of best alpha =  0.0001 The cross validation log loss is: 1.0454118043451743
For values of best alpha =  0.0001 The test log loss is: 0.96519172962454
```

Testing on best Hyperparameter

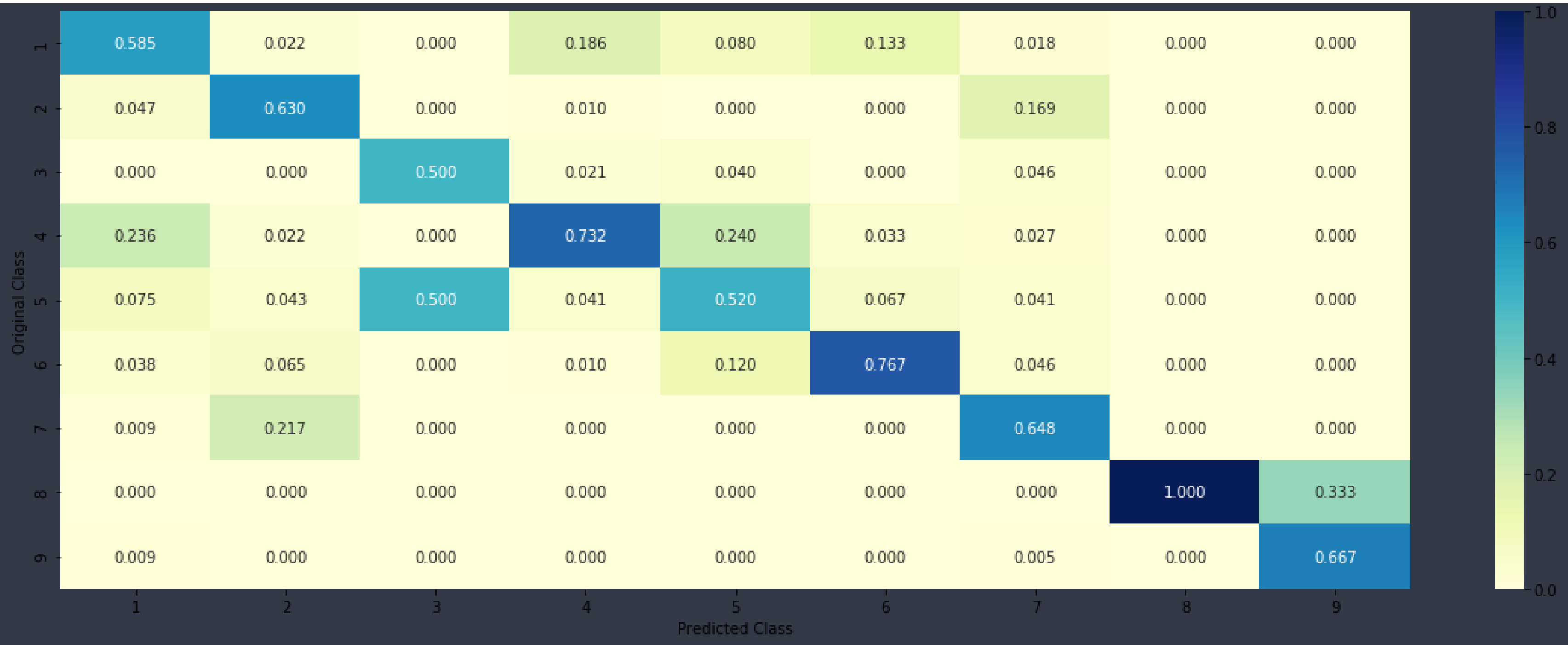
In [138]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.041968422634854
Number of missclassified point : 0.34962406015037595
-------------------- Confusion matrix --------------------
```

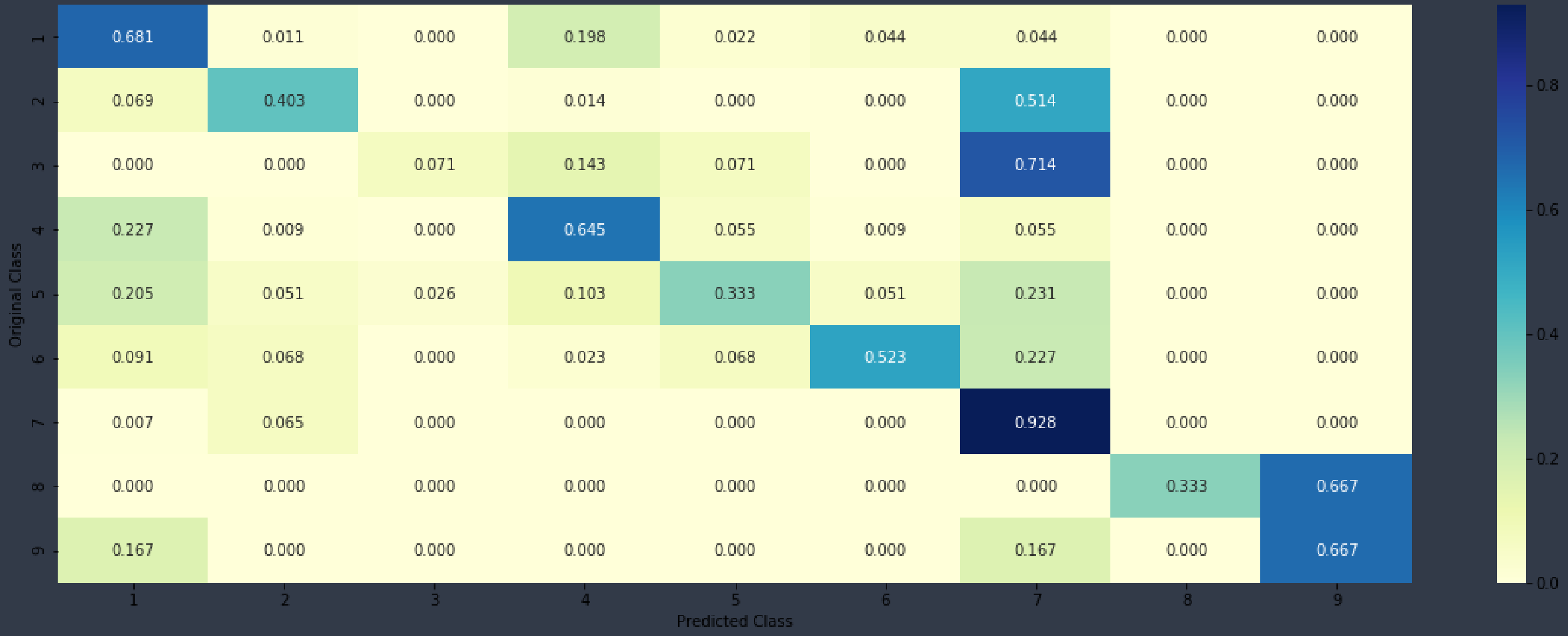


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	62.000	1.000	0.000	18.000	2.000	4.000	4.000	0.000	0.000
2	5.000	29.000	0.000	1.000	0.000	0.000	37.000	0.000	0.000
3	0.000	0.000	1.000	2.000	1.000	0.000	10.000	0.000	0.000
4	25.000	1.000	0.000	71.000	6.000	1.000	6.000	0.000	0.000
5	8.000	2.000	1.000	4.000	13.000	2.000	9.000	0.000	0.000
6	4.000	3.000	0.000	1.000	3.000	23.000	10.000	0.000	0.000
7	1.000	10.000	0.000	0.000	0.000	0.000	142.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	2.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	4.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.681	0.011	0.000	0.198	0.022	0.044	0.044	0.000	0.000
2	0.069	0.403	0.000	0.014	0.000	0.000	0.514	0.000	0.000
3	0.000	0.000	0.071	0.143	0.071	0.000	0.714	0.000	0.000
4	0.227	0.009	0.000	0.645	0.055	0.009	0.055	0.000	0.000
5	0.205	0.051	0.026	0.103	0.333	0.051	0.231	0.000	0.000
6	0.091	0.068	0.000	0.023	0.068	0.523	0.227	0.000	0.000
7	0.007	0.065	0.000	0.000	0.000	0.000	0.928	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.667
9	0.167	0.000	0.000	0.000	0.000	0.000	0.167	0.000	0.667

Important Features of Predicted point

In [146]:
```
test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

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

```
Predicted Class : 1
Predicted Class Probabilities: [[8.57e-01 8.65e-02 2.00e-04 2.05e-02 8.70e-03 1.72e-02 8.80e-03 8.00e-04
  3.00e-04]]
Actual Class : 1
--------------------------------------------------
82 Text feature [hydrophobic] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

Logistic regression without class balancing

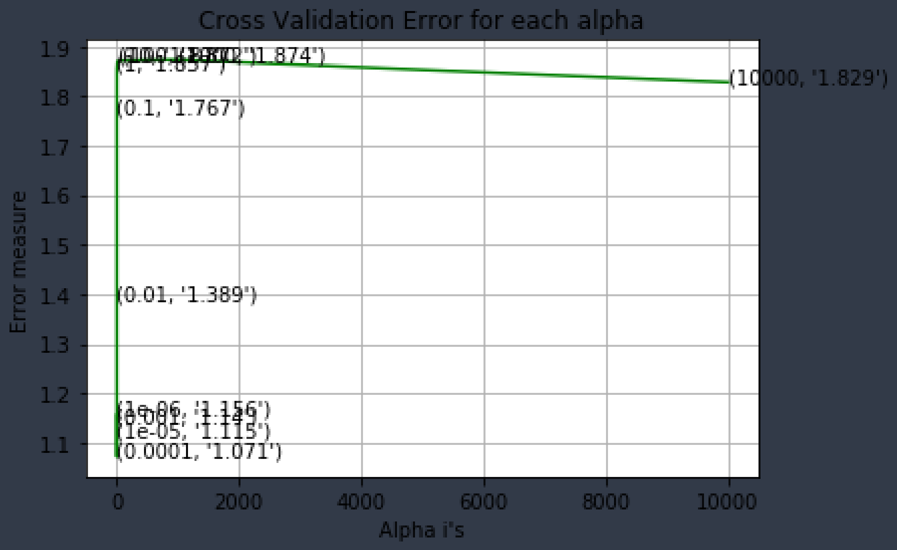
Hyperparameter Tuning

In [148]:
```
alpha = [10**i for i in range(-6,5) ]
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",cv=5)
    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 esti
mates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
```

```python
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",cv=5)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1559549264282034
for alpha = 1e-05
Log Loss : 1.1149171483211784
for alpha = 0.0001
Log Loss : 1.0712633464177883
```

```
for alpha = 0.001
Log Loss : 1.1403290818739777
for alpha = 0.01
Log Loss : 1.3889956724146117
for alpha = 0.1
Log Loss : 1.7672218790771852
for alpha = 1
Log Loss : 1.8565490594889116
for alpha = 10
Log Loss : 1.870332384337331
for alpha = 100
Log Loss : 1.871927143298842
for alpha = 1000
Log Loss : 1.8736052213365837
for alpha = 10000
Log Loss : 1.8286287881278562
```

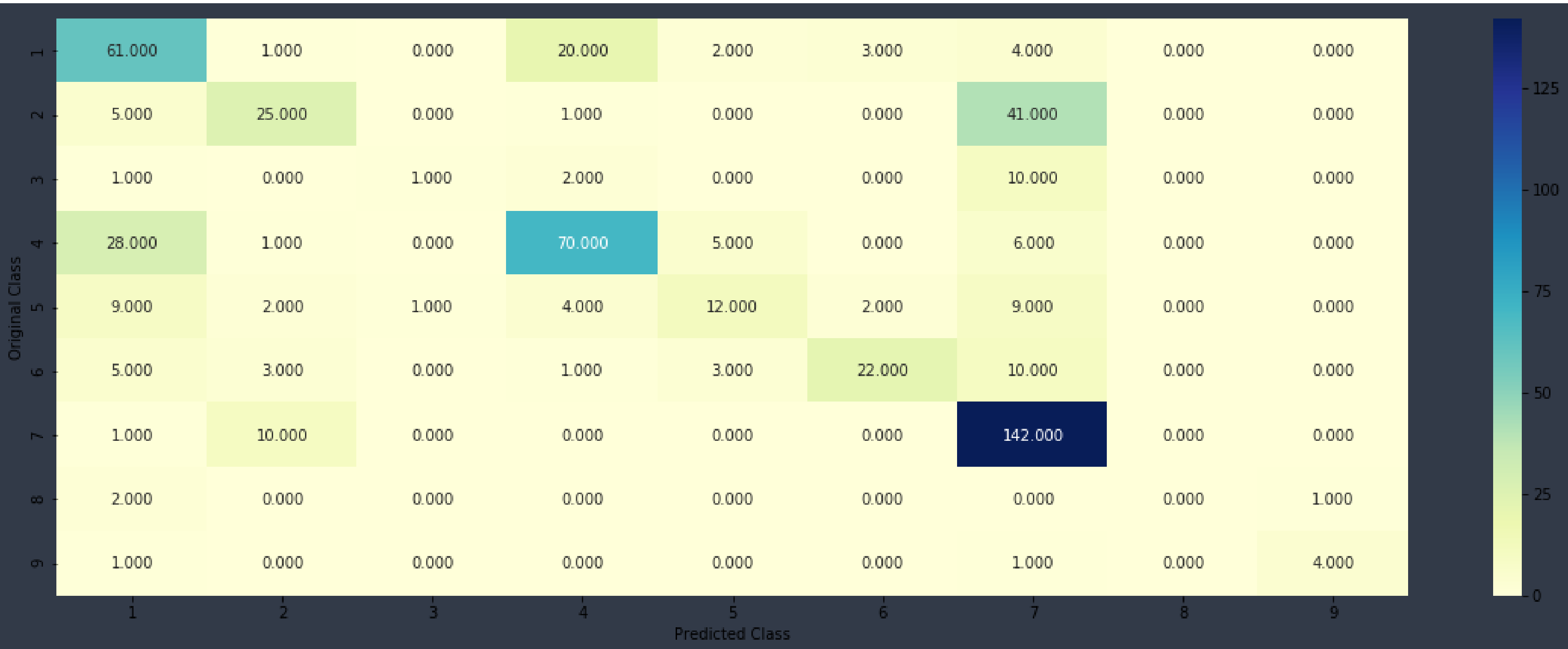


```
For values of best alpha =  0.0001 The train log loss is: 0.3920800535702973
For values of best alpha =  0.0001 The cross validation log loss is: 1.0712633464177883

For values of best alpha =  0.0001 The test log loss is: 0.9783379895169934
```

Testing on Best Hyperparameter

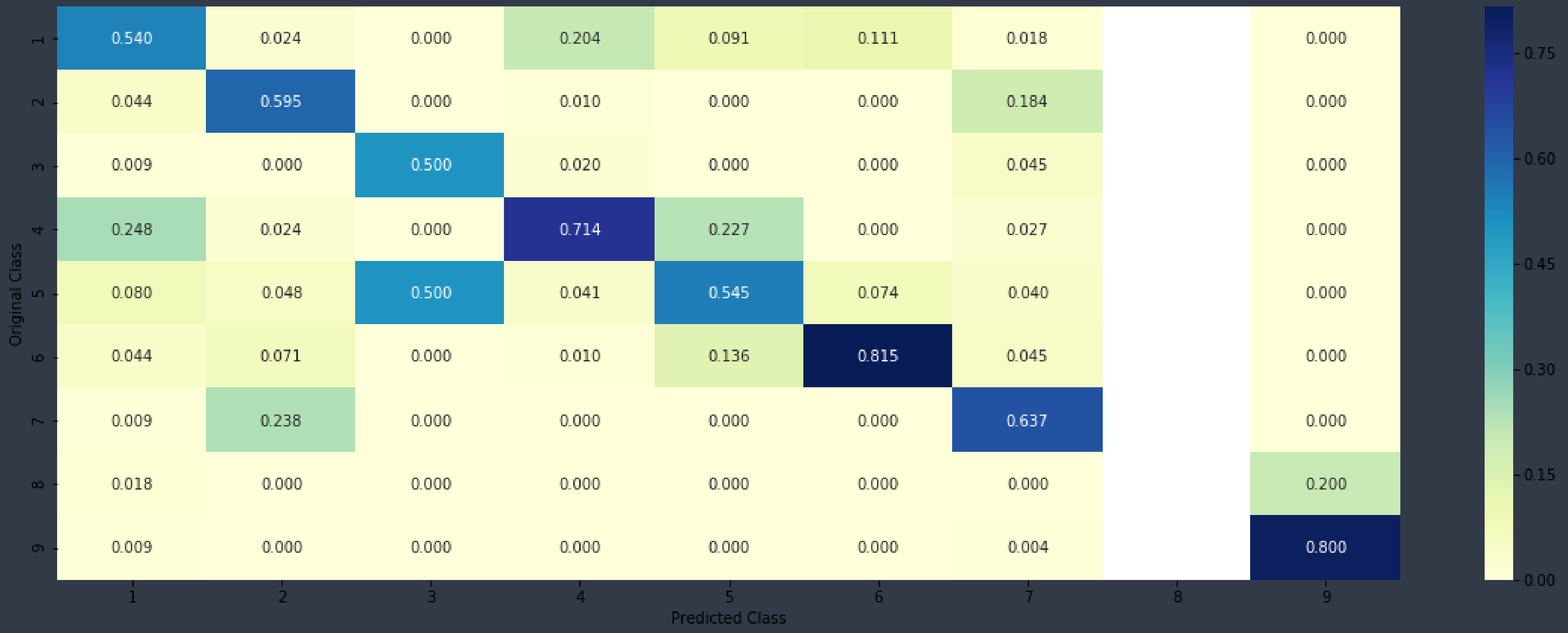
In [149]:
```
clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.0599596854692228
Number of missclassified point : 0.36654135338345867
-------------------- Confusion matrix --------------------
```



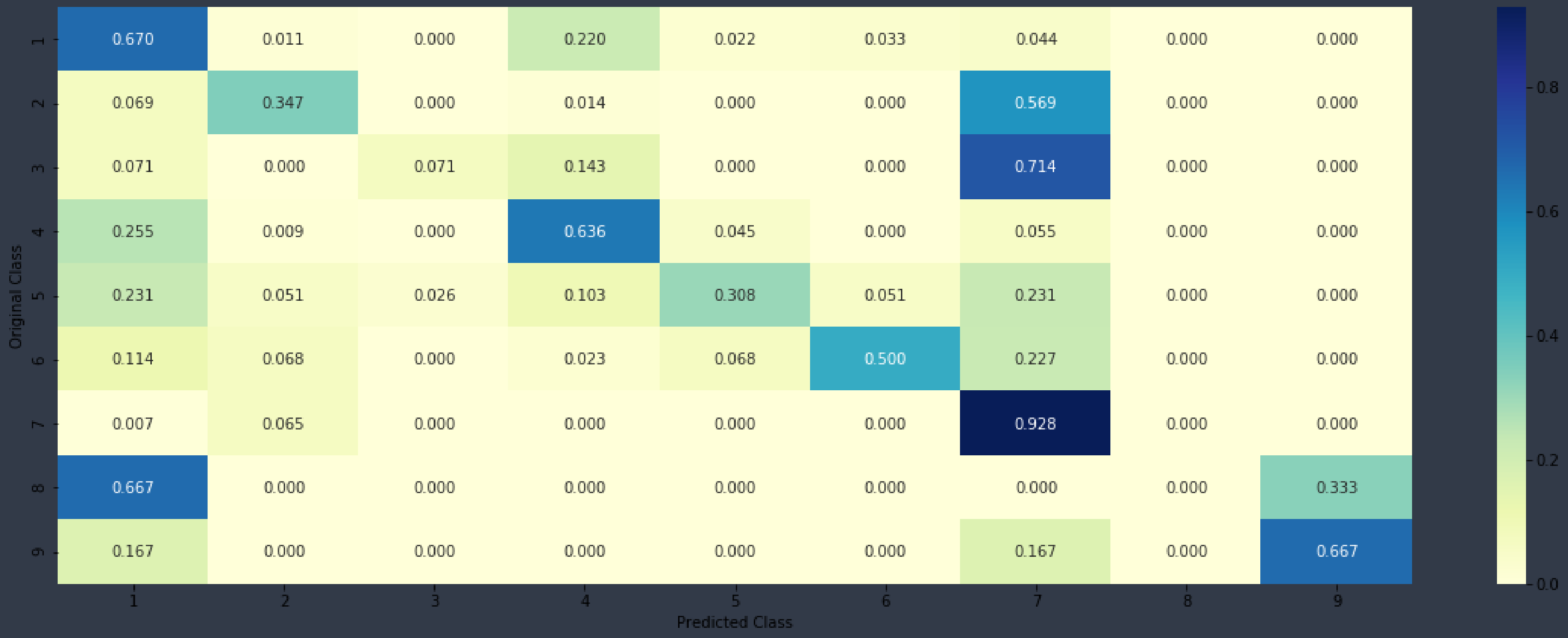
	1	2	3	4	5	6	7	8	9
1	61.000	1.000	0.000	20.000	2.000	3.000	4.000	0.000	0.000
2	5.000	25.000	0.000	1.000	0.000	0.000	41.000	0.000	0.000
3	1.000	0.000	1.000	2.000	0.000	0.000	10.000	0.000	0.000
4	28.000	1.000	0.000	70.000	5.000	0.000	6.000	0.000	0.000
5	9.000	2.000	1.000	4.000	12.000	2.000	9.000	0.000	0.000
6	5.000	3.000	0.000	1.000	3.000	22.000	10.000	0.000	0.000
7	1.000	10.000	0.000	0.000	0.000	0.000	142.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.540	0.024	0.000	0.204	0.091	0.111	0.018		0.000
2	0.044	0.595	0.000	0.010	0.000	0.000	0.184		0.000
3	0.009	0.000	0.500	0.020	0.000	0.000	0.045		0.000
4	0.248	0.024	0.000	0.714	0.227	0.000	0.027		0.000
5	0.080	0.048	0.500	0.041	0.545	0.074	0.040		0.000
6	0.044	0.071	0.000	0.010	0.136	0.815	0.045		0.000
7	0.009	0.238	0.000	0.000	0.000	0.000	0.637		0.000
8	0.018	0.000	0.000	0.000	0.000	0.000	0.000		0.200
9	0.009	0.000	0.000	0.000	0.000	0.000	0.004		0.800

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



In [151]:
```
test_point_index = 200
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.040e-02 6.000e-03 7.300e-03 5.000e-03 2.810e-02 3.200e-03 9.379e-01
  2.000e-03 1.000e-04]]
Actual Class : 7
--------------------------------------------------
15 Text feature [downstream] present in test data point [True]
27 Text feature [activate] present in test data point [True]
35 Text feature [activation] present in test data point [True]
38 Text feature [activated] present in test data point [True]
41 Text feature [constitutive] present in test data point [True]
44 Text feature [insertion] present in test data point [True]
71 Text feature [ras] present in test data point [True]
77 Text feature [activating] present in test data point [True]
80 Text feature [versus] present in test data point [True]
96 Text feature [overexpression] present in test data point [True]
Out of the top  100  features  10 are present in query point
```

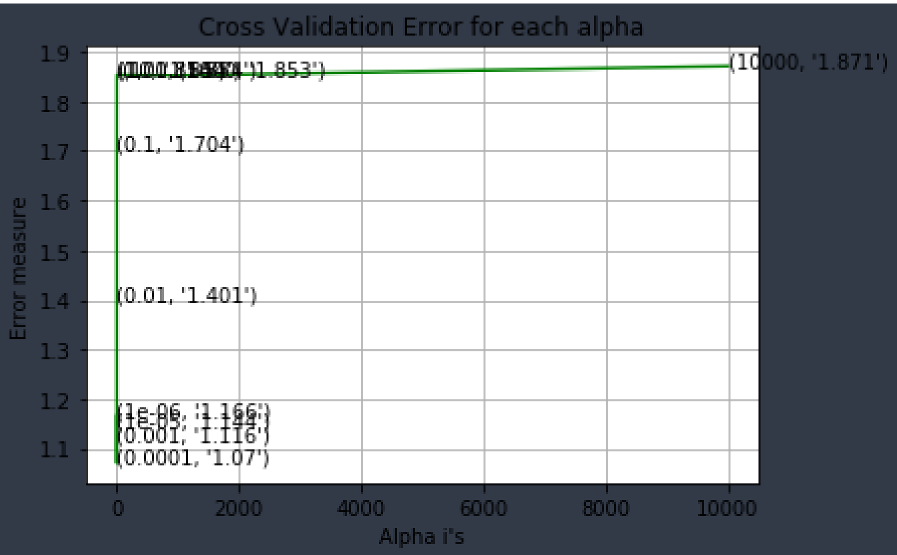
Linear Support Vector Machine

In [153]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hinge', ra
ndom_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    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 esti
mates
    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
='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
```

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

```
for alpha = 1e-06
Log Loss : 1.1659292412707971
for alpha = 1e-05
Log Loss : 1.1442115091152125
for alpha = 0.0001
Log Loss : 1.0703971477460354
for alpha = 0.001
Log Loss : 1.1157960521419852
for alpha = 0.01
Log Loss : 1.400609694723615
for alpha = 0.1
Log Loss : 1.7035406847783277
for alpha = 1
Log Loss : 1.85386080399216
for alpha = 10
Log Loss : 1.8538620745497503
for alpha = 100
Log Loss : 1.8538621739712606
for alpha = 1000
Log Loss : 1.8529450409693369
for alpha = 10000
Log Loss : 1.8710899229375983
```

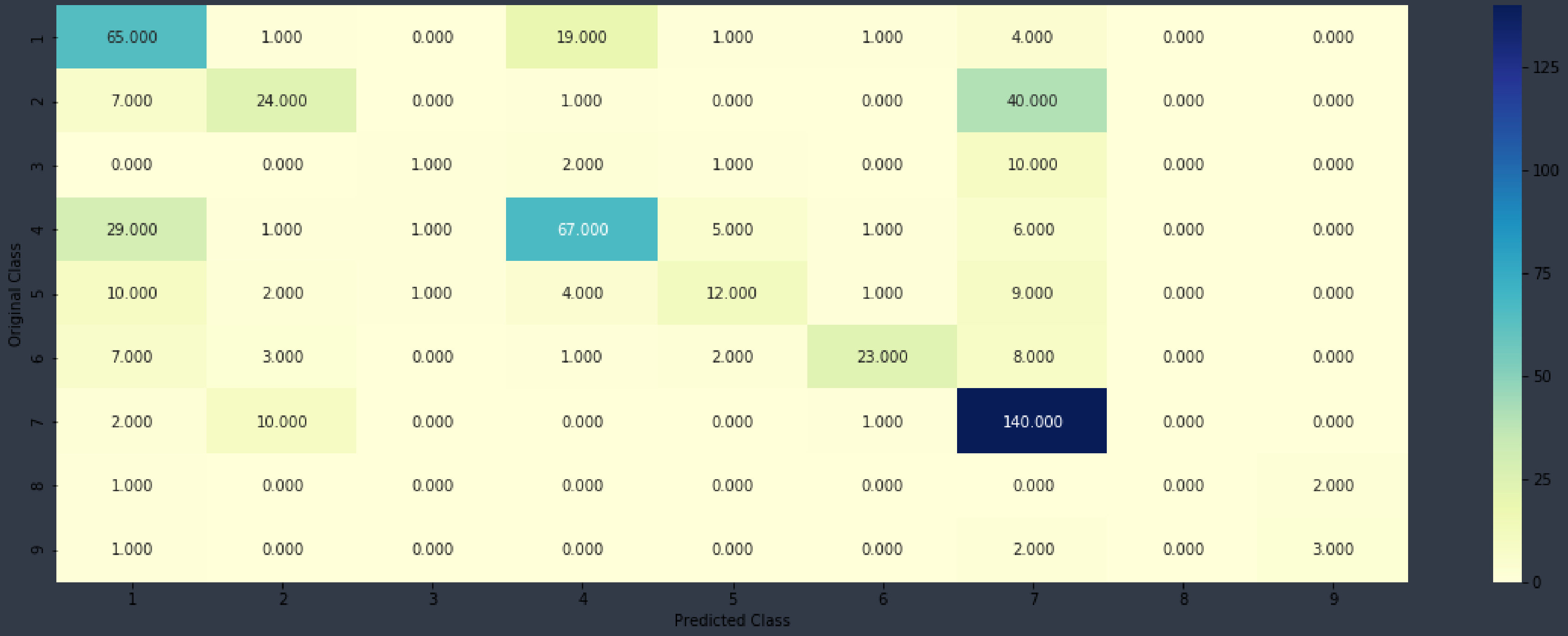


```
For values of best alpha =  0.0001 The train log loss is: 0.32398436587639856
For values of best alpha =  0.0001 The cross validation log loss is: 1.0703971477460354
For values of best alpha =  0.0001 The test log loss is: 0.9990877646882539
```

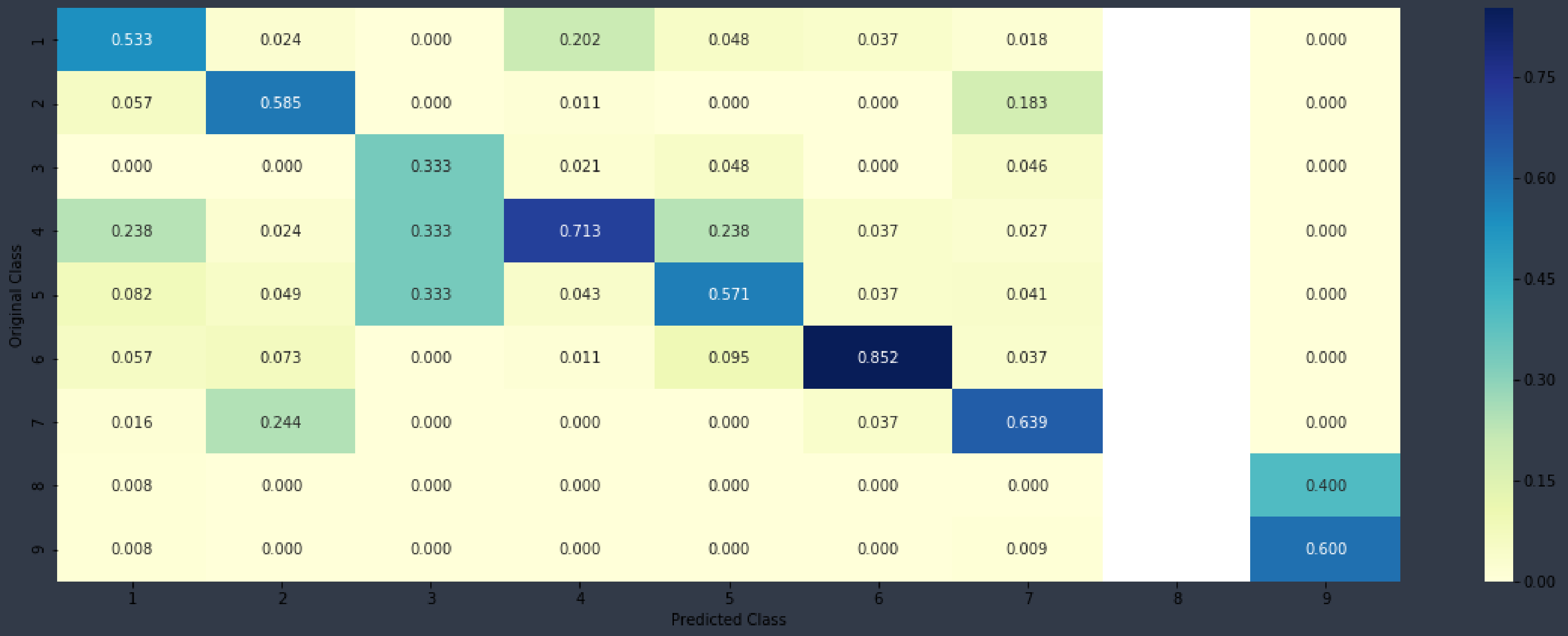
Testing on best Hyperparameter

In [154]:
```
clf = SGDClassifier(class_weight='balanced' ,alpha=alpha[best_alpha], penalty='l2', loss
='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
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 estimate
s
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotC
oding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.0728527928389726
Number of missclassified point : 0.37030075187969924
-------------------- Confusion matrix --------------------
```

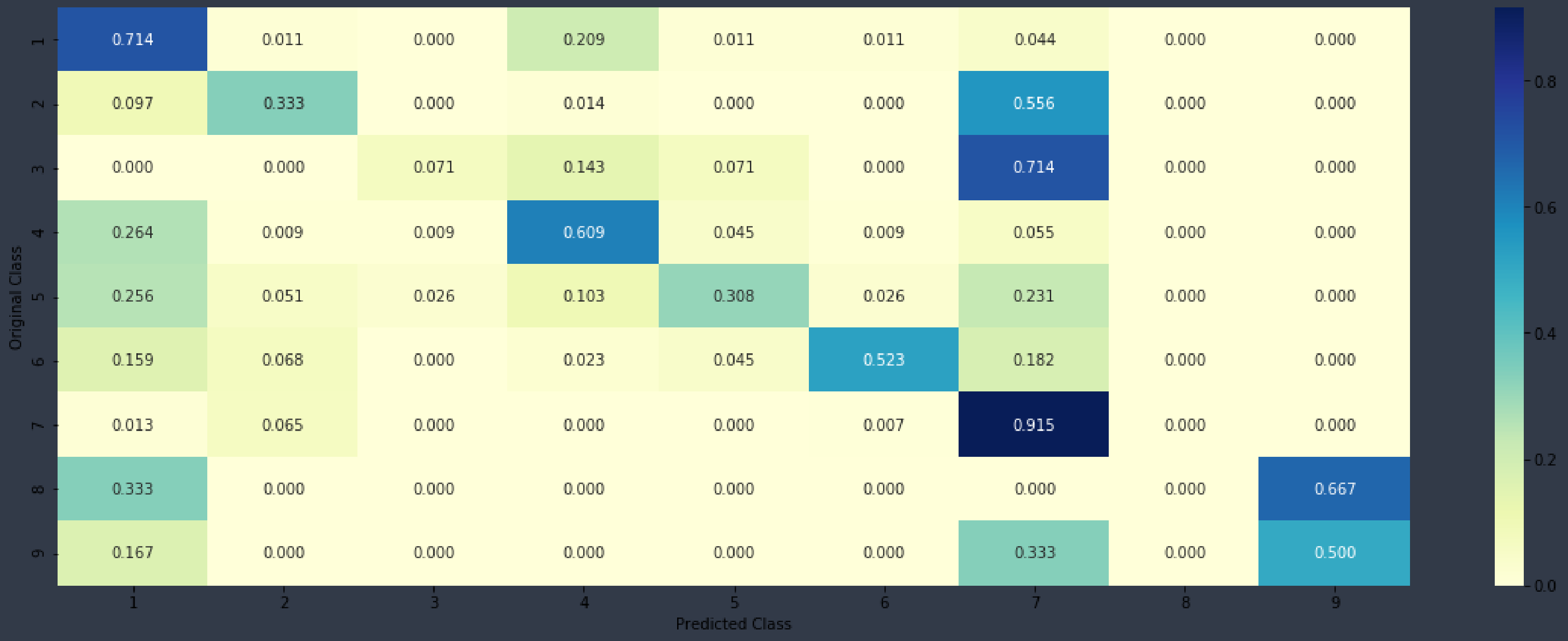


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.533	0.024	0.000	0.202	0.048	0.037	0.018		0.000
2	0.057	0.585	0.000	0.011	0.000	0.000	0.183		0.000
3	0.000	0.000	0.333	0.021	0.048	0.000	0.046		0.000
4	0.238	0.024	0.333	0.713	0.238	0.037	0.027		0.000
5	0.082	0.049	0.333	0.043	0.571	0.037	0.041		0.000
6	0.057	0.073	0.000	0.011	0.095	0.852	0.037		0.000
7	0.016	0.244	0.000	0.000	0.000	0.037	0.639		0.000
8	0.008	0.000	0.000	0.000	0.000	0.000	0.000		0.400
9	0.008	0.000	0.000	0.000	0.000	0.000	0.009		0.600

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



	1	2	3	4	5	6	7	8	9
1	0.714	0.011	0.000	0.209	0.011	0.011	0.044	0.000	0.000
2	0.097	0.333	0.000	0.014	0.000	0.000	0.556	0.000	0.000
3	0.000	0.000	0.071	0.143	0.071	0.000	0.714	0.000	0.000
4	0.264	0.009	0.009	0.609	0.045	0.009	0.055	0.000	0.000
5	0.256	0.051	0.026	0.103	0.308	0.026	0.231	0.000	0.000
6	0.159	0.068	0.000	0.023	0.045	0.523	0.182	0.000	0.000
7	0.013	0.065	0.000	0.000	0.000	0.007	0.915	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.667
9	0.167	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.500

Important features for predicted point

In [157]:

```
test_point_index = 500
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[4.189e-01 1.090e-02 9.000e-04 4.734e-01 3.010e-02 4.800e-03 6.060e-02
  2.000e-04 3.000e-04]]
Actual Class : 4
--------------------------------------------------
92 Text feature [suppressor] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

Random Forest Classifier

In [159]:
```
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, rand
om_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv =None)
        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))
```

```
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_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",cv =None)
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 i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2541660340473217
for n_estimators = 100 and max depth =  10
Log Loss : 1.2596999017093822
for n_estimators = 200 and max depth =  5
Log Loss : 1.2530940065971095
for n_estimators = 200 and max depth =  10
Log Loss : 1.2570530443599808
for n_estimators = 500 and max depth =  5
Log Loss : 1.2447175777923838
for n_estimators = 500 and max depth =  10
Log Loss : 1.2520506545712338
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2356188827440346
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2529955421254342
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2356419291049852
```

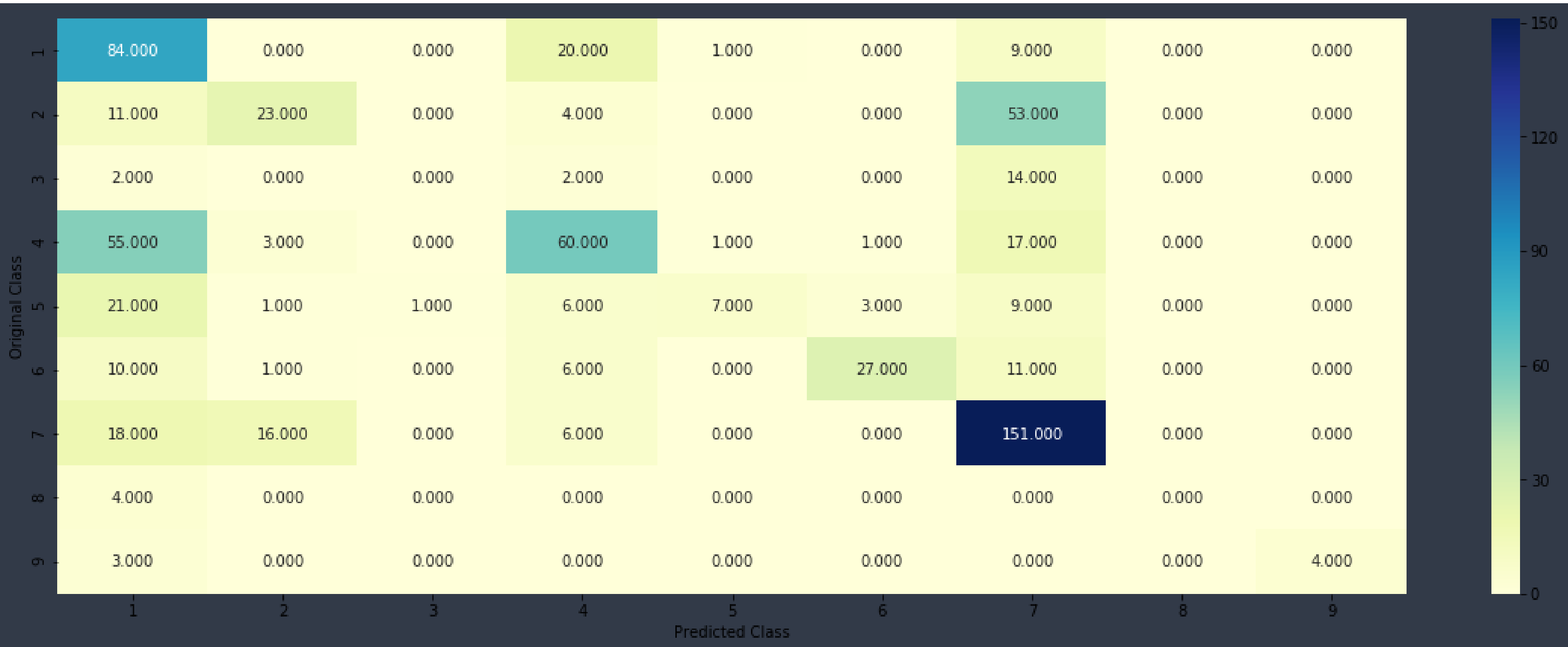
```
for n_estimators = 2000 and max depth =  10
Log Loss : 1.252482517148329
For values of best estimator =  1000 The train log loss is: 0.8568784269766475
For values of best estimator =  1000 The cross validation log loss is: 1.2356188827440346
For values of best estimator =  1000 The test log loss is: 1.2043254095369837
```

Testing on best Hyperparameter

In [164]:

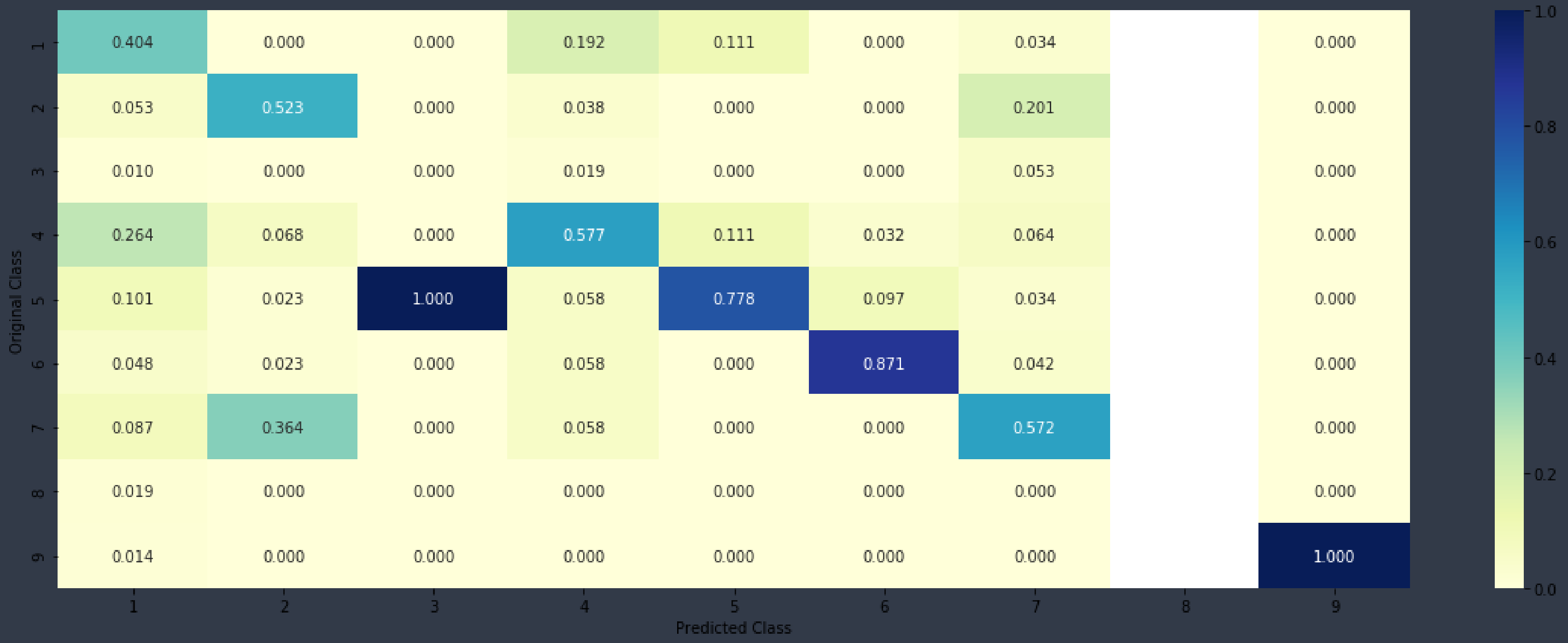
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', ma
x_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

```
Log Loss : 1.2043254095369837
Number of missclassified point : 0.46466165413533833
-------------------- Confusion matrix --------------------
```



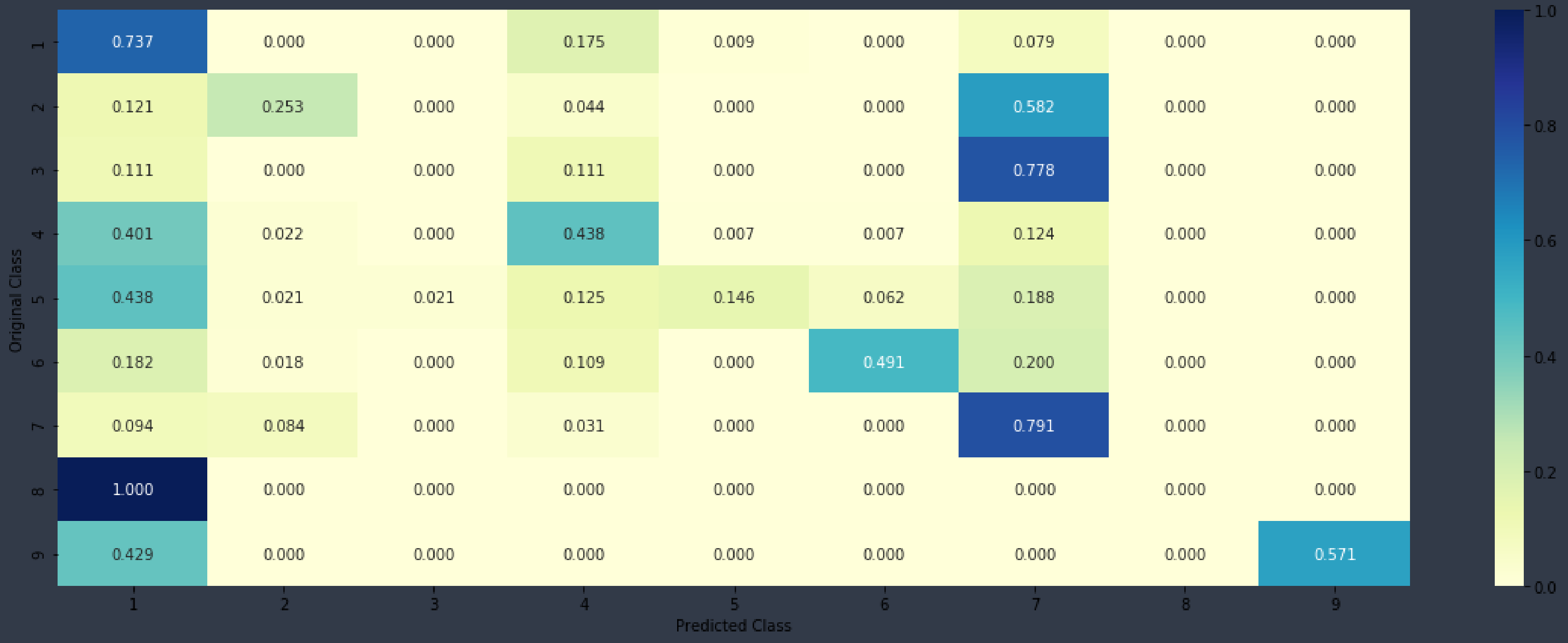
	1	2	3	4	5	6	7	8	9
1	84.000	0.000	0.000	20.000	1.000	0.000	9.000	0.000	0.000
2	11.000	23.000	0.000	4.000	0.000	0.000	53.000	0.000	0.000
3	2.000	0.000	0.000	2.000	0.000	0.000	14.000	0.000	0.000
4	55.000	3.000	0.000	60.000	1.000	1.000	17.000	0.000	0.000
5	21.000	1.000	1.000	6.000	7.000	3.000	9.000	0.000	0.000
6	10.000	1.000	0.000	6.000	0.000	27.000	11.000	0.000	0.000
7	18.000	16.000	0.000	6.000	0.000	0.000	151.000	0.000	0.000
8	4.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	3.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.404	0.000	0.000	0.192	0.111	0.000	0.034		0.000
2	0.053	0.523	0.000	0.038	0.000	0.000	0.201		0.000
3	0.010	0.000	0.000	0.019	0.000	0.000	0.053		0.000
4	0.264	0.068	0.000	0.577	0.111	0.032	0.064		0.000
5	0.101	0.023	1.000	0.058	0.778	0.097	0.034		0.000
6	0.048	0.023	0.000	0.058	0.000	0.871	0.042		0.000
7	0.087	0.364	0.000	0.058	0.000	0.000	0.572		0.000
8	0.019	0.000	0.000	0.000	0.000	0.000	0.000		0.000
9	0.014	0.000	0.000	0.000	0.000	0.000	0.000		1.000

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



Feature importance of predicted point

In [167]:
```
test_point_index = 120
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0226 0.562  0.0099 0.0208 0.0278 0.0253 0.326  0.0042 0.0015]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] 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 [phosphorylation] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [loss] present in test data point [True]
12 Text feature [activated] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
21 Text feature [akt] present in test data point [True]
22 Text feature [cells] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [growth] present in test data point [True]
28 Text feature [cell] present in test data point [True]
29 Text feature [therapy] present in test data point [True]
31 Text feature [protein] present in test data point [True]
35 Text feature [kinases] present in test data point [True]
43 Text feature [oncogene] present in test data point [True]
46 Text feature [months] present in test data point [True]
48 Text feature [expression] present in test data point [True]
51 Text feature [drug] present in test data point [True]
53 Text feature [patients] present in test data point [True]
54 Text feature [survival] present in test data point [True]
56 Text feature [downstream] present in test data point [True]
58 Text feature [inhibition] present in test data point [True]
60 Text feature [treated] present in test data point [True]
63 Text feature [dna] present in test data point [True]
65 Text feature [expressing] present in test data point [True]
66 Text feature [imatinib] present in test data point [True]
68 Text feature [mapk] present in test data point [True]
73 Text feature [response] present in test data point [True]
```

```
75 Text feature [clinical] present in test data point [True]
76 Text feature [resistance] present in test data point [True]
82 Text feature [advanced] present in test data point [True]
84 Text feature [ic50] present in test data point [True]
85 Text feature [inhibited] present in test data point [True]
87 Text feature [metastatic] present in test data point [True]
88 Text feature [assays] present in test data point [True]
90 Text feature [sensitivity] present in test data point [True]
91 Text feature [lines] present in test data point [True]
92 Text feature [harboring] present in test data point [True]
99 Text feature [partial] present in test data point [True]
Out of the top  100  features  43 are present in query point
```

Stacking the models

In [169]:

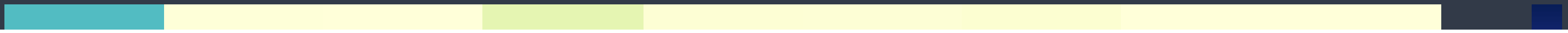
```
clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', class_weight='balanced', ra
ndom_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid",cv=None)

clf2 = SGDClassifier(alpha=.0001, penalty='l2', loss='hinge', class_weight='balanced', r
andom_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid",cv=None)


clf3 = MultinomialNB(alpha=0.01)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid",cv=None)

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_pro
ba(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_oneh
otCoding))))
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_classifie
r=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_lo
ss(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
```

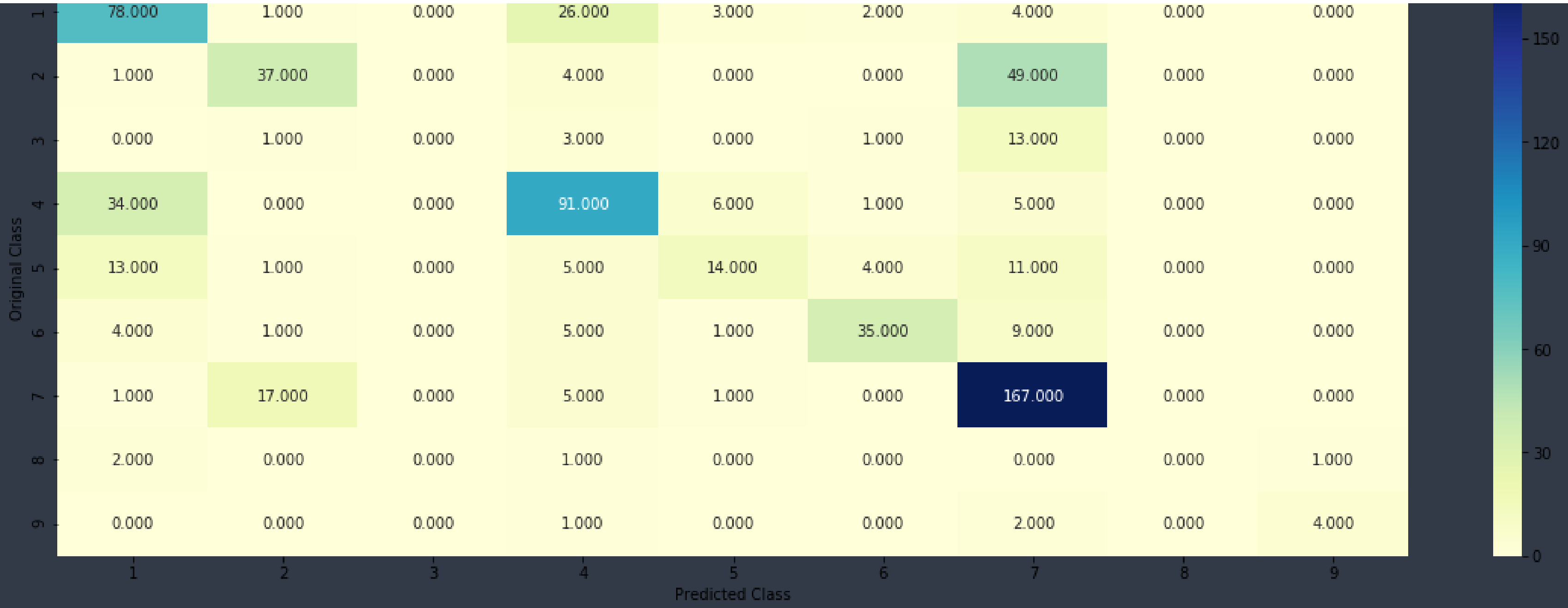
```
Logistic Regression :  Log Loss: 1.04
Support vector machines : Log Loss: 1.07
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.171
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.980
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.398
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.165
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.418
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.871
```

Testing on the best Hyperparameter

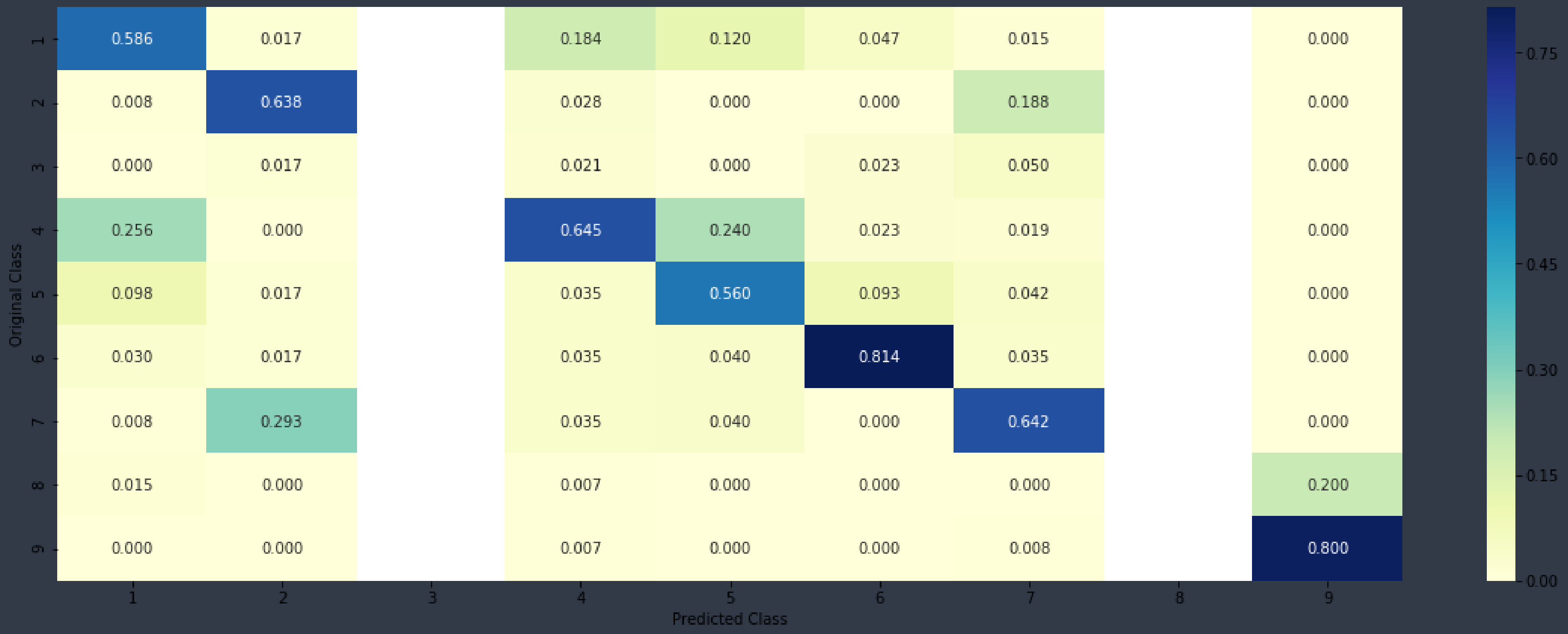
In [171]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr
, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error = log_loss(test_y, sclf.predict_proba(test_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_onehotCo
ding)- 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 : 1.1132412722560951
Log loss (CV) on the stacking classifier : 1.1132412722560951
Log loss (test) on the stacking classifier : 1.1132412722560951
Number of missclassified point : 0.3593984962406015
-------------------- Confusion matrix --------------------
```

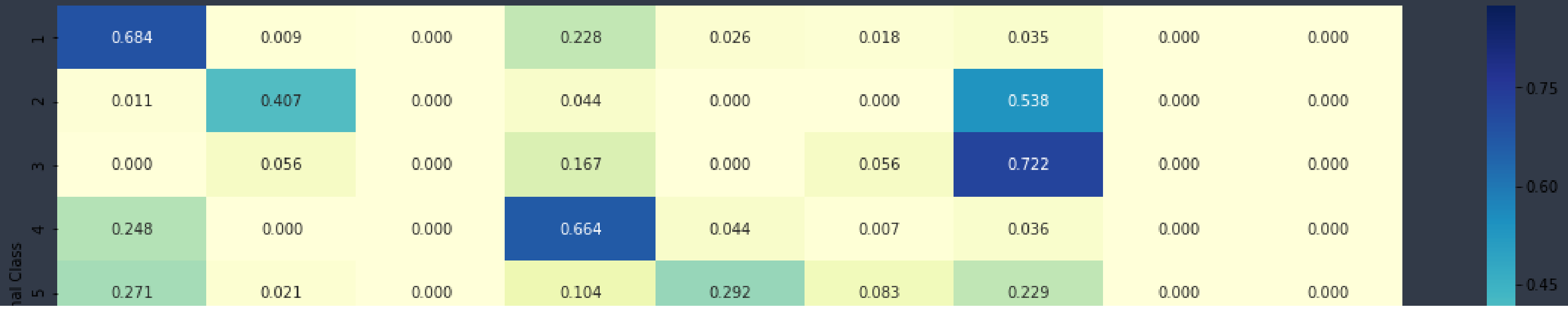


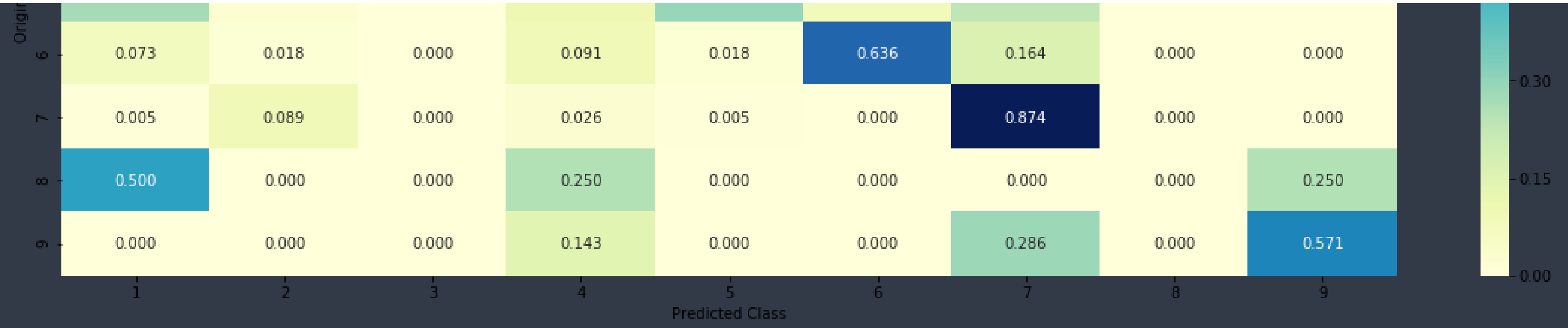


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

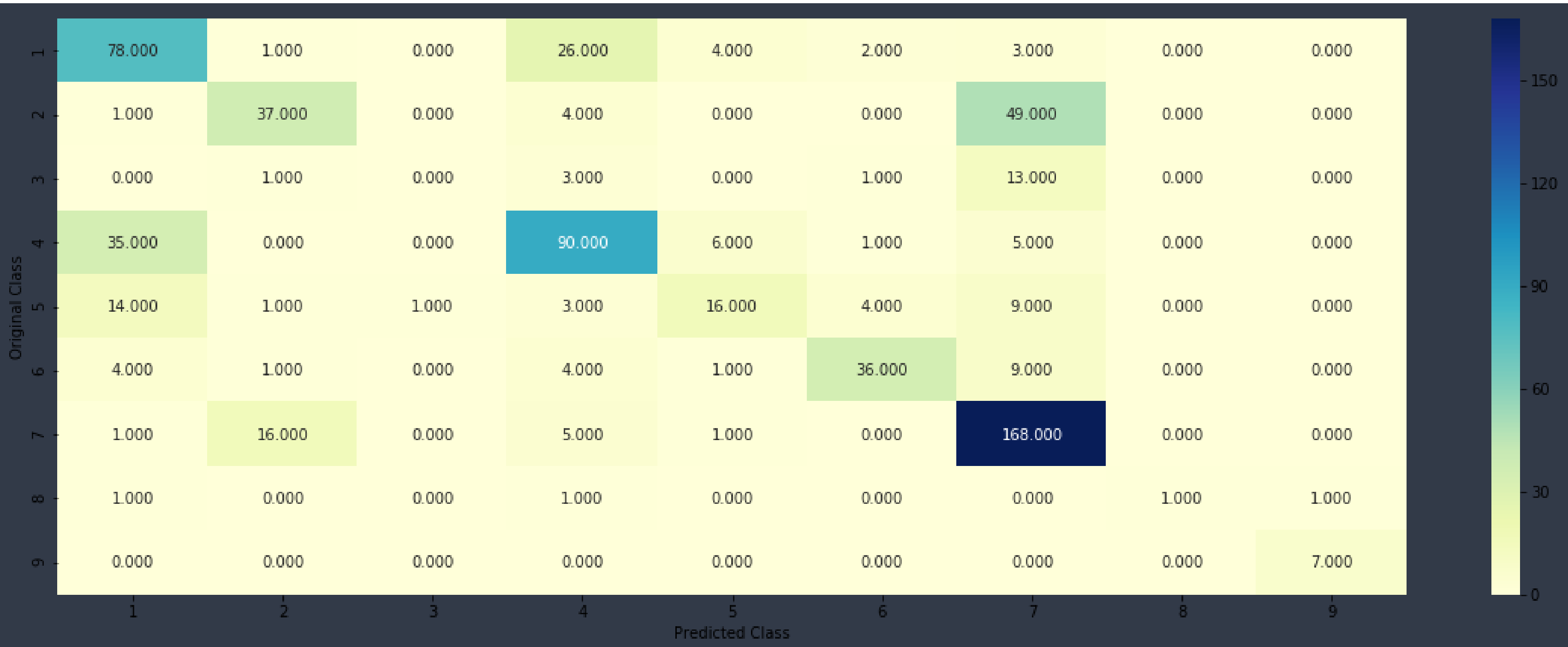




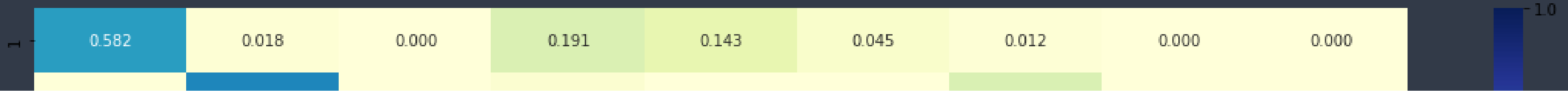
Voting Classifier

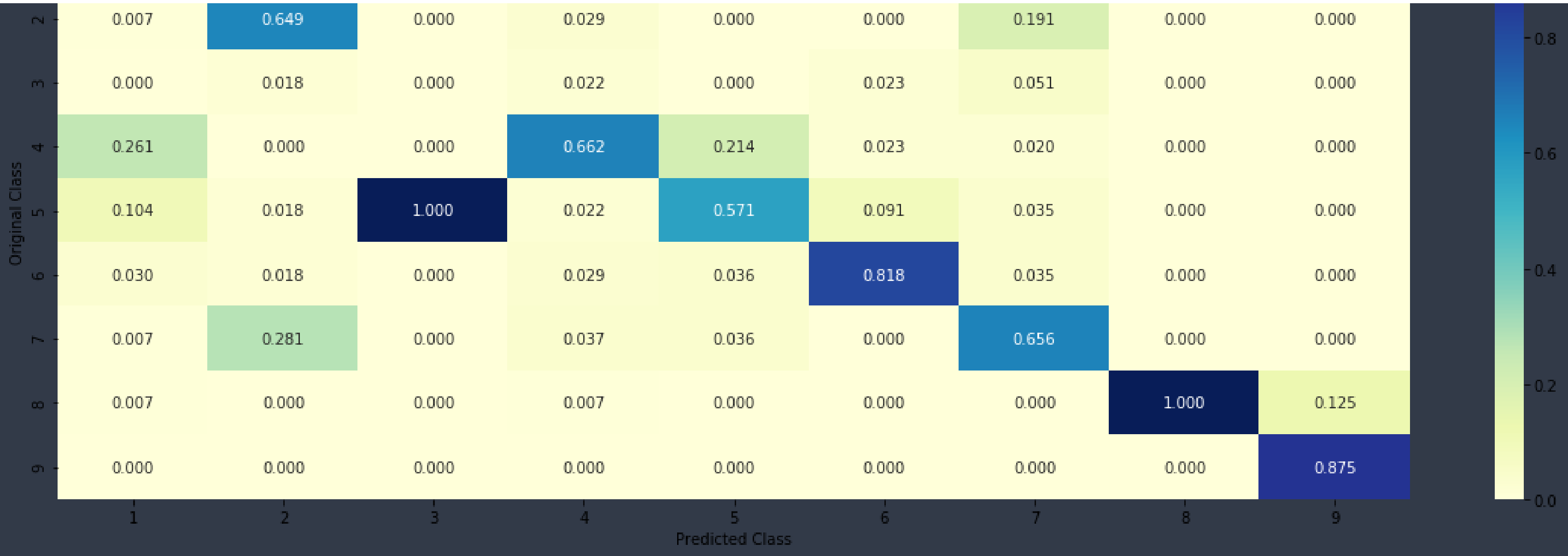
```
In [172]:
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3
)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba
(train_x_onehotCoding)))
print("Log loss (CV) on the 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(t
est_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCo
ding)- 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 : 0.45087866459950166
Log loss (CV) on the VotingClassifier : 1.0634424403380565
Log loss (test) on the VotingClassifier : 1.0035931814918881
Number of missclassified point : 0.34887218045112783
-------------------- Confusion matrix --------------------
```

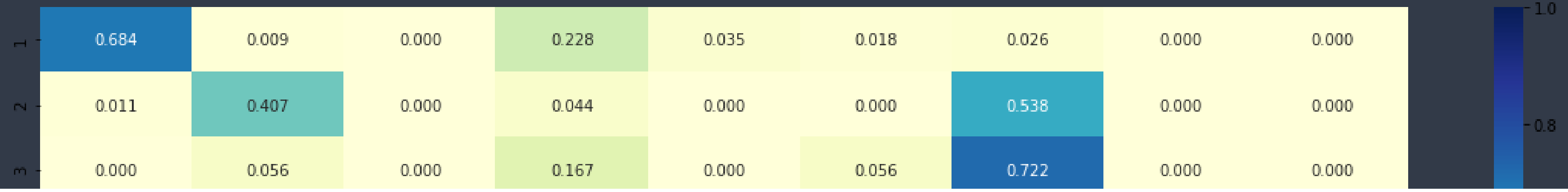


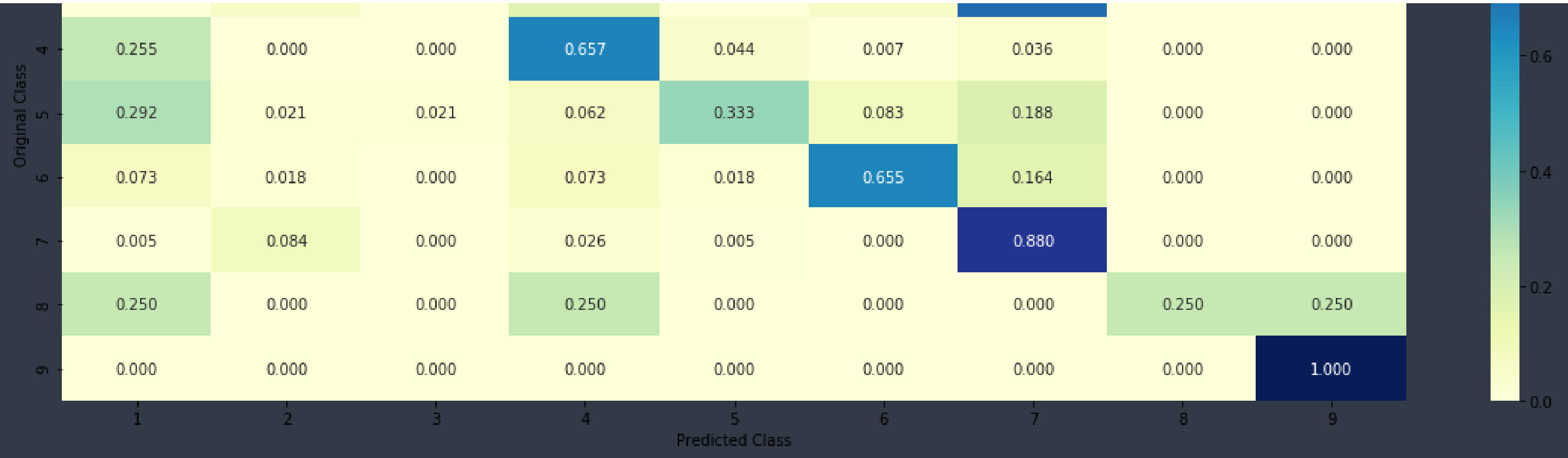
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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





2 Applying Logistic Regression to count Vectorizer inclusing bigrams and unigrams

In [181]:
```
text_vectorizer = CountVectorizer(ngram_range=(1,2),min_df=3,max_features=50000)
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*numbe
r 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 o
ccured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
```

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

```
Total number of unique words in train data : 50000
```

In [182]:
```
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 [198]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_fe
ature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_featu
re_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_one
hotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCod
ing)).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)).toc
sr()
cv_y = np.array(list(cv_df['Class']))



print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 52180)
(number of data points * number of features) in test data =  (665, 52180)
(number of data points * number of features) in cross validation data = (532, 52180)
```

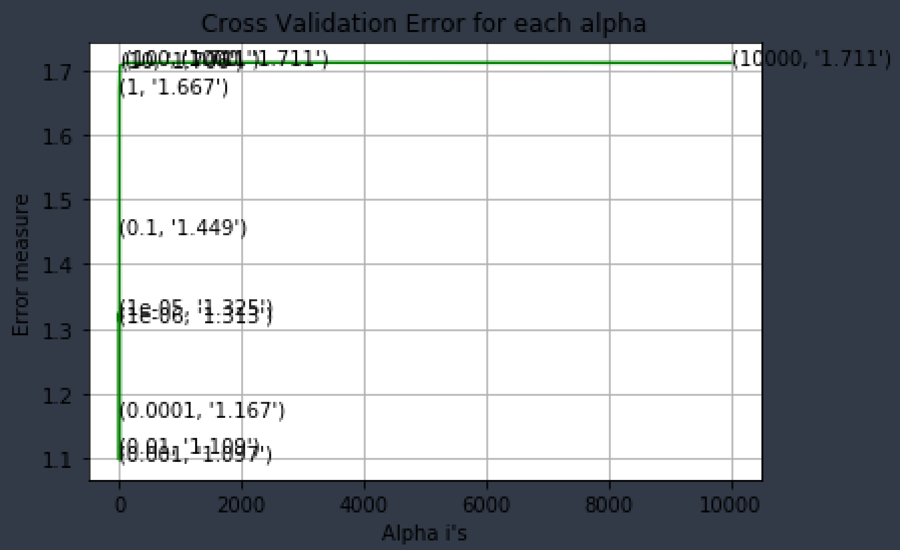
Logistic Regression

In [184]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=5)
    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 esti
mates
    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",cv=5)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.312663941693749
for alpha = 1e-05
Log Loss : 1.3247906772361884
for alpha = 0.0001
Log Loss : 1.1669836354987388
for alpha = 0.001
Log Loss : 1.0972744832951915
for alpha = 0.01
Log Loss : 1.1089677254632957
for alpha = 0.1
Log Loss : 1.4488449769734313
for alpha = 1
Log Loss : 1.667153066145188
for alpha = 10
Log Loss : 1.7063336201869346
for alpha = 100
Log Loss : 1.7109424308321661
for alpha = 1000
Log Loss : 1.7114709100427037
for alpha = 10000
Log Loss : 1.7109768627363455
```

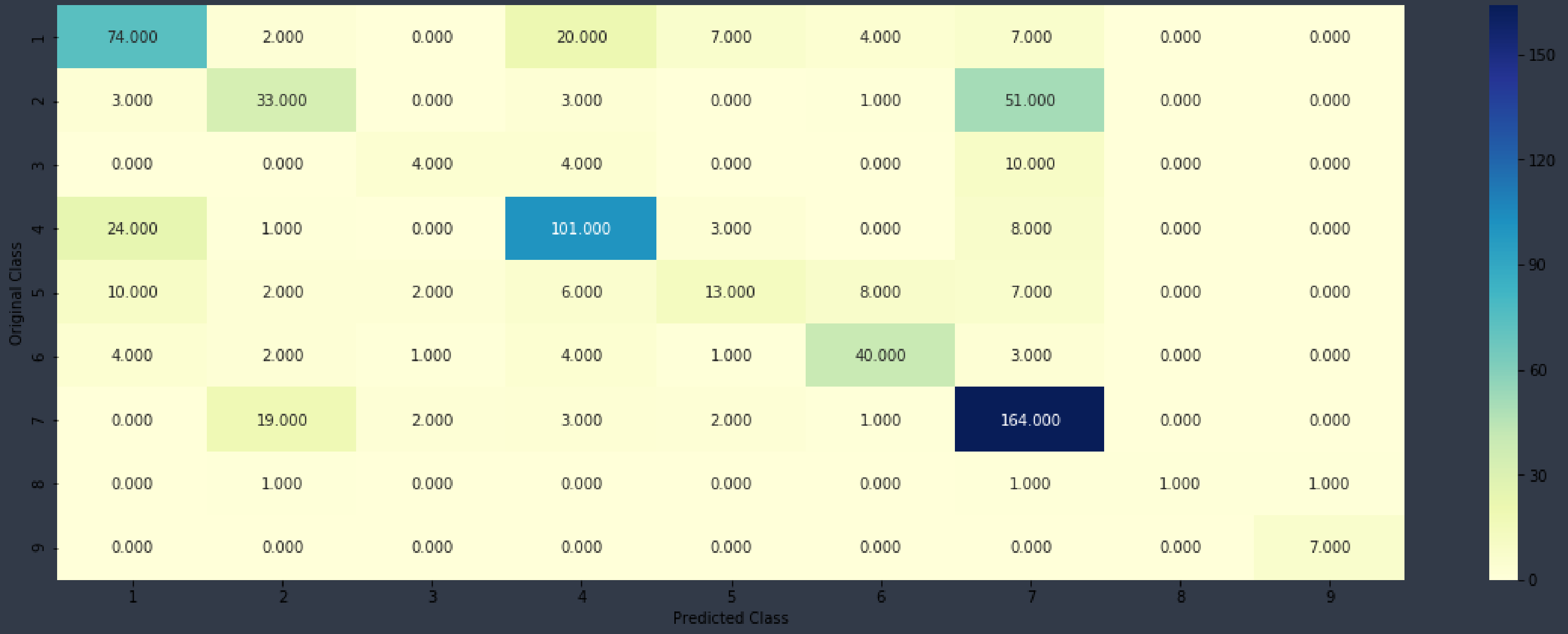


```
For values of best alpha =  0.001 The train log loss is: 0.5411039563708149
For values of best alpha =  0.001 The cross validation log loss is: 1.0972744832951915
For values of best alpha =  0.001 The test log loss is: 1.0145501408978073
```

In [186]:

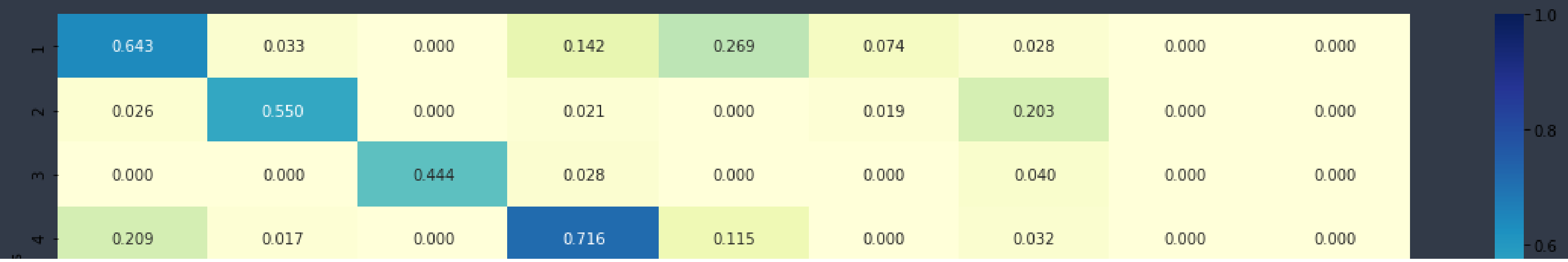
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

```
Log Loss : 1.0041322887229134
Number of missclassified point : 0.34285714285714286
-------------------- Confusion matrix --------------------
```

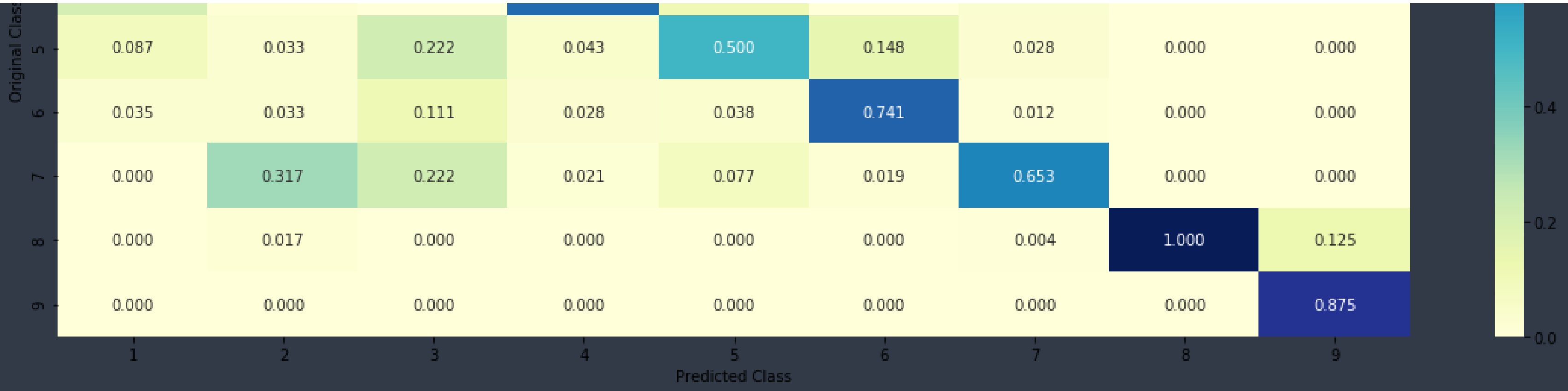


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	74.000	2.000	0.000	20.000	7.000	4.000	7.000	0.000	0.000
2	3.000	33.000	0.000	3.000	0.000	1.000	51.000	0.000	0.000
3	0.000	0.000	4.000	4.000	0.000	0.000	10.000	0.000	0.000
4	24.000	1.000	0.000	101.000	3.000	0.000	8.000	0.000	0.000
5	10.000	2.000	2.000	6.000	13.000	8.000	7.000	0.000	0.000
6	4.000	2.000	1.000	4.000	1.000	40.000	3.000	0.000	0.000
7	0.000	19.000	2.000	3.000	2.000	1.000	164.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	7.000

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

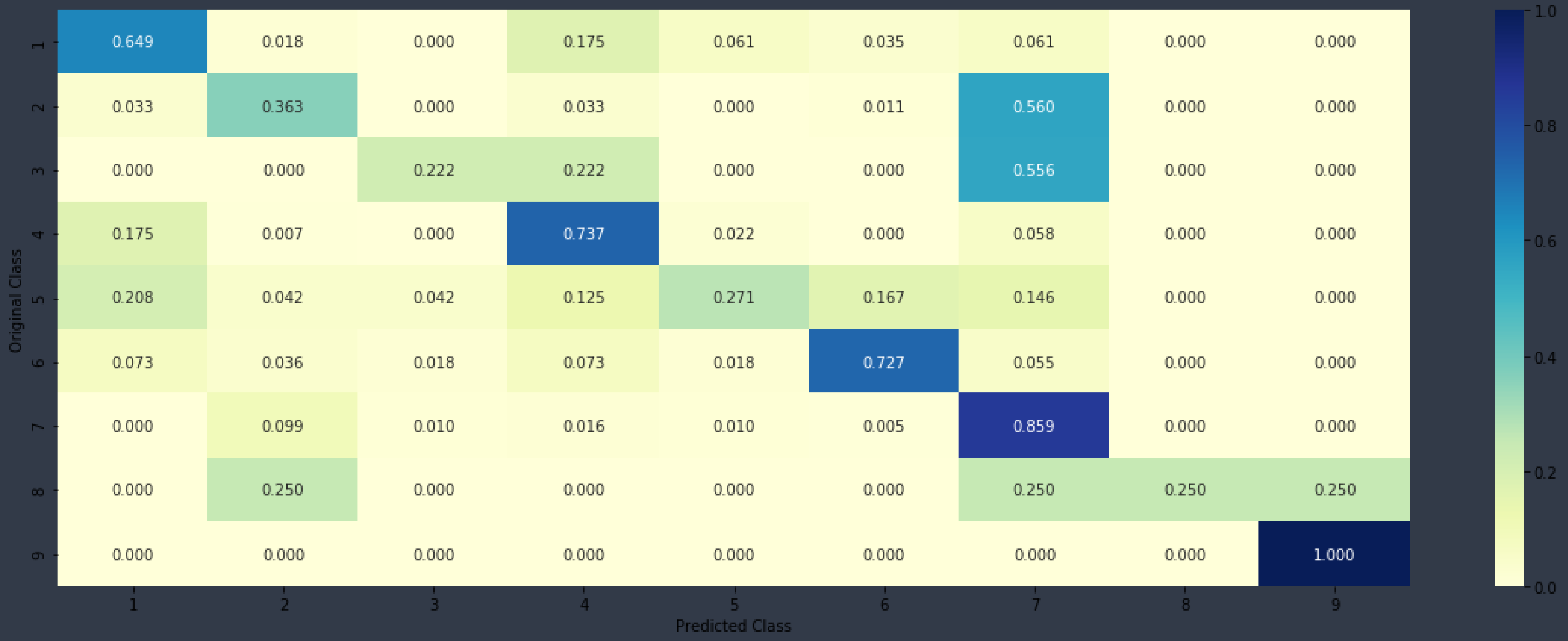


1	0.643	0.033	0.000	0.142	0.269	0.074	0.028	0.000	0.000
2	0.026	0.550	0.000	0.021	0.000	0.019	0.203	0.000	0.000
3	0.000	0.000	0.444	0.028	0.000	0.000	0.040	0.000	0.000
4	0.209	0.017	0.000	0.716	0.115	0.000	0.032	0.000	0.000



	1	2	3	4	5	6	7	8	9
5	0.087	0.033	0.222	0.043	0.500	0.148	0.028	0.000	0.000
6	0.035	0.033	0.111	0.028	0.038	0.741	0.012	0.000	0.000
7	0.000	0.317	0.222	0.021	0.077	0.019	0.653	0.000	0.000
8	0.000	0.017	0.000	0.000	0.000	0.000	0.004	1.000	0.125
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.875

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



	1	2	3	4	5	6	7	8	9
1	0.649	0.018	0.000	0.175	0.061	0.035	0.061	0.000	0.000
2	0.033	0.363	0.000	0.033	0.000	0.011	0.560	0.000	0.000
3	0.000	0.000	0.222	0.222	0.000	0.000	0.556	0.000	0.000
4	0.175	0.007	0.000	0.737	0.022	0.000	0.058	0.000	0.000
5	0.208	0.042	0.042	0.125	0.271	0.167	0.146	0.000	0.000
6	0.073	0.036	0.018	0.073	0.018	0.727	0.055	0.000	0.000
7	0.000	0.099	0.010	0.016	0.010	0.005	0.859	0.000	0.000
8	0.000	0.250	0.000	0.000	0.000	0.000	0.250	0.250	0.250
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

Important features for predicted points

In [192]:
```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(ngram_range=(1,2),min_df=3,max_features=50000)

    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 quer
y point")
```

In [197]:

```
test_point_index = 100
no_feature = 1000
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodi
ng[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3212 0.0471 0.0059 0.4865 0.0192 0.008  0.1005 0.009  0.0025]]
Actual Class : 4
--------------------------------------------------
186 Text feature [suppressor] present in test data point [True]
279 Text feature [dn] present in test data point [True]
427 Text feature [phosphatases] present in test data point [True]
437 Text feature [degradation] present in test data point [True]
609 Text feature [microscopy] present in test data point [True]
669 Text feature [pcmv] present in test data point [True]
```

```
857 Text feature [tgfbr1] present in test data point [True]
888 Text feature [bsa] present in test data point [True]
915 Text feature [pcdna3] present in test data point [True]
928 Text feature [lacks] present in test data point [True]
931 Text feature [tagged] present in test data point [True]
965 Text feature [tgf] present in test data point [True]
Out of the top  1000  features  12 are present in query point
```

Experimienting some feature engineering techniques

Combining tfidf nad Bow vectorizer with unigram and bigrams and also the response coding feature and applying balanced Logistic Regression

In [199]:

```
tfidf_text_vectorizer = TfidfVectorizer(min_df=3)
tfidf_train_text_feature_onehotCoding = tfidf_text_vectorizer.fit_transform(train_df['TE
XT'])
# getting all the feature names (words)
tfidf_train_text_features= tfidf_text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*numbe
r of features) vector
tfidf_train_text_fea_counts = tfidf_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 o
ccured
tfidf_text_fea_dict = dict(zip(list(tfidf_train_text_features),tfidf_train_text_fea_coun
ts))
```

```
print("Total number of unique words in train data :", len(tfidf_train_text_features))
tfidf_train_text_feature_onehotCoding = normalize(tfidf_train_text_feature_onehotCoding,
 axis=0)

# we use the same vectorizer that was trained on train data
tfidf_test_text_feature_onehotCoding = tfidf_text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
tfidf_test_text_feature_onehotCoding = normalize(tfidf_test_text_feature_onehotCoding, a
xis=0)

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

```
Total number of unique words in train data : 53645
```

In [200]:

```
train_x_onehotCoding = hstack((train_x_onehotCoding,tfidf_train_text_feature_onehotCodin
g)).tocsr()
test_x_onehotCoding = hstack((test_x_onehotCoding, tfidf_test_text_feature_onehotCoding
)).tocsr()
cv_x_onehotCoding = hstack((cv_x_onehotCoding, tfidf_cv_text_feature_onehotCoding)).tocs
r()

train_x_onehotCoding = hstack((train_x_onehotCoding,train_x_responseCoding)).tocsr()
test_x_onehotCoding = hstack((test_x_onehotCoding, test_x_responseCoding)).tocsr()
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_x_responseCoding)).tocsr()
```

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCod
ing.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCodin
g.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_on
ehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 105852)
(number of data points * number of features) in test data =  (665, 105852)
(number of data points * number of features) in cross validation data = (532, 105852)
```

In [ ]:

```
###
```

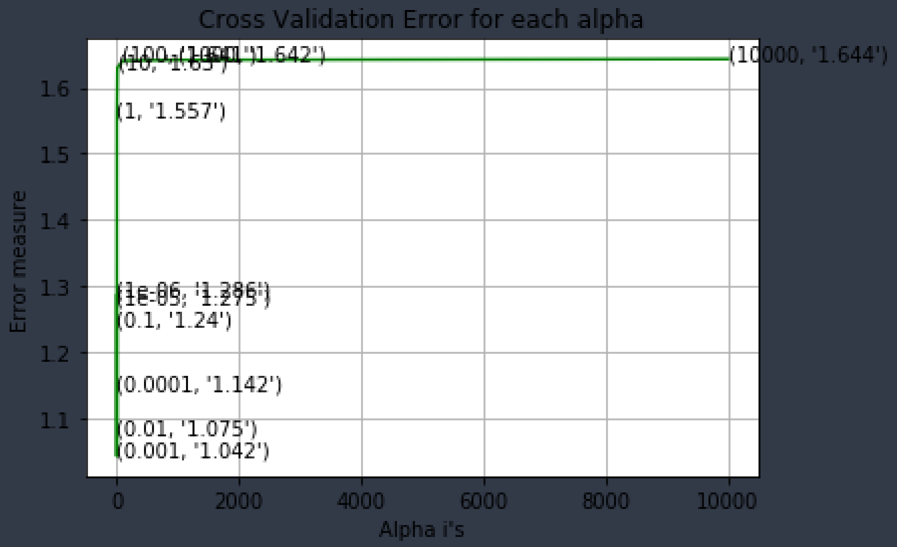
In [203]:

```
alpha = [10**i for i in range(-6,5) ]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=3)
    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 esti
mates
    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",cv=3)
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_los
s(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 i
s:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss
(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.2855738303938442
```

```
for alpha = 1e-05
Log Loss : 1.2750659790760819
for alpha = 0.0001
Log Loss : 1.1418386735464319
for alpha = 0.001
Log Loss : 1.041743220913038
for alpha = 0.01
Log Loss : 1.0753196088837385
for alpha = 0.1
Log Loss : 1.2398815001593497
for alpha = 1
Log Loss : 1.5569954355547513
for alpha = 10
Log Loss : 1.630400609890844
for alpha = 100
Log Loss : 1.64094499589639
for alpha = 1000
Log Loss : 1.6421676663646674
for alpha = 10000
Log Loss : 1.6435043726220684
```

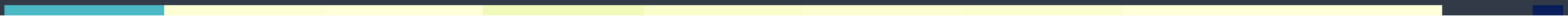


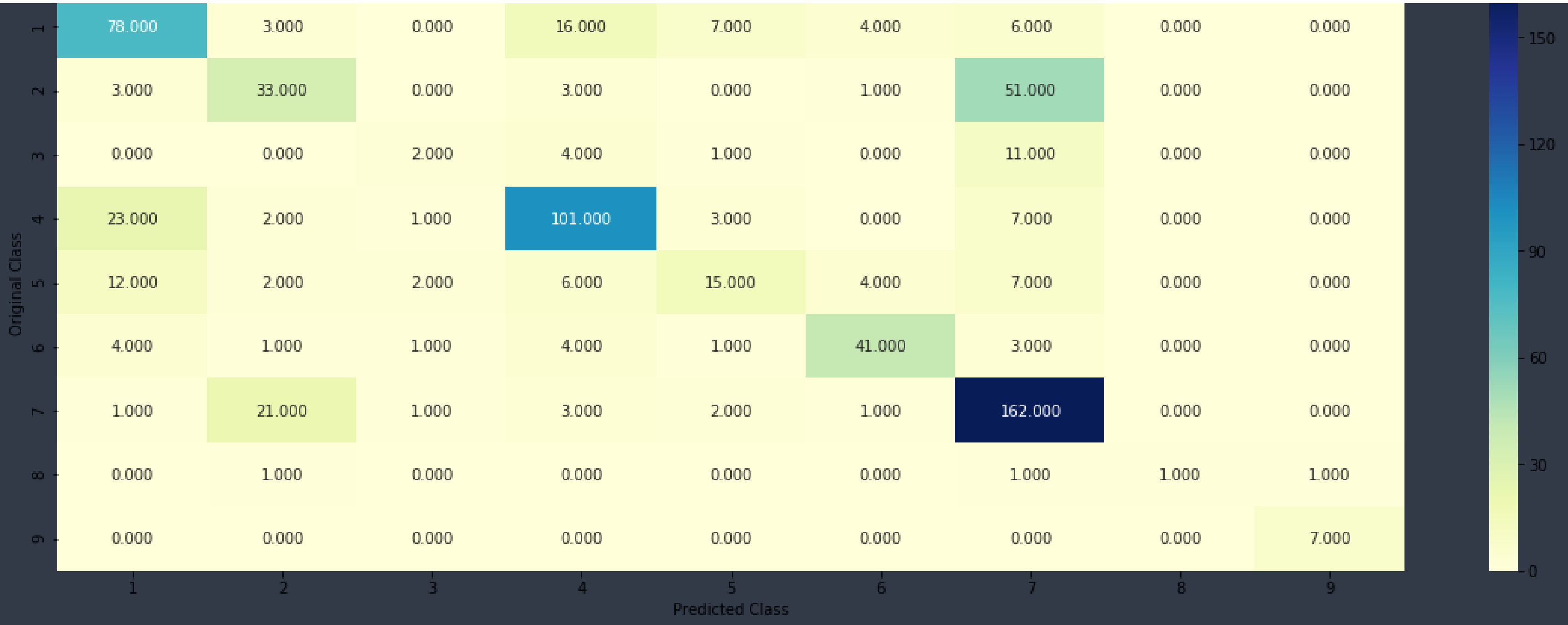
```
For values of best alpha =  0.001 The train log loss is: 0.5403156525032562
For values of best alpha =  0.001 The cross validation log loss is: 1.041743220913038
For values of best alpha =  0.001 The test log loss is: 0.9706049728214963
```

Here we can see that we are able to reduce the test log_loss to 0.97060 less than 1

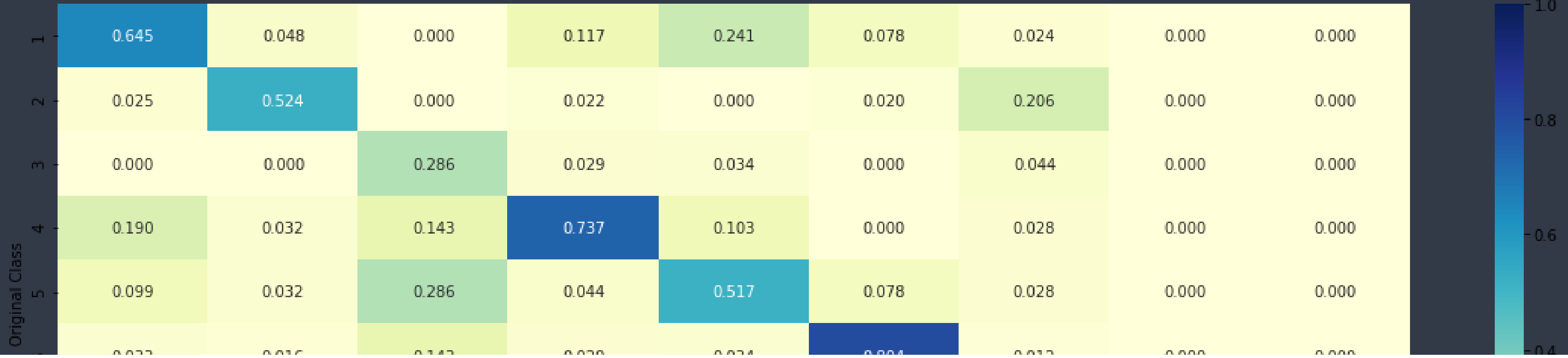
```
In [204]:
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid",cv=None)
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimate
s
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(test_x_oneho
tCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

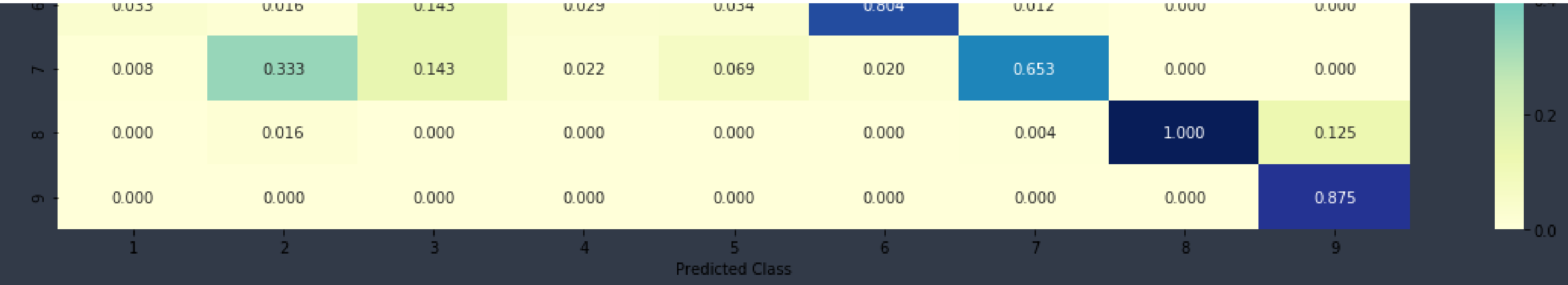
```
Log Loss : 0.9706049728214963
Number of missclassified point : 0.3383458646616541
-------------------- Confusion matrix --------------------
```



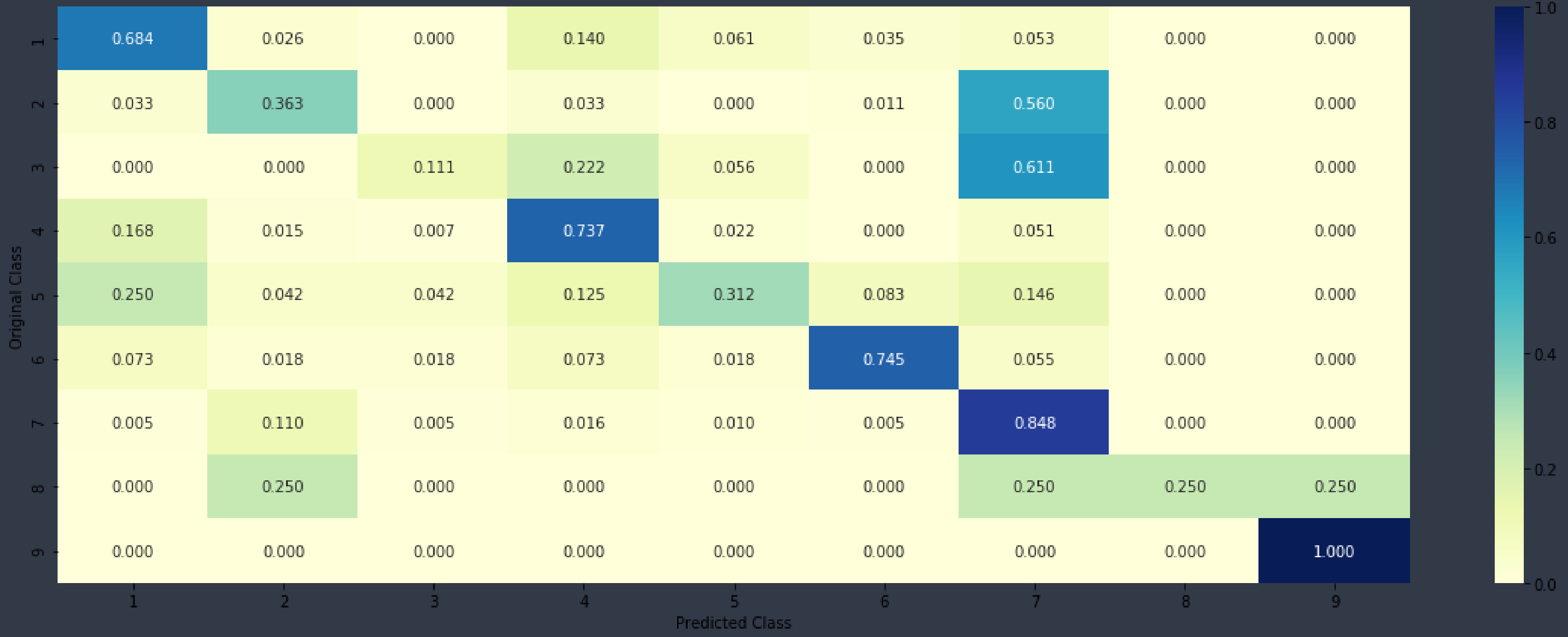


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Conclusion

This model is the best model we have got with test logg_loss of 0.97 and wrong prediction percentage of 33%