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 (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 (https://www.youtube.com/watch?v=UwbuW7oK8rk)
3. https://www.youtube.com/watch?v=qxXRKVompI8 (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 (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 (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

Task 1 : Replace CountVectorizer() by TfidfVectorizer()

3. Exploratory Data Analysis

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

from mlxtend.classifier import StackingClassifier

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

```
Using TensorFlow backend.
d:\wwin_env\lib\site-packages\tensorflow\python\framework\dtypes.py:523: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
d:\wwin_env\lib\site-packages\tensorflow\python\framework\dtypes.py:524: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
d:\wwin_env\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
d:\wwin_env\lib\site-packages\tensorflow\python\framework\dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
d:\wwin_env\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
d:\wwin_env\lib\site-packages\tensorflow\python\framework\dtypes.py:532: FutureWarning: Passing (type, 1) or '1type' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
import os
os.getcwd()
```

Out[2]:
```
'D:\\applied_ai_course_practice\\assignments_2\\Assignments_DonorsChoose_2018\\case_study\\Case_study_2'
```

In [ ]:

In [4]:
```
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']
```

Out[4]:

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

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 [5]:
```
# 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']
```

Out[5]:

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

3.1.3. Preprocessing of text

In [6]:
```
# loading stop words from nltk library
import nltk
# nltk.download('stopwords')

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 [7]:
```
#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 : 36.877283399999996 seconds
```

In [8]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[8]:

	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 [9]:
```
result[result.isnull().any(axis=1)]
```

Out[9]:

	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 [10]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [11]:
```
result[result['ID']==1109]
```

Out[11]:

	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 [12]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

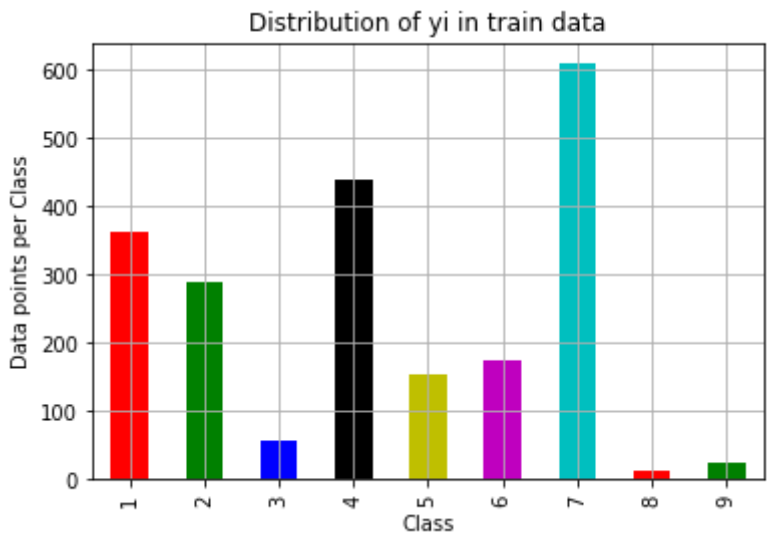
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [13]:
```
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])
```

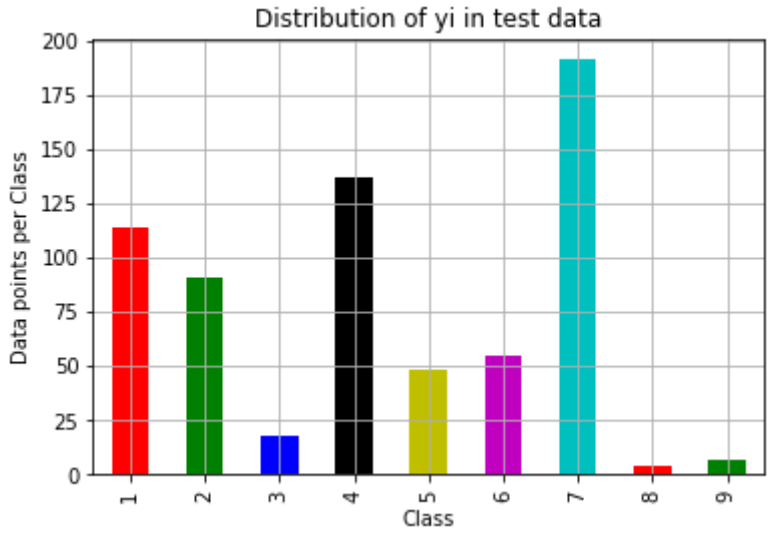
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

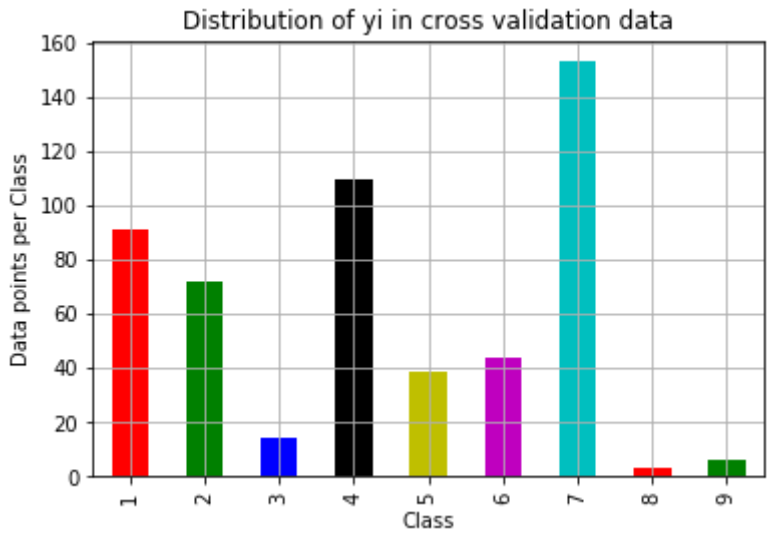
In [14]:
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

my_colors = ['r','g','b','k','y','m','c']
train_class_distribution.plot(kind='bar',color=my_colors)
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_dist


print('-'*80)
# my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar',color=my_colors)
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_distri

print('-'*80)
# my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar',color=my_colors)
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_distributi
```




```
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 %)
```

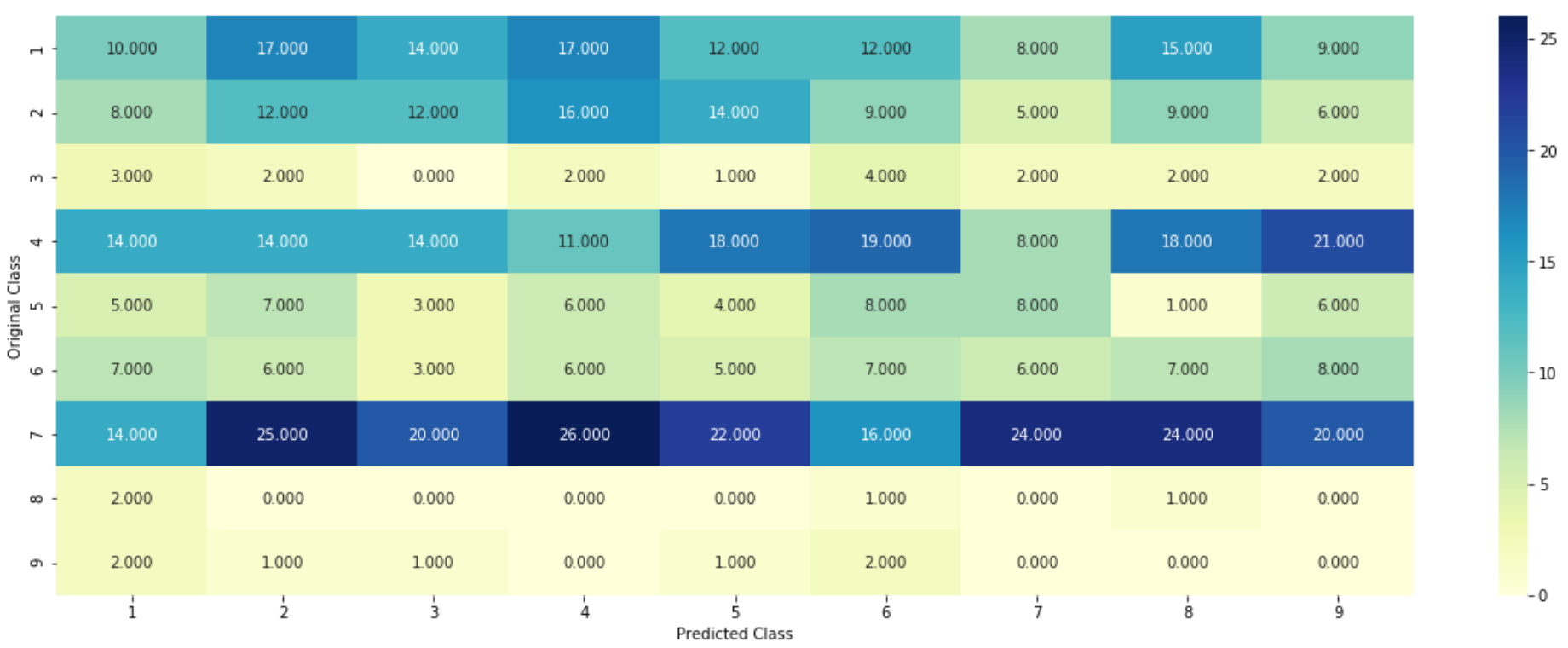
3.2 Prediction using a 'Random' Model

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

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

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

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

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

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

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

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.536542521817824
Log loss on Test Data using Random Model 2.4619279657881608
-------------------- Confusion matrix --------------------
```




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


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

3.3 Univariate Analysis

In [17]:
```
# code for response coding with Laplace smoothing.
# ----------------------

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we will add t
    # 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?

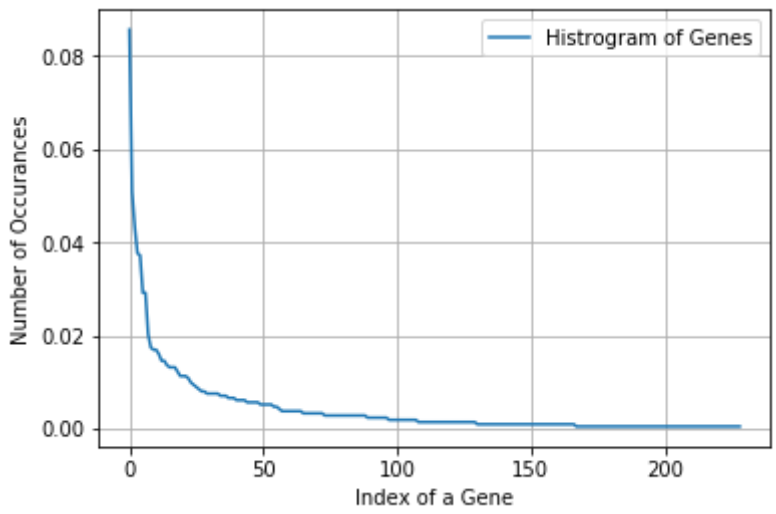
In [18]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 229
BRCA1     182
TP53      108
EGFR       92
PTEN       80
BRCA2      79
BRAF       62
KIT        62
ERBB2      43
ALK        37
PDGFRA     36
Name: Gene, dtype: int64
```

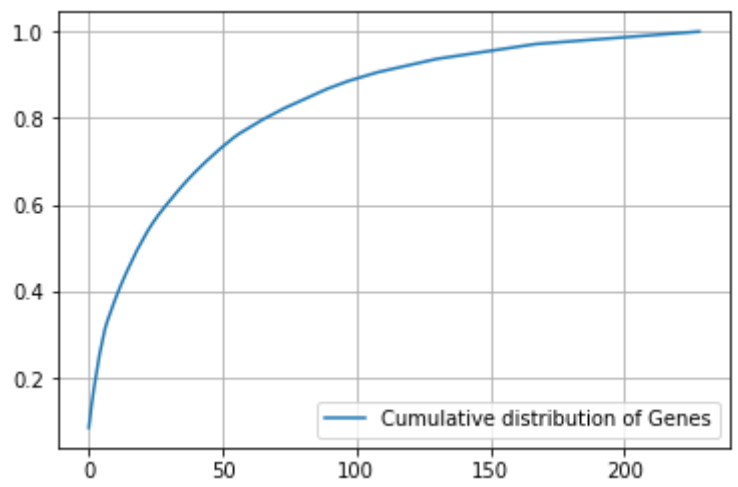
In [19]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, and they are distibuted
```

```
Ans: There are 229 different categories of genes in the train data, and they are distibuted as follows
```

In [20]:
```
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 [21]:
```
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 [22]:
```
#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 [23]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The shape of gene feature:",
```

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

In [24]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
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 [25]:
```
train_df['Gene'].head()
```

Out[25]:
```
327     ROS1
1759    IDH1
3122    KRAS
1302    MLH1
242     EGFR
Name: Gene, dtype: object
```

In [26]:
```
gene_vectorizer.get_feature_names()
```

```
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf3',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt1',
 'flt3',
 'foxa1',
 'foxl2',
 'foxp1',
```

In [27]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:",
```

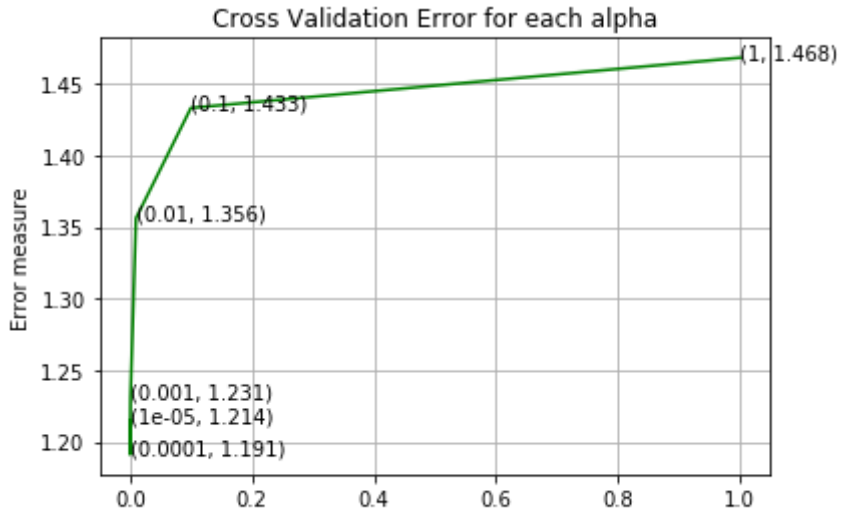
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 2
29)
```

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 [28]:
```
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.2143176195070209
For values of alpha =  0.0001 The log loss is: 1.1911127561743882
For values of alpha =  0.001 The log loss is: 1.2311929583261052
For values of alpha =  0.01 The log loss is: 1.3561045424768787
For values of alpha =  0.1 The log loss is: 1.4330643710636428
For values of alpha =  1 The log loss is: 1.4681123457028933
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0036741633733037
For values of best alpha =  0.0001 The cross validation log loss is: 1.1911127561743882
For values of best alpha =  0.0001 The test log loss is: 1.1738995614628518
```

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 [29]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in train da

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  229  genes in train dataset?
Ans
1. In test data 639 out of 665 : 96.09022556390977
2. In cross validation data 509 out of  532 : 95.67669172932331
```

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?

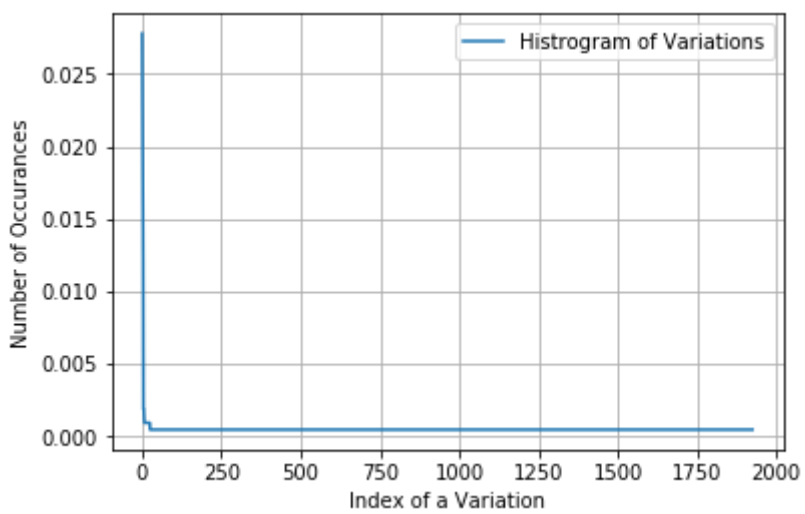
In [30]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1925
Truncating_Mutations    59
Deletion                48
Amplification           44
Fusions                 26
G12V                     4
Overexpression           4
T58I                     3
EWSR1-ETV1_Fusion        2
G12A                     2
T167A                    2
Name: Variation, dtype: int64
```

In [31]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the train data, and they are
```

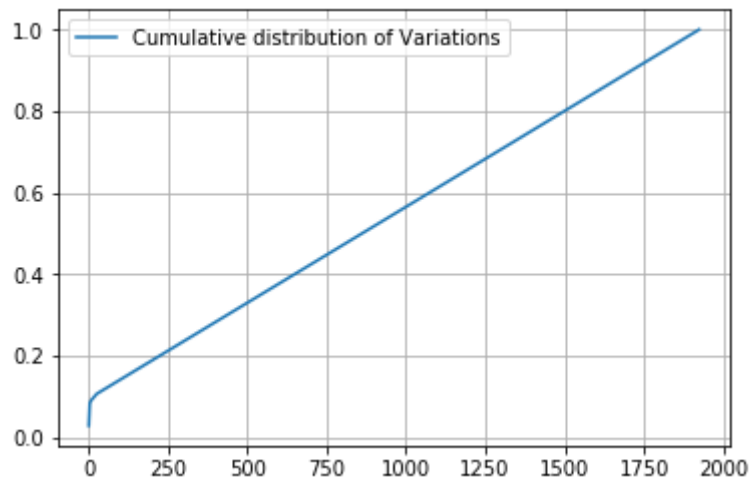
```
Ans: There are 1925 different categories of variations in the train data, and they are distibuted as follows
```

In [32]:
```
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 [33]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02777778 0.05037665 0.07109228 ... 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 [34]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [35]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Vari
```

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

In [36]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [37]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Varia
```

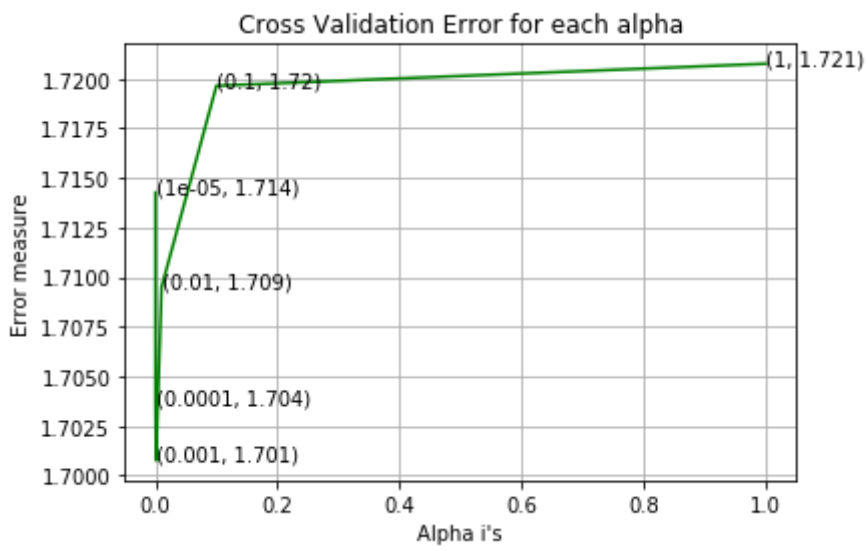
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation f
eature: (2124, 1951)
```

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

Let's build a model just like the earlier!

In [38]:
```
alpha = [10 ** x for x in range(-5, 1)]


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.714267082420089
For values of alpha =  0.0001 The log loss is: 1.7036434230177617
For values of alpha =  0.001 The log loss is: 1.7007619872095474
For values of alpha =  0.01 The log loss is: 1.7094933891604993
For values of alpha =  0.1 The log loss is: 1.7196512765864045
For values of alpha =  1 The log loss is: 1.7207625977363186
```




```
For values of best alpha =  0.001 The train log loss is: 1.0904906124055396
For values of best alpha =  0.001 The cross validation log loss is: 1.7007619872095474
For values of best alpha =  0.001 The test log loss is: 1.7063038942708804
```

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 [39]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validatio
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1925  genes in test and cross validation data sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 54 out of  532 : 10.150375939849624
```

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 [40]:
```
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 [41]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [42]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 53434
```

In [43]:
```
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 [44]:
```
#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 [45]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1)).T
```

In [46]:
```
# 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 [47]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

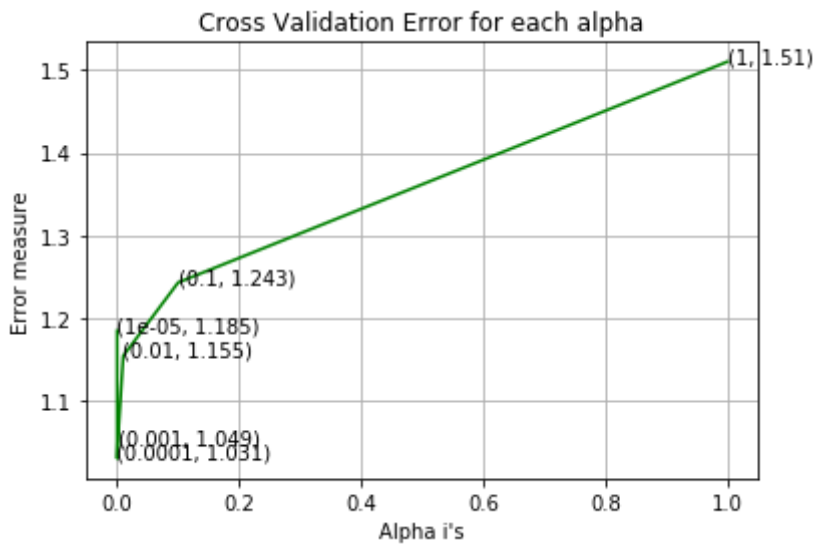
In [48]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
9190233: 1, 0.05823100306945728: 1, 0.0979955933572199: 1, 0.42661186695397163: 1, 3.2700557148152827: 1, 0.027600755
497549957: 1, 0.038456720512585384: 1, 0.3513104090313337: 1, 0.0293048610747639: 1, 0.10815852960942735: 1, 0.127575
18719356534: 1, 0.08689629442813492: 1, 0.11707031515568544: 1, 0.06434256300947291: 1, 0.17814766794077805: 1, 0.049
160657695585155: 1, 0.13559153704706522: 1, 1.3927528047387825: 1, 0.025018271222152437: 1, 0.008202033287859154: 1,
0.028406317073542076: 1, 0.12459282488861748: 1, 0.6566892376020795: 1, 0.21959657716792083: 1, 0.06959197020433365:
1, 0.16817551890574586: 1, 0.0368805221187444: 1, 0.2808346585881971: 1, 0.10688745535346028: 1, 0.09883200487384483:
1, 0.012892939258942804: 1, 0.01602030065575051: 1, 0.12319766854999104: 1, 0.11694307639723468: 1, 0.095415311681649
17: 1, 0.024640012013061242: 1, 2.2681495894238908: 1, 0.049482125432310536: 1, 0.7591048240019442: 1, 0.047073618014
99268: 1, 0.022365474113713136: 1, 0.03160422054436528: 1, 0.02836928202606325: 1, 0.26741732327406326: 1, 0.67877350
77366575: 1, 0.8158650675260805: 1, 0.15536888837621357: 1, 0.16703110291204645: 1, 0.1770156730847508: 1, 0.51095595
1094857: 1, 0.13328501314100616: 1, 0.6999663496951457: 1, 0.45737632271478285: 1, 0.0665642548095819: 1, 0.273466466
67566793: 1, 0.13185400830644597: 1, 0.04036578641717106: 1, 0.085747131150174: 1, 0.3198636697116834: 1, 0.052359861
783946665: 1, 0.7782054924794368: 1, 0.025656663170878568: 1, 0.08034152323437675: 1, 0.33482268637588347: 1, 5.96277
35358475515: 1, 0.029024409972975604: 1, 0.01680403832321599: 1, 0.05361108909700848: 1, 0.21923466759377283: 1, 0.11
122488633977012: 1, 0.02415958072114745: 1, 0.15460667802594105: 1, 0.24017196384248063: 1, 0.2691993271433872: 1, 0.
09519822759437142: 1, 0.6765431926109853: 1, 0.07653186791855561: 1, 0.03007013710853881: 1, 0.6562691046399154: 1,
0.042474626012263525: 1, 1.7199192001384058: 1, 0.04840580268519623: 1, 0.060170014647757564: 1, 0.05046836987665484
5: 1, 0.07274894929616486: 1, 2.779358233400238: 1, 0.03212618689162201: 1, 0.24174179363716683: 1, 0.401369570717643
9: 1, 0.16450479642308302: 1, 0.21899237242240147: 1, 0.06362133704145027: 1, 1.1650004441091792: 1, 0.03996738752050
603: 1, 0.018468175916401344: 1, 0.02385298645598924: 1, 0.03766029195171794: 1, 0.4324716204734418: 1, 0.06818434176
```

In [49]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html


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

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

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

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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.185227249436652
For values of alpha =  0.0001 The log loss is: 1.0310222685039376
For values of alpha =  0.001 The log loss is: 1.0488113214273338
For values of alpha =  0.01 The log loss is: 1.1548030843524393
For values of alpha =  0.1 The log loss is: 1.24339724498204
For values of alpha =  1 The log loss is: 1.5099022777864837
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6828780794301307
For values of best alpha =  0.0001 The cross validation log loss is: 1.0310222685039376
For values of best alpha =  0.0001 The test log loss is: 1.144341329455188
```

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

Ans. Yes, it seems like!

In [50]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(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 [51]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
95.842 % of word of test data appeared in train data
97.481 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [52]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [53]:
```
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 [54]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

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

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

Stacking the three types of features

In [55]:
```
# merging gene, variance and text features

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

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

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

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


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

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

In [56]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55614)
(number of data points * number of features) in test data =  (665, 55614)
(number of data points * number of features) in cross validation data = (532, 55614)
```

In [57]:
```
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

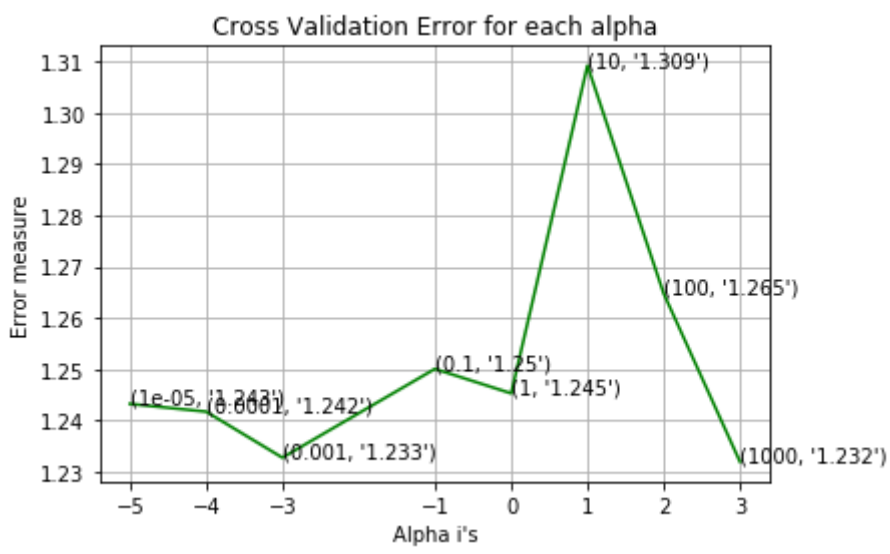
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [58]:

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

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for alpha = 1e-05
Log Loss : 1.2432361255429487
for alpha = 0.0001
Log Loss : 1.2417169739183116
for alpha = 0.001
Log Loss : 1.2327529152359675
for alpha = 0.1
Log Loss : 1.2500451684112275
for alpha = 1
Log Loss : 1.245279810773913
for alpha = 10
Log Loss : 1.309144917714293
for alpha = 100
Log Loss : 1.2647270184700168
for alpha = 1000
Log Loss : 1.2319731549171185
```

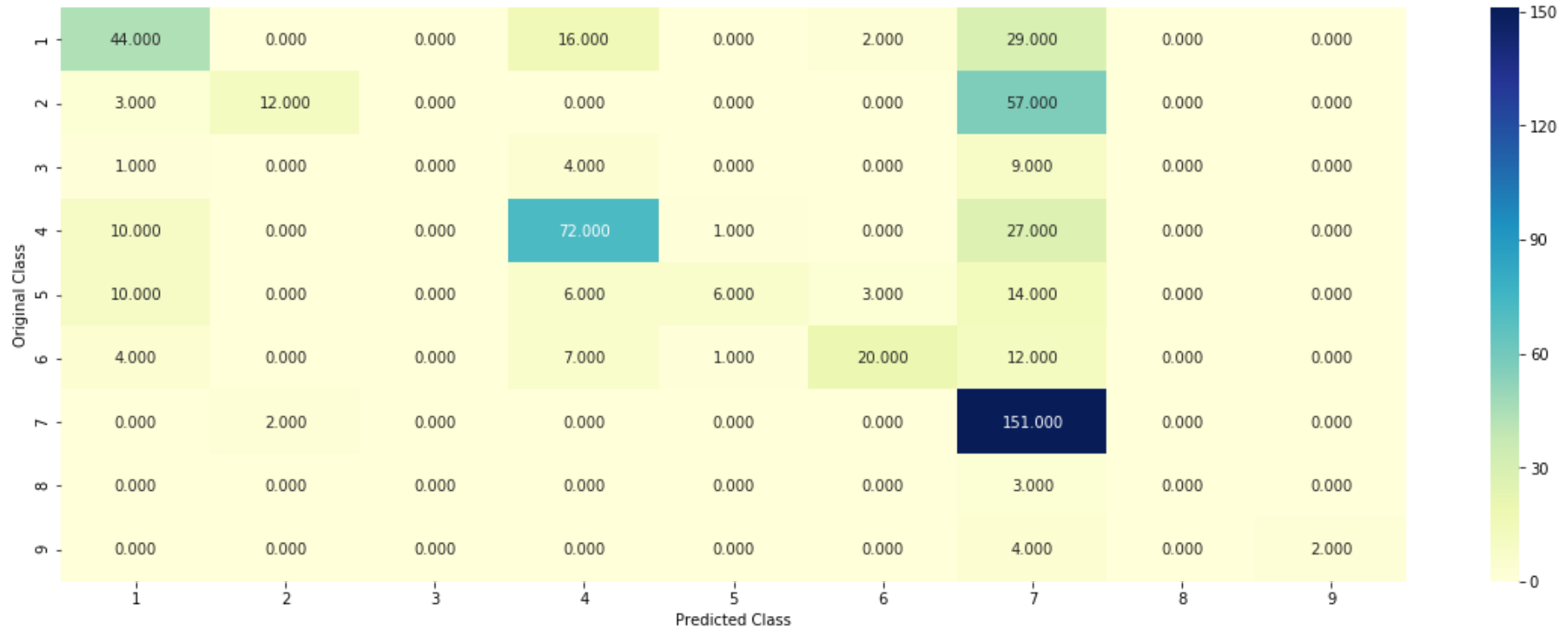



```
For values of best alpha =  1000 The train log loss is: 0.93868819827066
For values of best alpha =  1000 The cross validation log loss is: 1.2319731549171185
For values of best alpha =  1000 The test log loss is: 1.2201211988688265
```

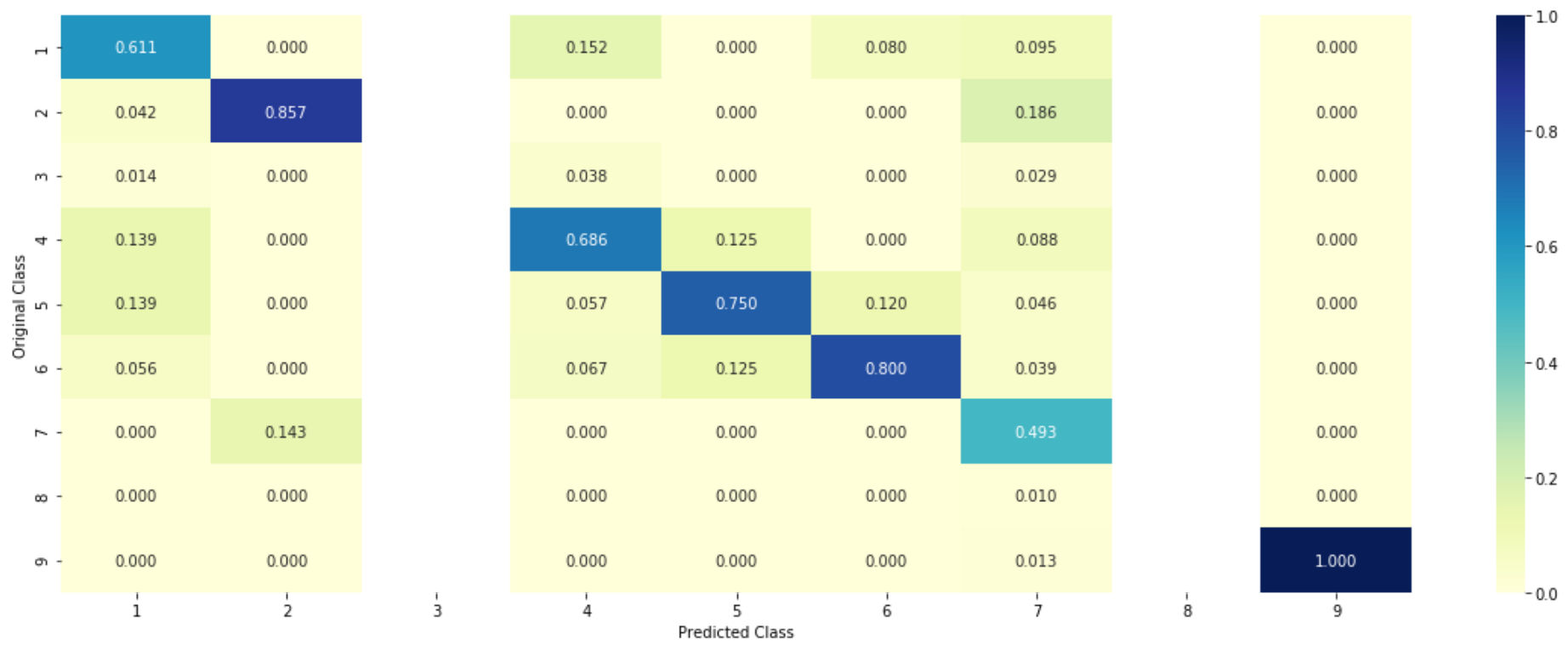
4.1.1.2. Testing the model with best hyper paramters

In [59]:
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

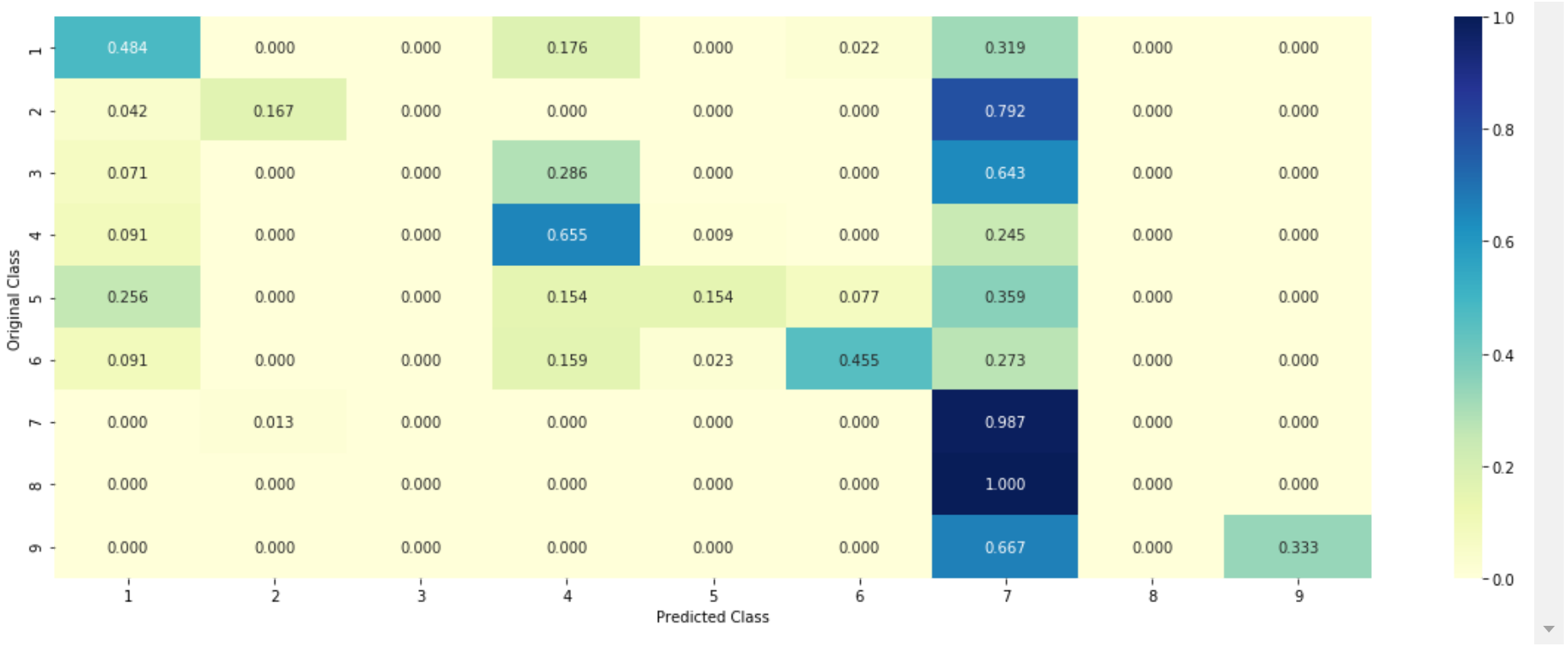
```
Log Loss : 1.2319731549171185
Number of missclassified point : 0.42293233082706766
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [60]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.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['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[4.625e-01 1.700e-03 7.000e-04 2.611e-01 2.458e-01 1.740e-02 1.070e-02
  2.000e-04 1.000e-04]]
Actual Class : 1
--------------------------------------------------
9 Text feature [protein] present in test data point [True]
10 Text feature [type] present in test data point [True]
16 Text feature [dna] present in test data point [True]
17 Text feature [one] present in test data point [True]
18 Text feature [results] present in test data point [True]
19 Text feature [wild] present in test data point [True]
20 Text feature [region] present in test data point [True]
21 Text feature [two] present in test data point [True]
22 Text feature [binding] present in test data point [True]
23 Text feature [containing] present in test data point [True]
25 Text feature [involved] present in test data point [True]
26 Text feature [therefore] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [using] present in test data point [True]
29 Text feature [control] present in test data point [True]
30 Text feature [role] present in test data point [True]
31 Text feature [table] present in test data point [True]
32 Text feature [expression] present in test data point [True]
33 Text feature [indicated] present in test data point [True]
34 Text feature [possible] present in test data point [True]
35 Text feature [human] present in test data point [True]
37 Text feature [either] present in test data point [True]
38 Text feature [indicate] present in test data point [True]
39 Text feature [determined] present in test data point [True]
40 Text feature [specific] present in test data point [True]
41 Text feature [sequences] present in test data point [True]
42 Text feature [reporter] present in test data point [True]
43 Text feature [gene] present in test data point [True]
44 Text feature [affect] present in test data point [True]
45 Text feature [following] present in test data point [True]
46 Text feature [four] present in test data point [True]
47 Text feature [three] present in test data point [True]
48 Text feature [function] present in test data point [True]
49 Text feature [however] present in test data point [True]
50 Text feature [shown] present in test data point [True]
51 Text feature [loss] present in test data point [True]
52 Text feature [indicating] present in test data point [True]
53 Text feature [effect] present in test data point [True]
55 Text feature [present] present in test data point [True]
56 Text feature [transcriptional] present in test data point [True]
57 Text feature [ability] present in test data point [True]
58 Text feature [result] present in test data point [True]
59 Text feature [important] present in test data point [True]
60 Text feature [several] present in test data point [True]
61 Text feature [suggest] present in test data point [True]
62 Text feature [obtained] present in test data point [True]
63 Text feature [essential] present in test data point [True]
64 Text feature [analysis] present in test data point [True]
65 Text feature [amino] present in test data point [True]
66 Text feature [p53] present in test data point [True]
67 Text feature [corresponding] present in test data point [True]
68 Text feature [transcription] present in test data point [True]
69 Text feature [may] present in test data point [True]
71 Text feature [length] present in test data point [True]
73 Text feature [critical] present in test data point [True]
74 Text feature [including] present in test data point [True]
75 Text feature [fig] present in test data point [True]
76 Text feature [used] present in test data point [True]
77 Text feature [well] present in test data point [True]
78 Text feature [similar] present in test data point [True]
79 Text feature [previous] present in test data point [True]
80 Text feature [another] present in test data point [True]
81 Text feature [together] present in test data point [True]
82 Text feature [15] present in test data point [True]
83 Text feature [proteins] present in test data point [True]
84 Text feature [different] present in test data point [True]
85 Text feature [terminal] present in test data point [True]
86 Text feature [observed] present in test data point [True]
87 Text feature [30] present in test data point [True]
88 Text feature [discussion] present in test data point [True]
89 Text feature [mediated] present in test data point [True]
90 Text feature [remaining] present in test data point [True]
```

```
91 Text feature [thus] present in test data point [True]
92 Text feature [addition] present in test data point [True]
93 Text feature [genes] present in test data point [True]
94 Text feature [multiple] present in test data point [True]
95 Text feature [full] present in test data point [True]
96 Text feature [10] present in test data point [True]
97 Text feature [described] present in test data point [True]
98 Text feature [previously] present in test data point [True]
Out of the top  100  features  80 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [61]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3503 0.0162 0.0025 0.0786 0.0183 0.1867 0.3462 0.0007 0.0004]]
Actual Class : 6
--------------------------------------------------
9 Text feature [protein] present in test data point [True]
10 Text feature [type] present in test data point [True]
16 Text feature [dna] present in test data point [True]
17 Text feature [one] present in test data point [True]
18 Text feature [results] present in test data point [True]
19 Text feature [wild] present in test data point [True]
20 Text feature [region] present in test data point [True]
21 Text feature [two] present in test data point [True]
22 Text feature [binding] present in test data point [True]
23 Text feature [containing] present in test data point [True]
24 Text feature [functions] present in test data point [True]
25 Text feature [involved] present in test data point [True]
26 Text feature [therefore] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [using] present in test data point [True]
29 Text feature [control] present in test data point [True]
30 Text feature [role] present in test data point [True]
31 Text feature [table] present in test data point [True]
32 Text feature [expression] present in test data point [True]
33 Text feature [indicated] present in test data point [True]
34 Text feature [possible] present in test data point [True]
35 Text feature [human] present in test data point [True]
36 Text feature [reduced] present in test data point [True]
37 Text feature [either] present in test data point [True]
39 Text feature [determined] present in test data point [True]
40 Text feature [specific] present in test data point [True]
41 Text feature [sequences] present in test data point [True]
43 Text feature [gene] present in test data point [True]
44 Text feature [affect] present in test data point [True]
45 Text feature [following] present in test data point [True]
46 Text feature [four] present in test data point [True]
47 Text feature [three] present in test data point [True]
48 Text feature [function] present in test data point [True]
49 Text feature [however] present in test data point [True]
50 Text feature [shown] present in test data point [True]
51 Text feature [loss] present in test data point [True]
53 Text feature [effect] present in test data point [True]
55 Text feature [present] present in test data point [True]
57 Text feature [ability] present in test data point [True]
58 Text feature [result] present in test data point [True]
59 Text feature [important] present in test data point [True]
60 Text feature [several] present in test data point [True]
61 Text feature [suggest] present in test data point [True]
64 Text feature [analysis] present in test data point [True]
67 Text feature [corresponding] present in test data point [True]
68 Text feature [transcription] present in test data point [True]
69 Text feature [may] present in test data point [True]
70 Text feature [respectively] present in test data point [True]
71 Text feature [length] present in test data point [True]
72 Text feature [performed] present in test data point [True]
73 Text feature [critical] present in test data point [True]
74 Text feature [including] present in test data point [True]
75 Text feature [fig] present in test data point [True]
76 Text feature [used] present in test data point [True]
77 Text feature [well] present in test data point [True]
78 Text feature [similar] present in test data point [True]
79 Text feature [previous] present in test data point [True]
80 Text feature [another] present in test data point [True]
81 Text feature [together] present in test data point [True]
82 Text feature [15] present in test data point [True]
83 Text feature [proteins] present in test data point [True]
84 Text feature [different] present in test data point [True]
85 Text feature [terminal] present in test data point [True]
86 Text feature [observed] present in test data point [True]
87 Text feature [30] present in test data point [True]
88 Text feature [discussion] present in test data point [True]
89 Text feature [mediated] present in test data point [True]
91 Text feature [thus] present in test data point [True]
92 Text feature [addition] present in test data point [True]
93 Text feature [genes] present in test data point [True]
94 Text feature [multiple] present in test data point [True]
95 Text feature [full] present in test data point [True]
96 Text feature [10] present in test data point [True]
```

```
97 Text feature [described] present in test data point [True]
98 Text feature [previously] present in test data point [True]
99 Text feature [furthermore] present in test data point [True]
Out of the top  100  features  76 are present in query point
```

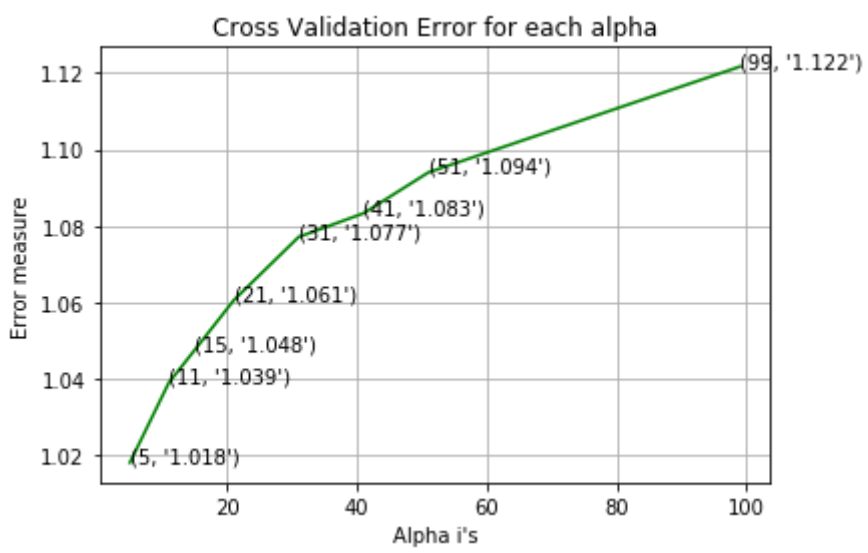
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [62]:

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

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


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for alpha = 5
Log Loss : 1.018188204702352
for alpha = 11
Log Loss : 1.0390696264776218
for alpha = 15
Log Loss : 1.0476352290571909
for alpha = 21
Log Loss : 1.0605937382961506
for alpha = 31
Log Loss : 1.0771391456776076
for alpha = 41
Log Loss : 1.0833274114220066
for alpha = 51
Log Loss : 1.0940162465861756
for alpha = 99
Log Loss : 1.1216539446538119
```

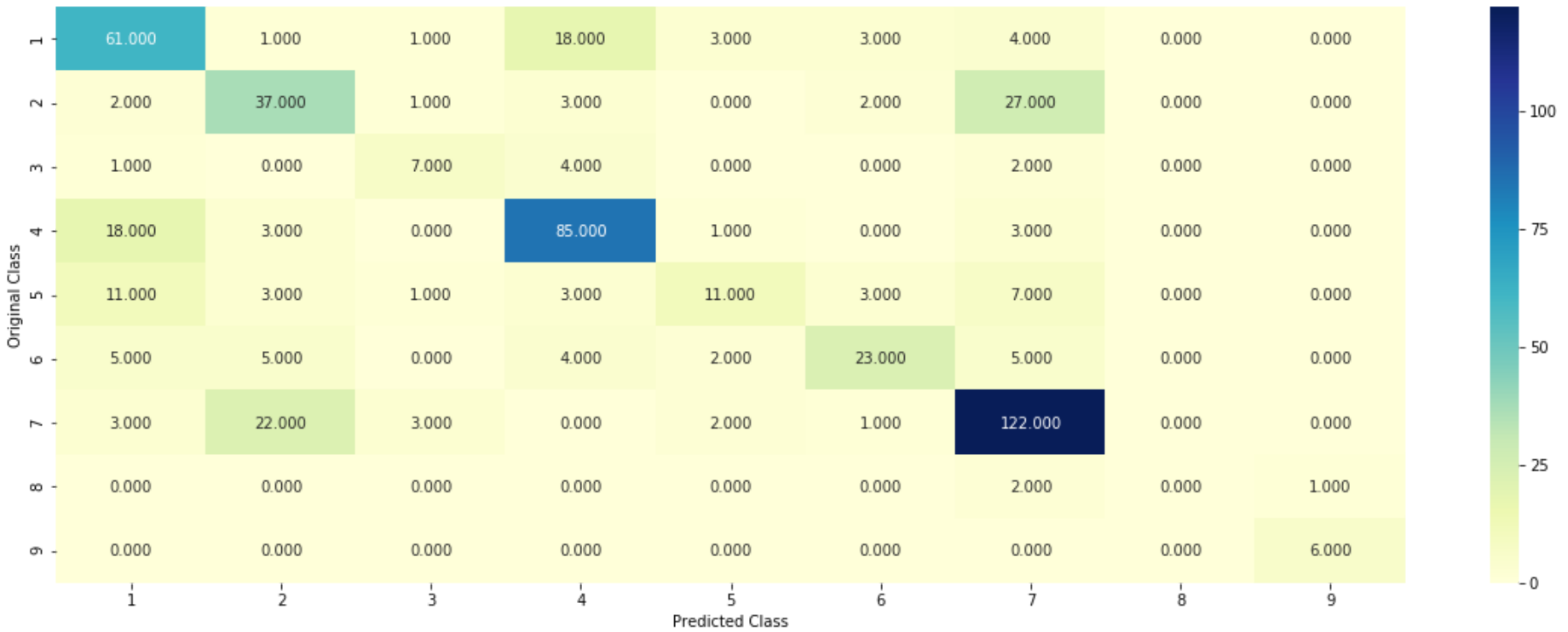



```
For values of best alpha =  5 The train log loss is: 0.49230858704152763
For values of best alpha =  5 The cross validation log loss is: 1.018188204702352
For values of best alpha =  5 The test log loss is: 1.0475598954018752
```

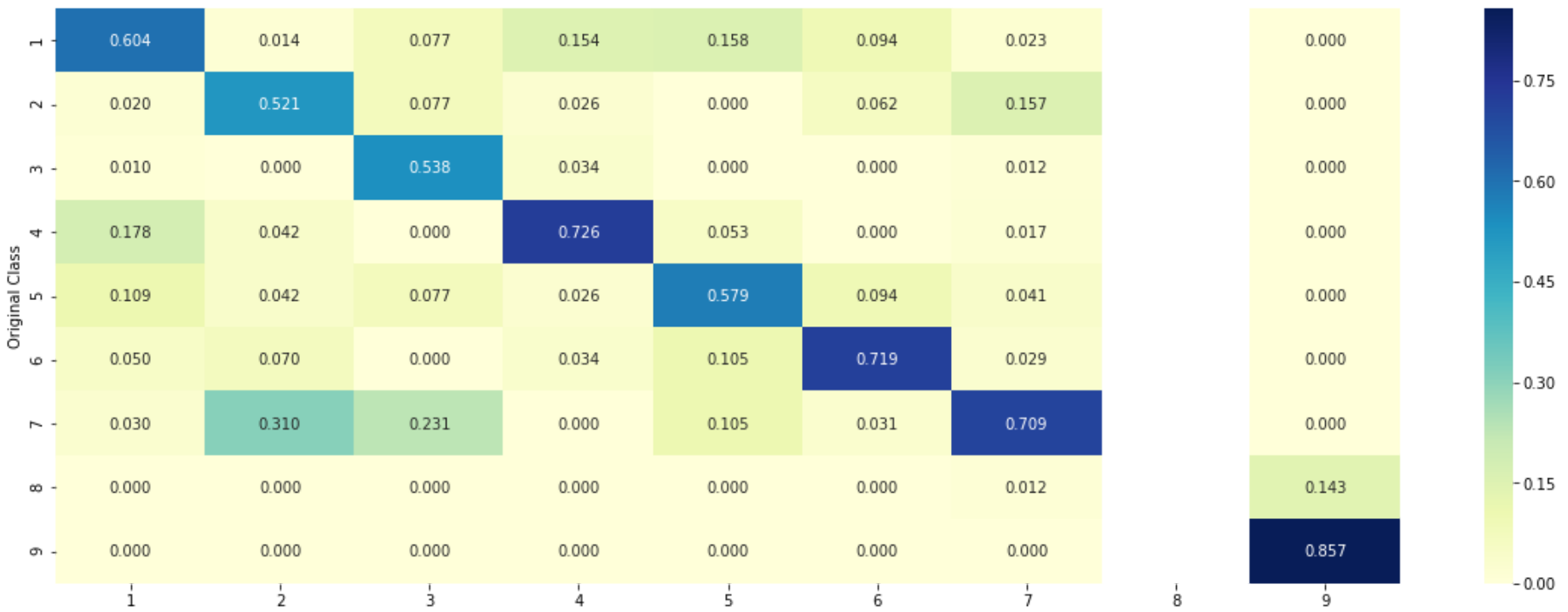
4.2.2. Testing the model with best hyper paramters

In [63]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

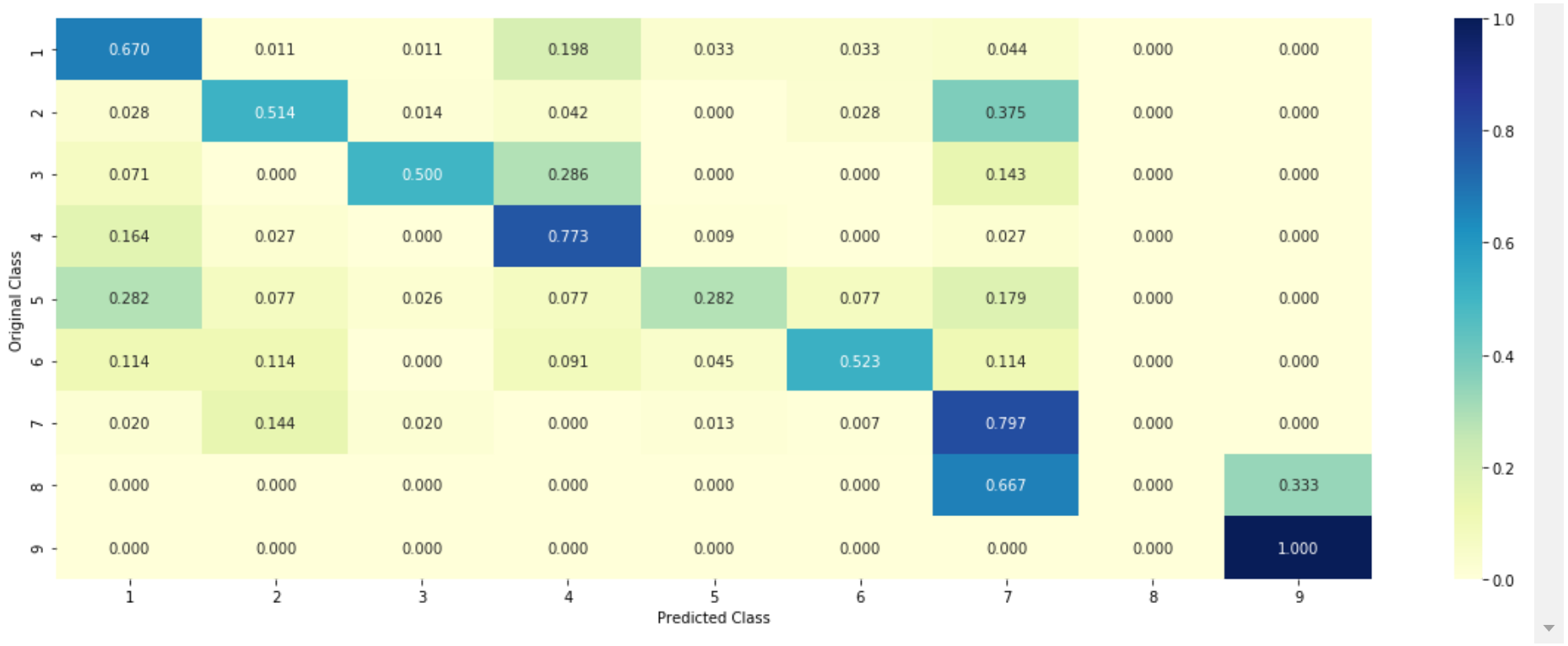
```
Log loss : 1.018188204702352
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [64]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

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

4.2.4. Sample Query Point-2

In [65]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

```
Predicted Class : 6
Actual Class : 6
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [1 1 6 6 6]
Fequency of nearest points : Counter({6: 3, 1: 2})
```

4.3. Logistic Regression

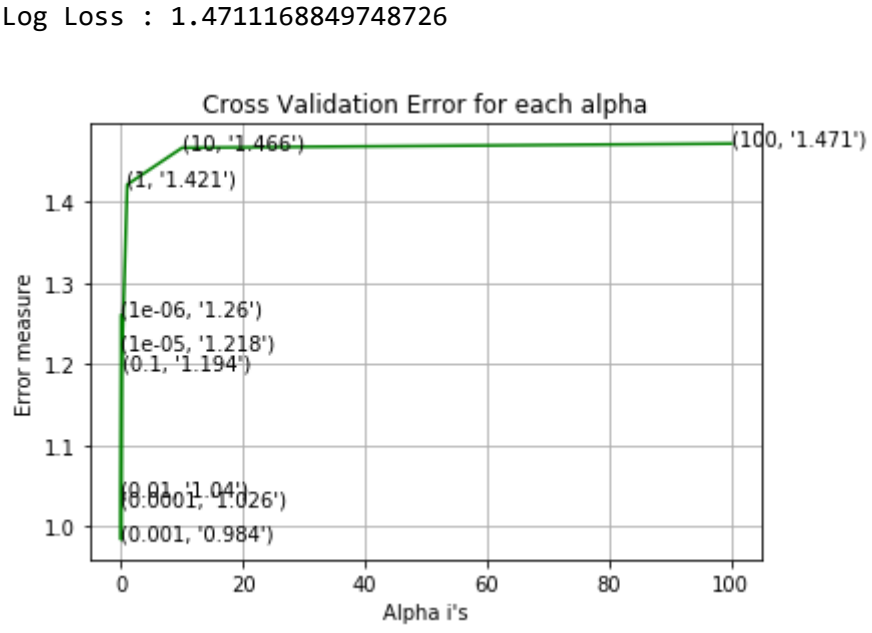
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [66]:

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for alpha = 1e-06
Log Loss : 1.2596262106861196
for alpha = 1e-05
Log Loss : 1.2175960425138106
for alpha = 0.0001
Log Loss : 1.0257875911332606
for alpha = 0.001
Log Loss : 0.9840241874578949
for alpha = 0.01
Log Loss : 1.0399521811072645
for alpha = 0.1
Log Loss : 1.1941744296411803
for alpha = 1
Log Loss : 1.4207233973001
for alpha = 10
Log Loss : 1.4659939646841194
for alpha = 100
Log Loss : 1.4711168849748726
```



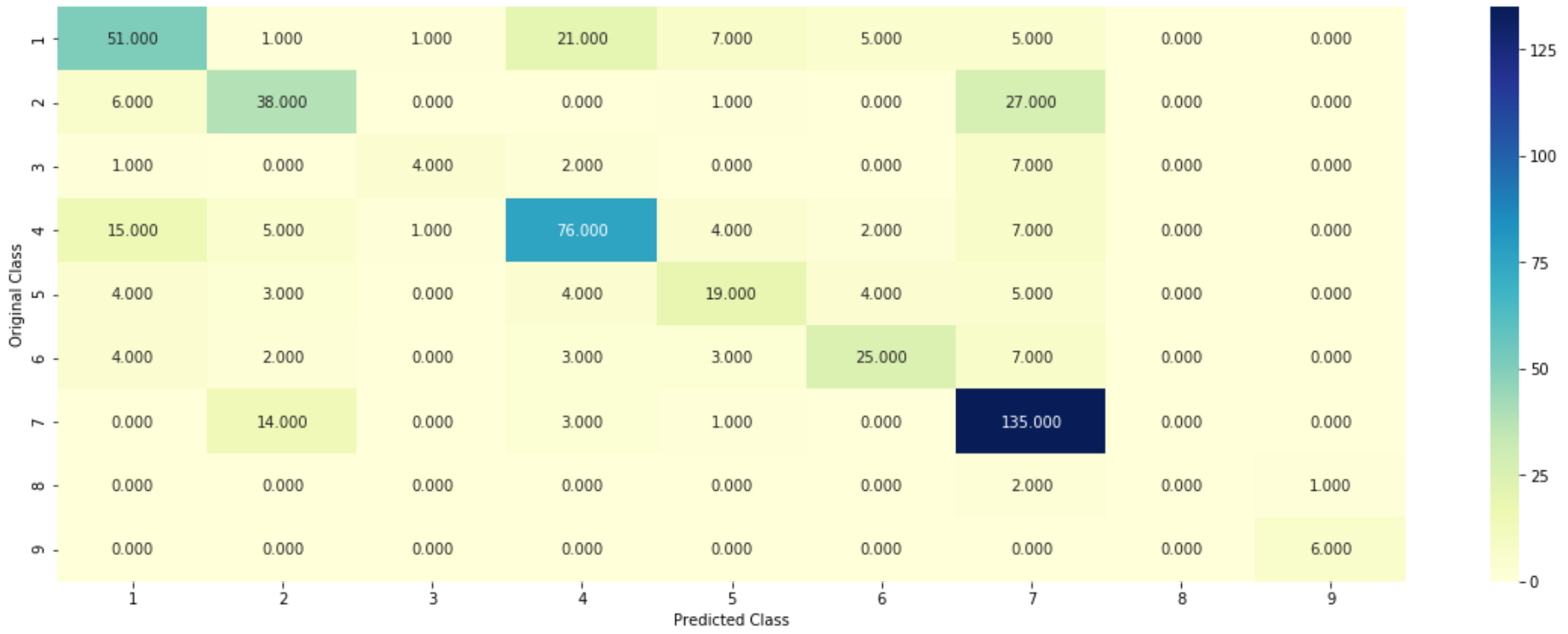
```
For values of best alpha =  0.001 The train log loss is: 0.5587460529759314
For values of best alpha =  0.001 The cross validation log loss is: 0.9840241874578949
For values of best alpha =  0.001 The test log loss is: 1.0499513044204818
```

4.3.1.2. Testing the model with best hyper paramters

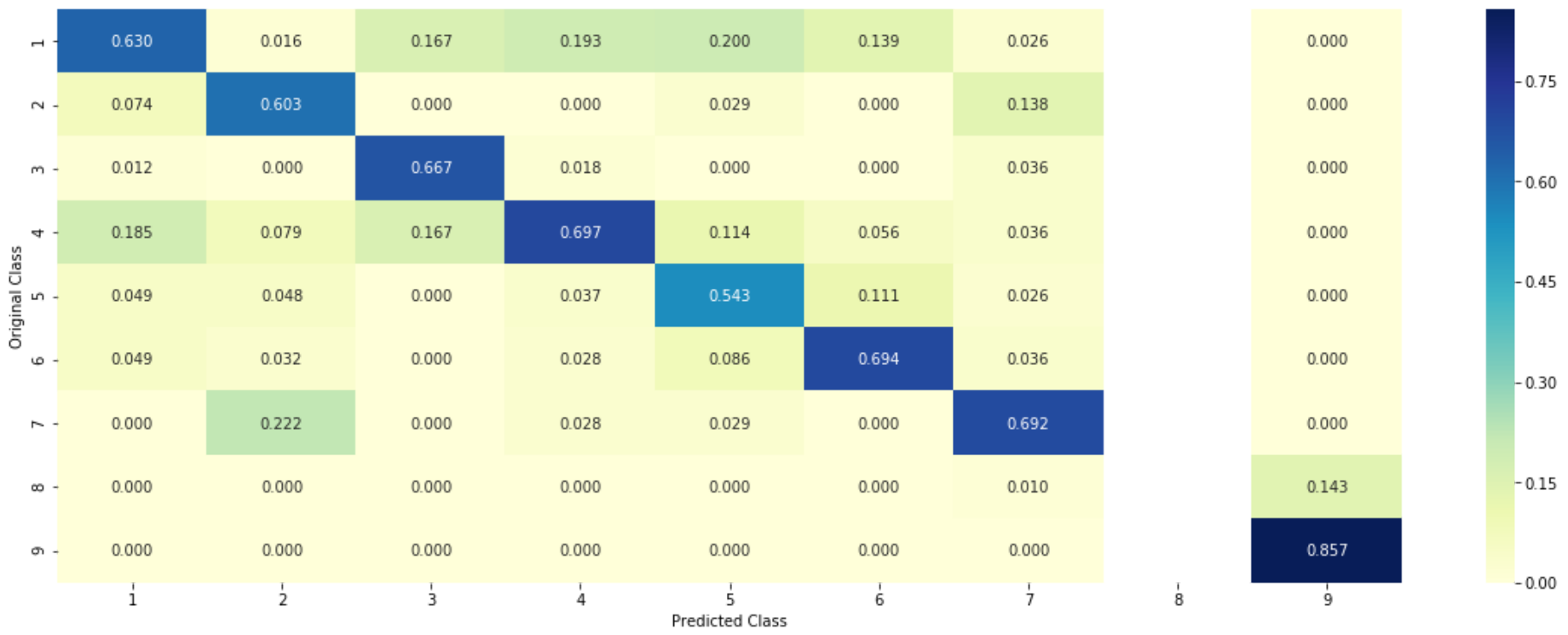
In [67]:

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

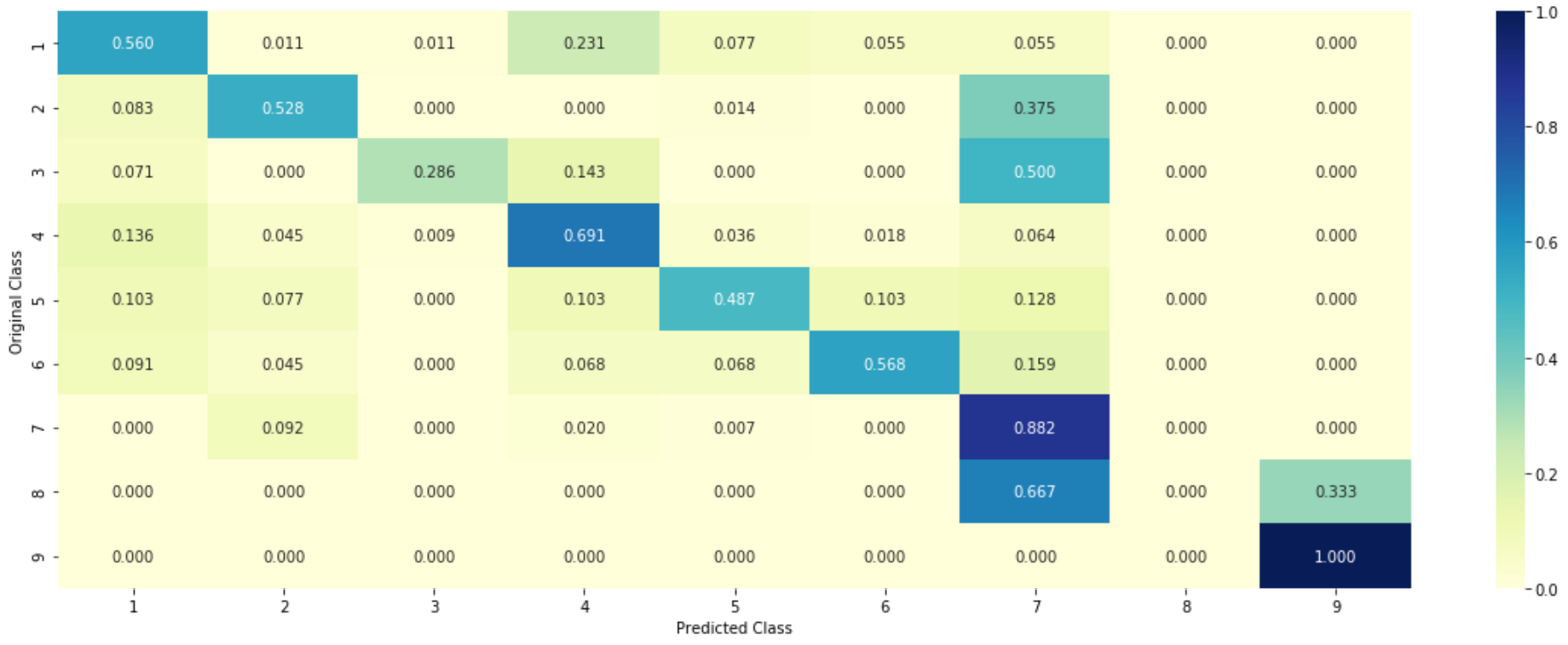
```
Log loss : 0.9840241874578949
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Feature Importance

In [68]:
```
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 [71]:
```
# 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 = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.017  0.038  0.006  0.0261 0.021  0.0137 0.869  0.0047 0.0044]]
Actual Class : 7
--------------------------------------------------
20 Text feature [constitutively] present in test data point [True]
22 Text feature [activated] present in test data point [True]
28 Text feature [constitutive] present in test data point [True]
38 Text feature [jaks] present in test data point [True]
41 Text feature [receptors] present in test data point [True]
73 Text feature [proliferate] present in test data point [True]
79 Text feature [stat] present in test data point [True]
105 Text feature [expressing] present in test data point [True]
122 Text feature [mapk] present in test data point [True]
132 Text feature [technology] present in test data point [True]
134 Text feature [doses] present in test data point [True]
135 Text feature [activation] present in test data point [True]
141 Text feature [murine] present in test data point [True]
202 Text feature [jak] present in test data point [True]
229 Text feature [phospho] present in test data point [True]
264 Text feature [serum] present in test data point [True]
272 Text feature [janus] present in test data point [True]
291 Text feature [transducer] present in test data point [True]
310 Text feature [kinase] present in test data point [True]
318 Text feature [inhibitor] present in test data point [True]
353 Text feature [activating] present in test data point [True]
370 Text feature [cytokine] present in test data point [True]
377 Text feature [signaling] present in test data point [True]
404 Text feature [sterically] present in test data point [True]
483 Text feature [stat5] present in test data point [True]
489 Text feature [jak1] present in test data point [True]
Out of the top  500  features  26 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [75]:
```
test_point_index = 98
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.4359 0.0105 0.0017 0.0069 0.4924 0.0396 0.006  0.0043 0.0027]]
Actual Class : 1
--------------------------------------------------
345 Text feature [substituents] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

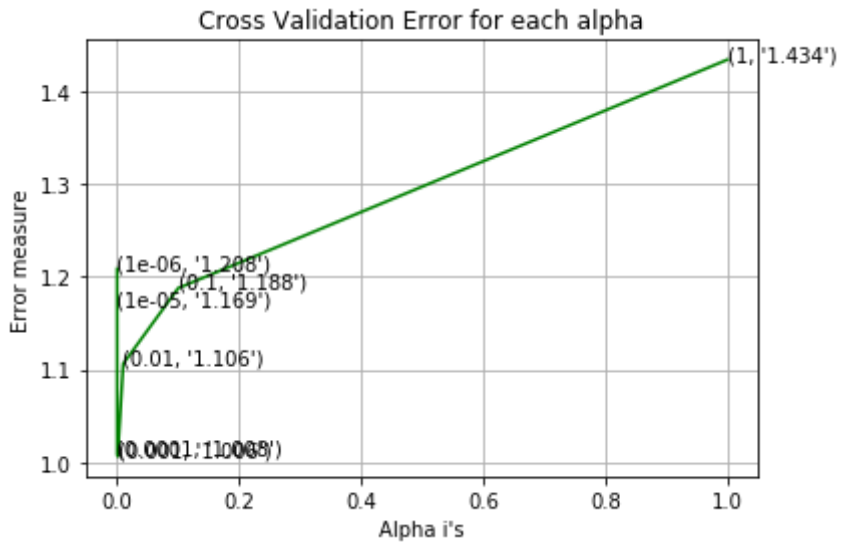
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [76]:

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for alpha = 1e-06
Log Loss : 1.208403735344943
for alpha = 1e-05
Log Loss : 1.1685958606792166
for alpha = 0.0001
Log Loss : 1.0083158052945014
for alpha = 0.001
Log Loss : 1.0060257927876197
for alpha = 0.01
Log Loss : 1.1060121001654075
for alpha = 0.1
Log Loss : 1.1875613964831362
for alpha = 1
Log Loss : 1.4339956465768093
```



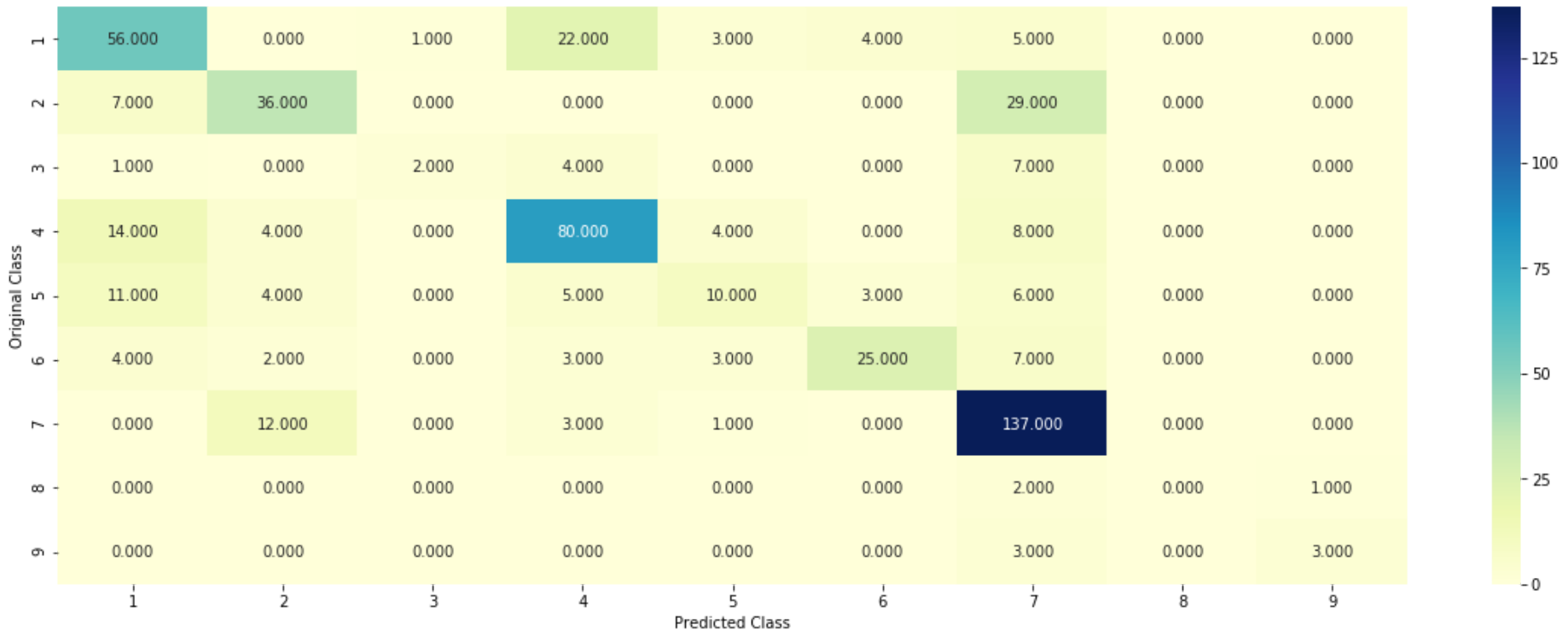

```
For values of best alpha =  0.001 The train log loss is: 0.547207389864498
For values of best alpha =  0.001 The cross validation log loss is: 1.0060257927876197
For values of best alpha =  0.001 The test log loss is: 1.0608145256104677
```

4.3.2.2. Testing model with best hyper parameters

In [77]:

```
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 : 1.0060257927876197
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```




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


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

4.3.2.3. Feature Importance, Correctly Classified point

In [78]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 6
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[4.000e-04 1.230e-02 0.000e+00 4.000e-04 5.000e-04 1.000e-04 9.863e-01
  0.000e+00 0.000e+00]]
Actual Class : 7
--------------------------------------------------
42 Text feature [constitutively] present in test data point [True]
63 Text feature [activated] present in test data point [True]
71 Text feature [constitutive] present in test data point [True]
77 Text feature [ligand] present in test data point [True]
90 Text feature [nude] present in test data point [True]
97 Text feature [receptors] present in test data point [True]
98 Text feature [downstream] present in test data point [True]
102 Text feature [expressing] present in test data point [True]
103 Text feature [oncoproteins] present in test data point [True]
105 Text feature [grew] present in test data point [True]
109 Text feature [mitogen] present in test data point [True]
110 Text feature [agar] present in test data point [True]
123 Text feature [renewed] present in test data point [True]
129 Text feature [proliferate] present in test data point [True]
144 Text feature [rauen] present in test data point [True]
150 Text feature [activation] present in test data point [True]
151 Text feature [oncogenes] present in test data point [True]
157 Text feature [oncogene] present in test data point [True]
165 Text feature [doses] present in test data point [True]
170 Text feature [subcutaneous] present in test data point [True]
175 Text feature [mapk] present in test data point [True]
179 Text feature [overexpression] present in test data point [True]
182 Text feature [transform] present in test data point [True]
194 Text feature [antagonist] present in test data point [True]
200 Text feature [technology] present in test data point [True]
203 Text feature [kinase] present in test data point [True]
204 Text feature [murine] present in test data point [True]
208 Text feature [nf] present in test data point [True]
209 Text feature [allosterically] present in test data point [True]
232 Text feature [transducer] present in test data point [True]
242 Text feature [signaling] present in test data point [True]
244 Text feature [phospho] present in test data point [True]
249 Text feature [inhibitor] present in test data point [True]
263 Text feature [activating] present in test data point [True]
281 Text feature [anchorage] present in test data point [True]
283 Text feature [definitively] present in test data point [True]
293 Text feature [serum] present in test data point [True]
296 Text feature [layer] present in test data point [True]
299 Text feature [frederick] present in test data point [True]
310 Text feature [hrasg12v] present in test data point [True]
318 Text feature [tk] present in test data point [True]
326 Text feature [farnesyl] present in test data point [True]
327 Text feature [interleukin] present in test data point [True]
331 Text feature [therapeutics] present in test data point [True]
349 Text feature [transforming] present in test data point [True]
355 Text feature [pi3] present in test data point [True]
365 Text feature [akt] present in test data point [True]
398 Text feature [transformation] present in test data point [True]
415 Text feature [activate] present in test data point [True]
418 Text feature [cultured] present in test data point [True]
421 Text feature [nanomolar] present in test data point [True]
426 Text feature [enhanced] present in test data point [True]
434 Text feature [metastasis] present in test data point [True]
449 Text feature [slower] present in test data point [True]
461 Text feature [phosphorylation] present in test data point [True]
470 Text feature [basement] present in test data point [True]
474 Text feature [concentrations] present in test data point [True]
479 Text feature [dmba] present in test data point [True]
485 Text feature [rbd] present in test data point [True]
486 Text feature [lieu] present in test data point [True]
494 Text feature [erk] present in test data point [True]
Out of the top  500  features  61 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [81]:
```
test_point_index = 98
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.3288 0.0084 0.0095 0.0172 0.6062 0.0035 0.0144 0.0052 0.0068]]
Actual Class : 1
--------------------------------------------------
404 Text feature [asparagine] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

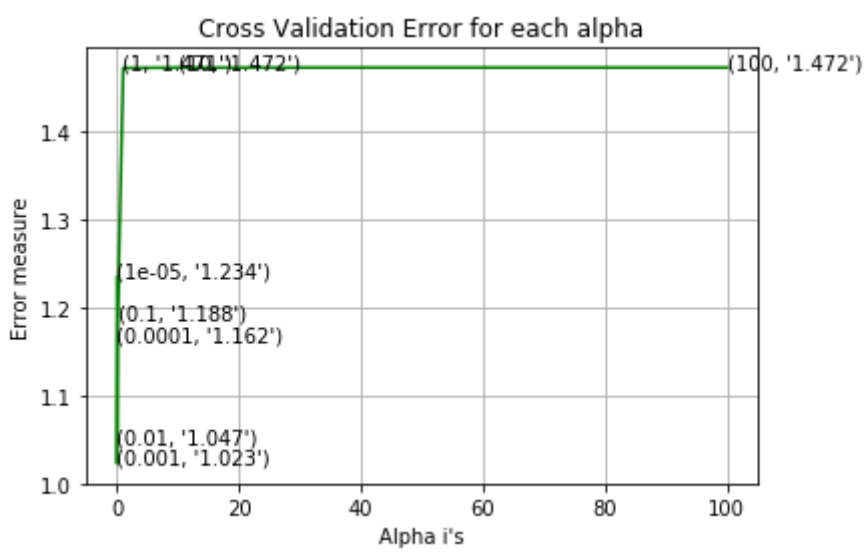
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [82]:

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for C = 1e-05
Log Loss : 1.2343352357716517
for C = 0.0001
Log Loss : 1.1616101701639137
for C = 0.001
Log Loss : 1.0230170111361154
for C = 0.01
Log Loss : 1.0468243802332045
for C = 0.1
Log Loss : 1.1882235047033607
for C = 1
Log Loss : 1.4714773156812868
for C = 10
Log Loss : 1.4719871861571372
for C = 100
Log Loss : 1.4719811990711835
```



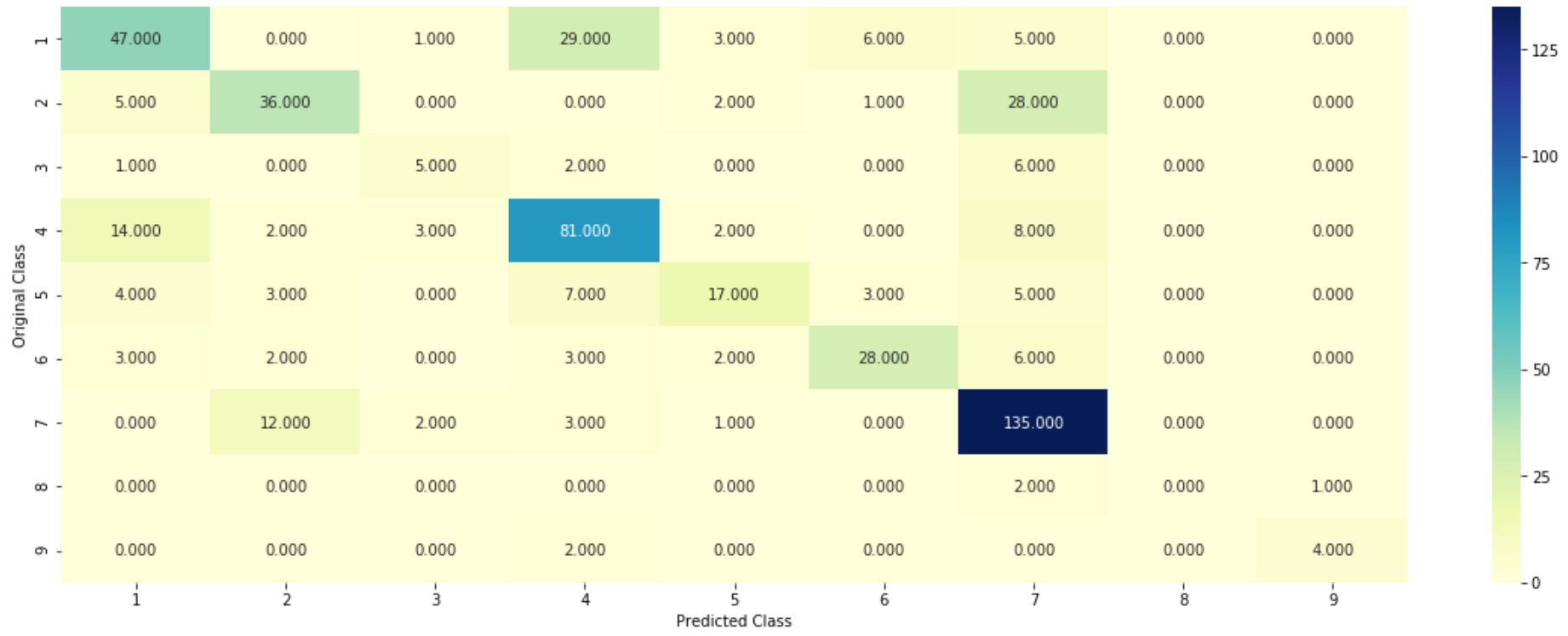

```
For values of best alpha =  0.001 The train log loss is: 0.6019953987183904
For values of best alpha =  0.001 The cross validation log loss is: 1.0230170111361154
For values of best alpha =  0.001 The test log loss is: 1.113708763407425
```

4.4.2. Testing model with best hyper parameters

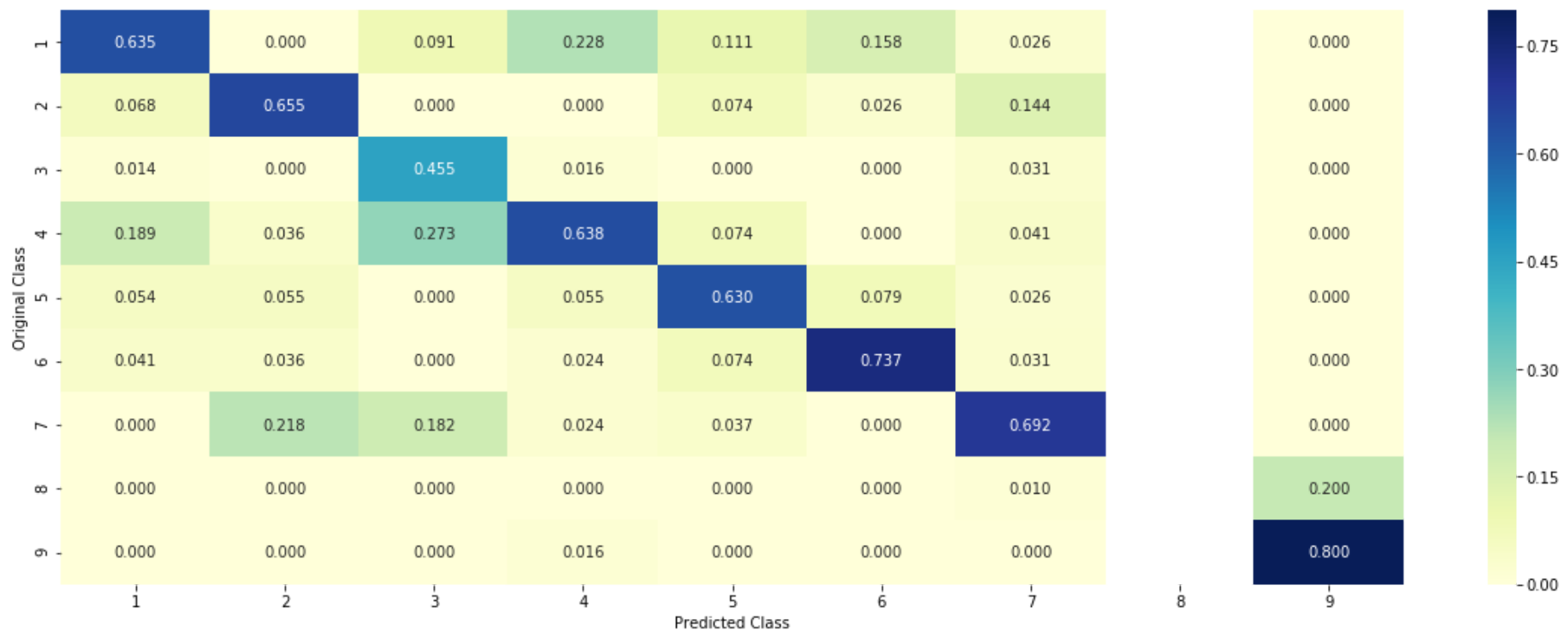
In [83]:

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

```
Log loss : 1.0230170111361154
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```




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




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

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [84]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0429 0.0461 0.0103 0.0496 0.0342 0.015  0.7918 0.0041 0.0059]]
Actual Class : 7
--------------------------------------------------
161 Text feature [constitutively] present in test data point [True]
205 Text feature [activated] present in test data point [True]
230 Text feature [constitutive] present in test data point [True]
262 Text feature [expressing] present in test data point [True]
281 Text feature [jaks] present in test data point [True]
286 Text feature [stat] present in test data point [True]
353 Text feature [receptors] present in test data point [True]
361 Text feature [doses] present in test data point [True]
366 Text feature [proliferate] present in test data point [True]
368 Text feature [transformation] present in test data point [True]
376 Text feature [murine] present in test data point [True]
402 Text feature [mapk] present in test data point [True]
408 Text feature [transducer] present in test data point [True]
487 Text feature [technology] present in test data point [True]
488 Text feature [ectopically] present in test data point [True]
490 Text feature [signaling] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [86]:
```
test_point_index = 98
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.3288 0.0084 0.0095 0.0172 0.6062 0.0035 0.0144 0.0052 0.0068]]
Actual Class : 1
--------------------------------------------------
362 Text feature [asparagine] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [87]:

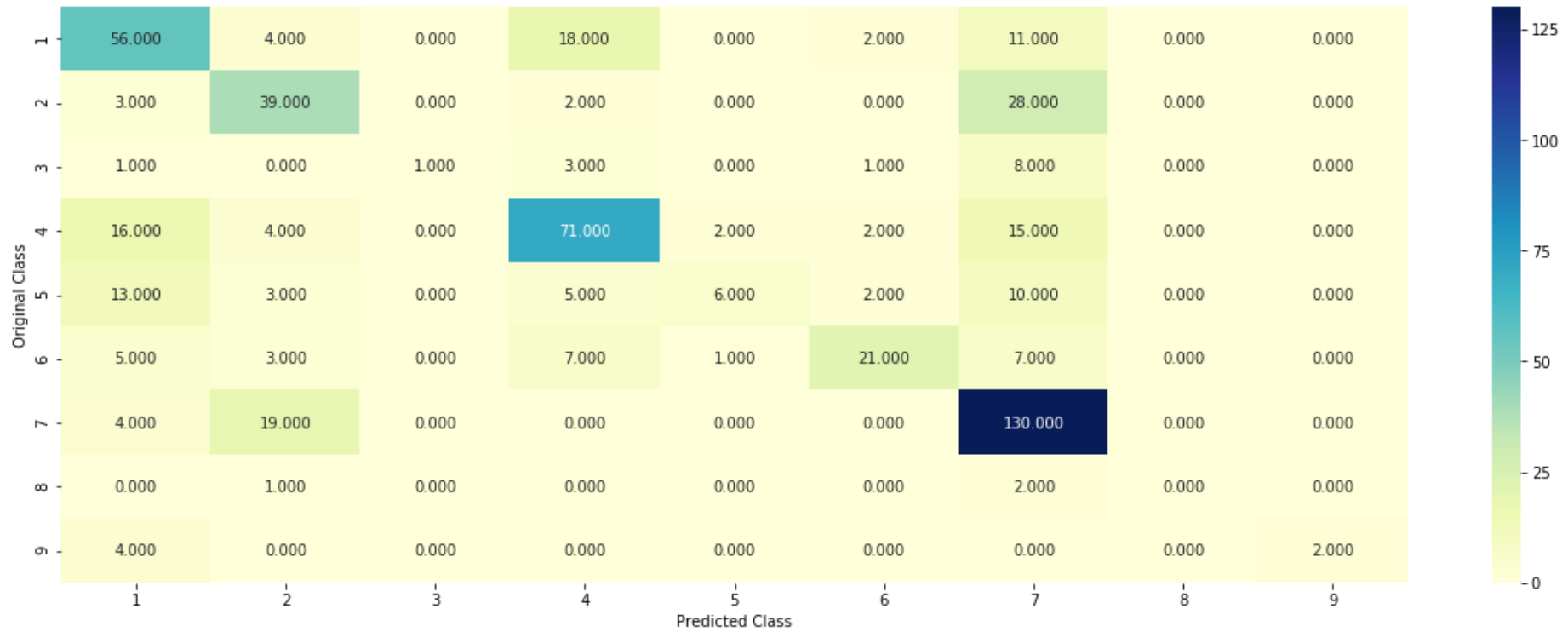
```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
       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

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train, predict_y,
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, pr
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(y_test, predict_y, l
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2661051381124329
for n_estimators = 100 and max depth =  10
Log Loss : 1.2074028513928254
for n_estimators = 200 and max depth =  5
Log Loss : 1.2549426731766813
for n_estimators = 200 and max depth =  10
Log Loss : 1.1985686885397313
for n_estimators = 500 and max depth =  5
Log Loss : 1.2534618179806944
for n_estimators = 500 and max depth =  10
Log Loss : 1.1861516753586883
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2507598907933966
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1837526224934958
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2515511578760292
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1841683117989419
For values of best estimator =  1000 The train log loss is: 0.6784700913468277
For values of best estimator =  1000 The cross validation log loss is: 1.1837526224934958
For values of best estimator =  1000 The test log loss is: 1.1669973560529792
```

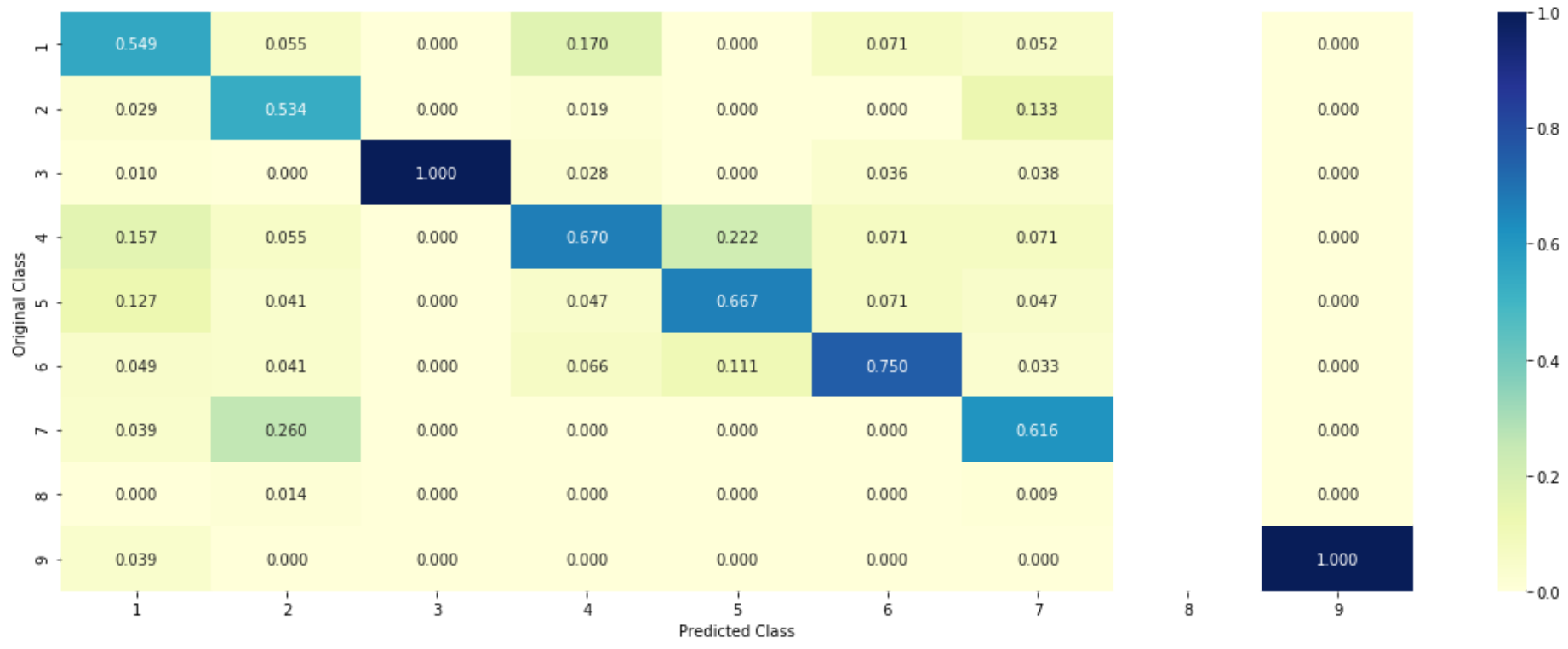
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [88]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1837526224934958
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```




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




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

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [89]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5358 0.0141 0.0168 0.0519 0.3055 0.0535 0.0148 0.004  0.0037]]
Actual Class : 1
--------------------------------------------------
5 Text feature [activation] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
15 Text feature [missense] present in test data point [True]
16 Text feature [function] present in test data point [True]
18 Text feature [loss] present in test data point [True]
19 Text feature [downstream] present in test data point [True]
23 Text feature [protein] present in test data point [True]
24 Text feature [yeast] present in test data point [True]
27 Text feature [signaling] present in test data point [True]
29 Text feature [stability] present in test data point [True]
32 Text feature [deleterious] present in test data point [True]
33 Text feature [clinical] present in test data point [True]
34 Text feature [functional] present in test data point [True]
37 Text feature [pathogenic] present in test data point [True]
39 Text feature [cells] present in test data point [True]
47 Text feature [cell] present in test data point [True]
50 Text feature [patients] present in test data point [True]
53 Text feature [brca1] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
56 Text feature [potential] present in test data point [True]
62 Text feature [phosphorylated] present in test data point [True]
65 Text feature [classified] present in test data point [True]
66 Text feature [brct] present in test data point [True]
67 Text feature [defective] present in test data point [True]
70 Text feature [terminal] present in test data point [True]
73 Text feature [neutral] present in test data point [True]
74 Text feature [brca2] present in test data point [True]
76 Text feature [predicted] present in test data point [True]
78 Text feature [pathogenicity] present in test data point [True]
79 Text feature [p53] present in test data point [True]
80 Text feature [unstable] present in test data point [True]
82 Text feature [repair] present in test data point [True]
94 Text feature [variants] present in test data point [True]
97 Text feature [response] present in test data point [True]
Out of the top  100  features  35 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [90]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index]
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3084 0.0863 0.0274 0.1054 0.0665 0.2402 0.152  0.0067 0.0071]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitor] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [activated] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
9 Text feature [suppressor] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [constitutively] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
14 Text feature [nonsense] present in test data point [True]
15 Text feature [missense] present in test data point [True]
16 Text feature [function] present in test data point [True]
17 Text feature [activate] present in test data point [True]
18 Text feature [loss] present in test data point [True]
23 Text feature [protein] present in test data point [True]
26 Text feature [receptor] present in test data point [True]
27 Text feature [signaling] present in test data point [True]
29 Text feature [stability] present in test data point [True]
31 Text feature [growth] present in test data point [True]
34 Text feature [functional] present in test data point [True]
35 Text feature [inhibited] present in test data point [True]
36 Text feature [survival] present in test data point [True]
37 Text feature [pathogenic] present in test data point [True]
38 Text feature [kinases] present in test data point [True]
39 Text feature [cells] present in test data point [True]
41 Text feature [expressing] present in test data point [True]
42 Text feature [extracellular] present in test data point [True]
46 Text feature [inhibition] present in test data point [True]
47 Text feature [cell] present in test data point [True]
48 Text feature [mitogen] present in test data point [True]
49 Text feature [proliferation] present in test data point [True]
50 Text feature [patients] present in test data point [True]
51 Text feature [phospho] present in test data point [True]
52 Text feature [active] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
57 Text feature [serum] present in test data point [True]
58 Text feature [harboring] present in test data point [True]
62 Text feature [phosphorylated] present in test data point [True]
69 Text feature [metastatic] present in test data point [True]
70 Text feature [terminal] present in test data point [True]
76 Text feature [predicted] present in test data point [True]
83 Text feature [ligand] present in test data point [True]
86 Text feature [autophosphorylation] present in test data point [True]
90 Text feature [soft] present in test data point [True]
93 Text feature [functions] present in test data point [True]
94 Text feature [variants] present in test data point [True]
97 Text feature [response] present in test data point [True]
98 Text feature [factor] present in test data point [True]
99 Text feature [families] present in test data point [True]
Out of the top  100  features  52 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [91]:
```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha
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, lab
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, predic
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_test, predict_y, label
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0905612101237026
for n_estimators = 10 and max depth =  3
Log Loss : 1.5737321919848857
for n_estimators = 10 and max depth =  5
Log Loss : 1.746112533979271
for n_estimators = 10 and max depth =  10
Log Loss : 1.639200638202826
for n_estimators = 50 and max depth =  2
Log Loss : 1.763206452736626
for n_estimators = 50 and max depth =  3
Log Loss : 1.4986211006578782
for n_estimators = 50 and max depth =  5
Log Loss : 1.5310851900387814
for n_estimators = 50 and max depth =  10
Log Loss : 1.6432835724214305
for n_estimators = 100 and max depth =  2
Log Loss : 1.6454491237928843
for n_estimators = 100 and max depth =  3
Log Loss : 1.5258648117744968
for n_estimators = 100 and max depth =  5
Log Loss : 1.4716415375589944
for n_estimators = 100 and max depth =  10
Log Loss : 1.675139165629154
for n_estimators = 200 and max depth =  2
Log Loss : 1.7235153341852172
for n_estimators = 200 and max depth =  3
Log Loss : 1.60824549568261
for n_estimators = 200 and max depth =  5
Log Loss : 1.4411686111895414
for n_estimators = 200 and max depth =  10
Log Loss : 1.689651879445109
for n_estimators = 500 and max depth =  2
Log Loss : 1.753187269330325
for n_estimators = 500 and max depth =  3
Log Loss : 1.6368046410033554
for n_estimators = 500 and max depth =  5
Log Loss : 1.4169628772612128
for n_estimators = 500 and max depth =  10
Log Loss : 1.7132748891590908
for n_estimators = 1000 and max depth =  2
Log Loss : 1.748868541190876
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6166174234796715
for n_estimators = 1000 and max depth =  5
Log Loss : 1.407575385361305
for n_estimators = 1000 and max depth =  10
```

```
Log Loss : 1.6945463207603786
For values of best alpha =  1000 The train log loss is: 0.05266733231671452
For values of best alpha =  1000 The cross validation log loss is: 1.407575385361305
For values of best alpha =  1000 The test log loss is: 1.3875976202293747
```

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

In [92]:
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], criterion='g
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.407575385361305
Number of mis-classified points : 0.5338345864661654
-------------------- Confusion matrix --------------------
```


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


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

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [96]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 3
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0155 0.7387 0.0322 0.0178 0.0145 0.0329 0.0378 0.0808 0.0298]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene 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
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [98]:
```
test_point_index = 98
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1
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 : 6
Predicted Class Probabilities: [[0.0697 0.0257 0.0499 0.0475 0.144  0.4197 0.0132 0.1302 0.1001]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene 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
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

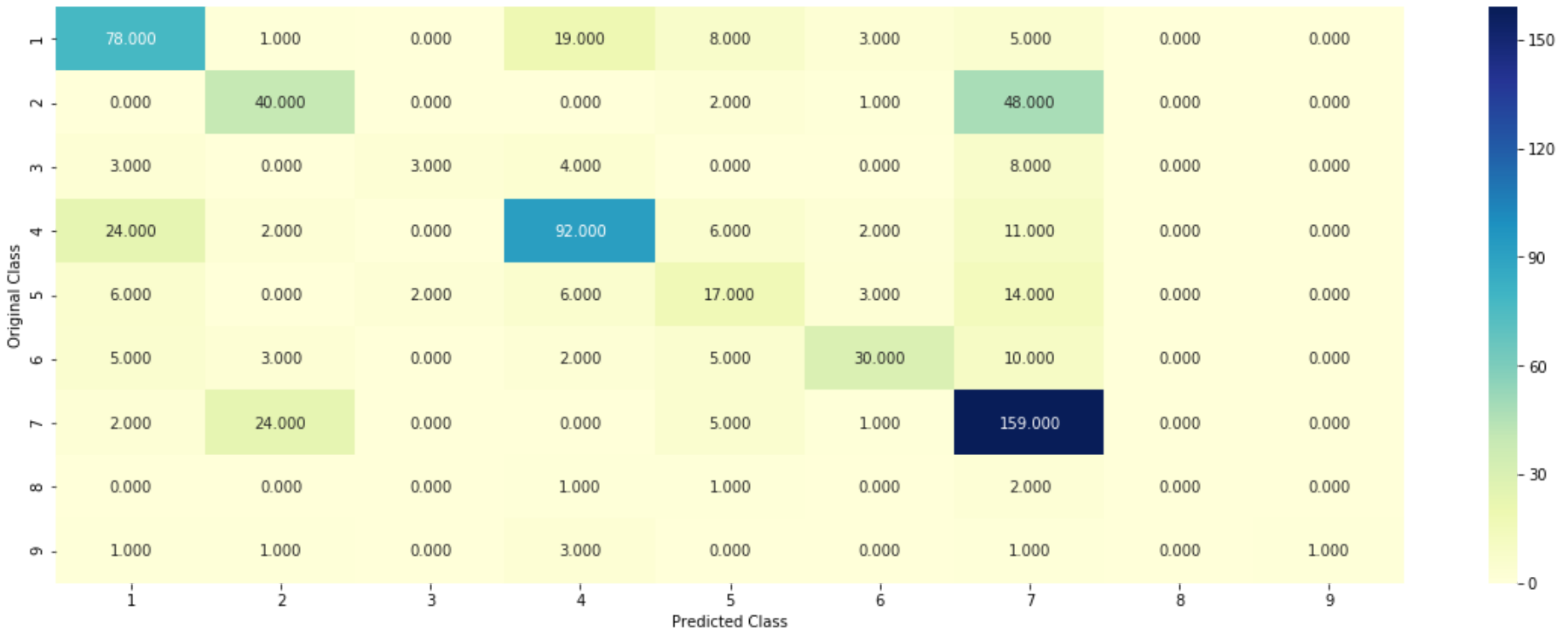
In [99]:
```
from tqdm import tqdm
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in tqdm(alpha):
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 0.99
Support vector machines : Log Loss: 1.47
Naive Bayes : Log Loss: 1.23
--------------------------------------------------
  0%|                                                                                                | 0/6 [00:00<?, ?it/s]
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
 17%|████████████                                                                            | 1/6 [00:07<00:37,  7.43s/it]
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
 33%|████████████████████████                                                                | 2/6 [00:14<00:29,  7.44s/it]
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.488
 50%|████████████████████████████████████                                                    | 3/6 [00:22<00:22,  7.44s/it]
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.077
 67%|████████████████████████████████████████████████                                        | 4/6 [00:29<00:14,  7.45s/it]
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.130
 83%|████████████████████████████████████████████████████████████                            | 5/6 [00:37<00:07,  7.45s/it]
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.338
100%|████████████████████████████████████████████████████████████████████████████████████████| 6/6 [00:44<00:00,  7.46s/it]
```

4.7.2 testing the model with the best hyper parameters

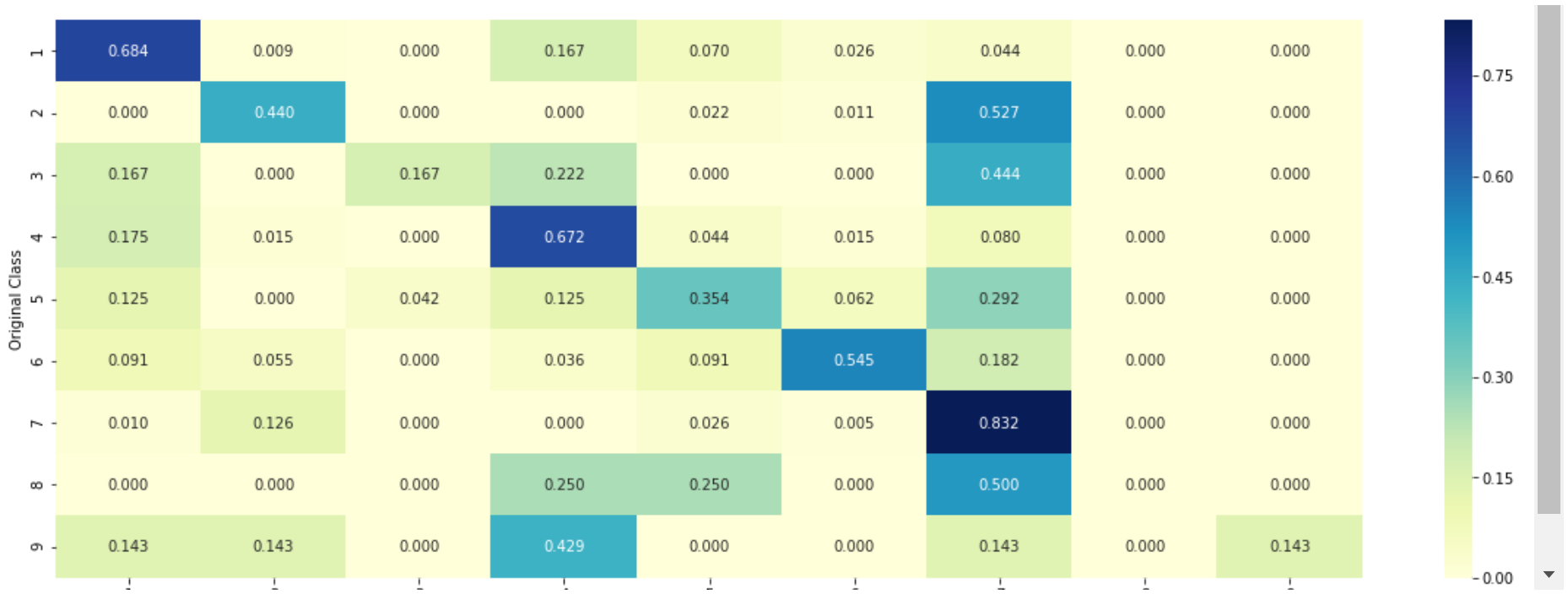
In [100]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.6457196304998355
Log loss (CV) on the stacking classifier : 1.0774203254087935
Log loss (test) on the stacking classifier : 1.109080234950771
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```




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

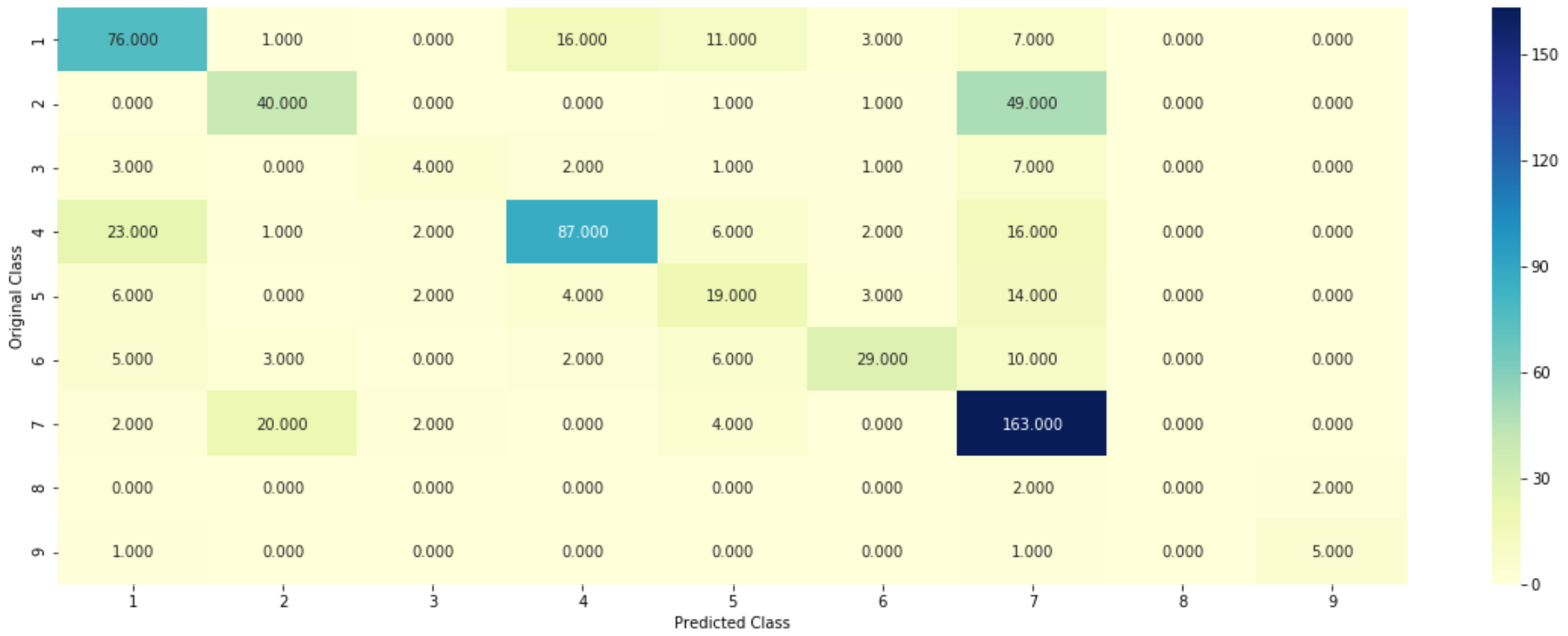

```
-------------------- Recall matrix (Row sum=1) --------------------
```



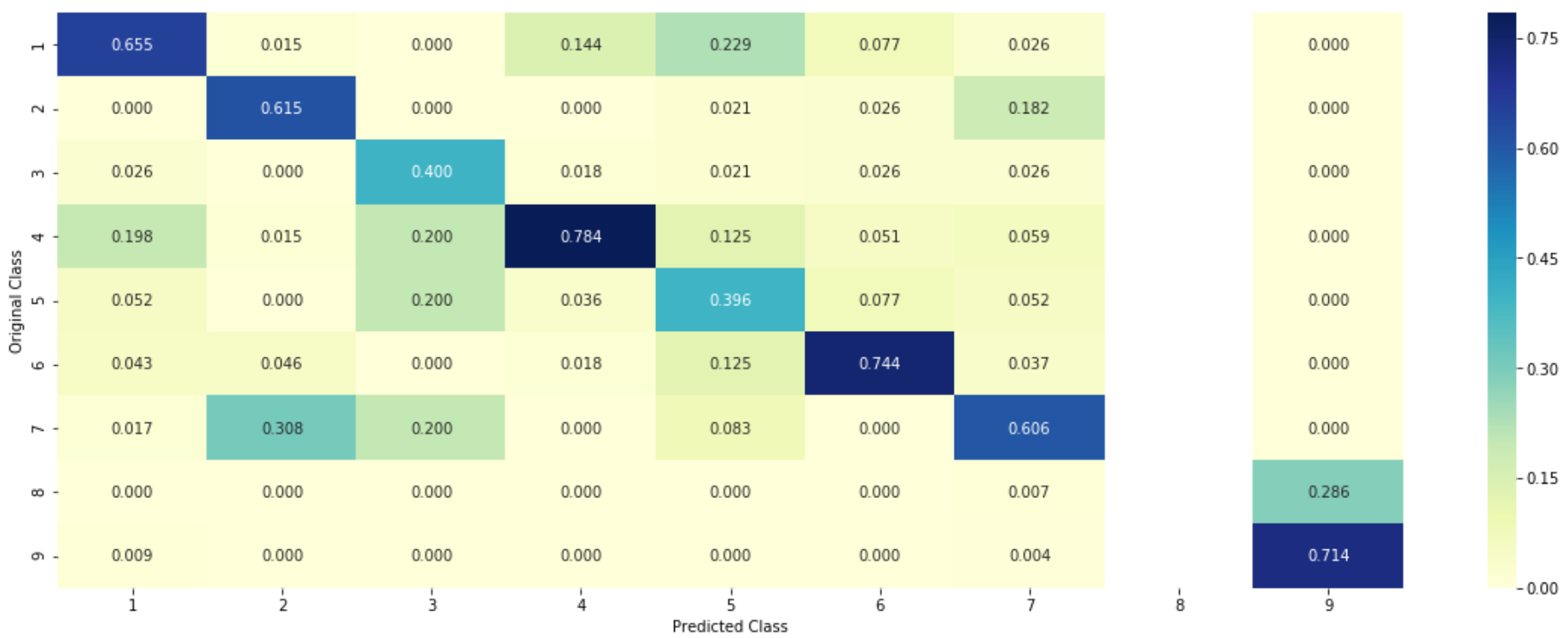
4.7.3 Maximum Voting classifier

In [101]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

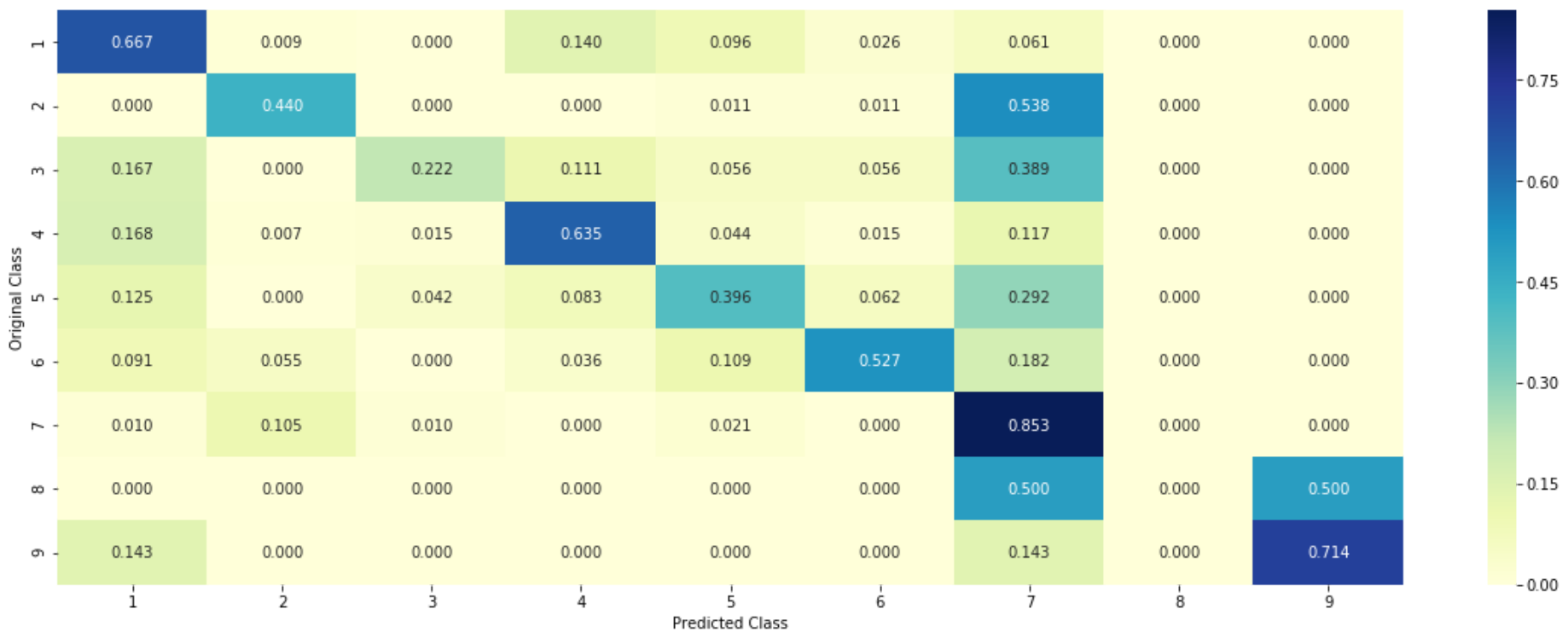
```
Log loss (train) on the VotingClassifier : 0.8652622242832645
Log loss (CV) on the VotingClassifier : 1.1072780811775143
Log loss (test) on the VotingClassifier : 1.1634526047932106
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```




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




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



selecting top 1000 words based on Tfidf values

In [113]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = text_vectorizer.transform(test_df["TEXT"])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])

# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

final = train_text_feature_onehotCoding.toarray().sum(axis=0)
top_1k_index = np.argsort(final)[::-1][0:1000]

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


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

```
Total number of unique words in train data : 53434
```

In [114]:
```
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 [115]:
```
#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)
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1)).T

train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding[:,top_1k_index], axis=0)

test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding[:,top_1k_index], axis=0)


cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding[:,top_1k_index], axis=0)
```

In [117]:
```
print( train_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)
```

```
(2124, 1000)
(665, 1000)
(532, 1000)
```

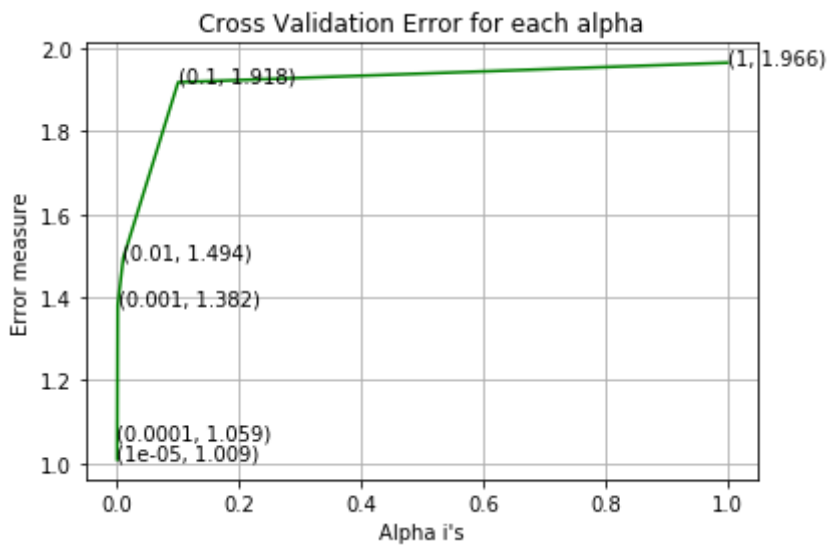
In [121]:
```
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
freq_dict=Counter(sorted_text_occur)
```

In [122]:
```
freq_dict
```

Out[122]:
```
Counter({0.8188319606646246: 1,
         1.254498078319216: 1,
         2.0923913690192535: 1,
         3.9543666232218015: 1,
         4.489142384465879: 1,
         5.086129035528415: 1,
         6.49927130313165: 1,
         1.314270005717873: 1,
         8.94692924556034: 1,
         0.05816716832682105: 1,
         10.61429553663936: 1,
         0.04948522550170249: 1,
         2.495130004237467: 1,
         14.601039559378648: 1,
         0.06071892515586797: 1,
         18.831387571996288: 1,
         3.510738313143918: 1,
         20.933296489664485: 1,
         21.635199925037398: 1,
```

In [123]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, y_train)

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.0091153017720516
For values of alpha =  0.0001 The log loss is: 1.0592067219145576
For values of alpha =  0.001 The log loss is: 1.3815396598329377
For values of alpha =  0.01 The log loss is: 1.4943473065512594
For values of alpha =  0.1 The log loss is: 1.9184499880560217
For values of alpha =  1 The log loss is: 1.965513633886195
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7423224298449157
For values of best alpha =  1e-05 The cross validation log loss is: 1.0091153017720516
For values of best alpha =  1e-05 The test log loss is: 1.1222621639912553
```

In [124]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(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 [125]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
95.842 % of word of test data appeared in train data
97.481 % of word of Cross Validation appeared in train data
```

Machine Learning Models

In [127]:
```
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

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

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

In [128]:
```
#Stacking the three types of features
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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


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

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

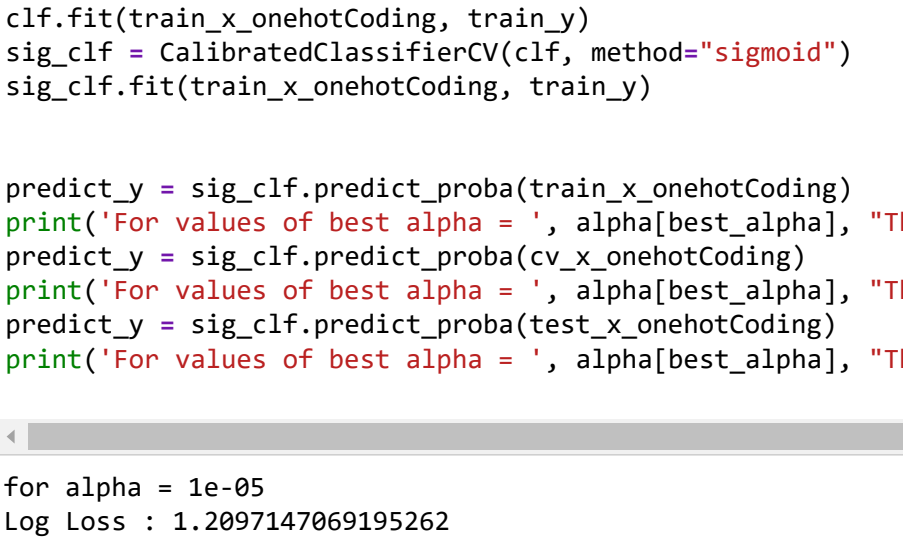
In [129]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
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)
```

```
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)
 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)
```

Naive Bayes

In [130]:
```
#Hyper parameter tuning
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

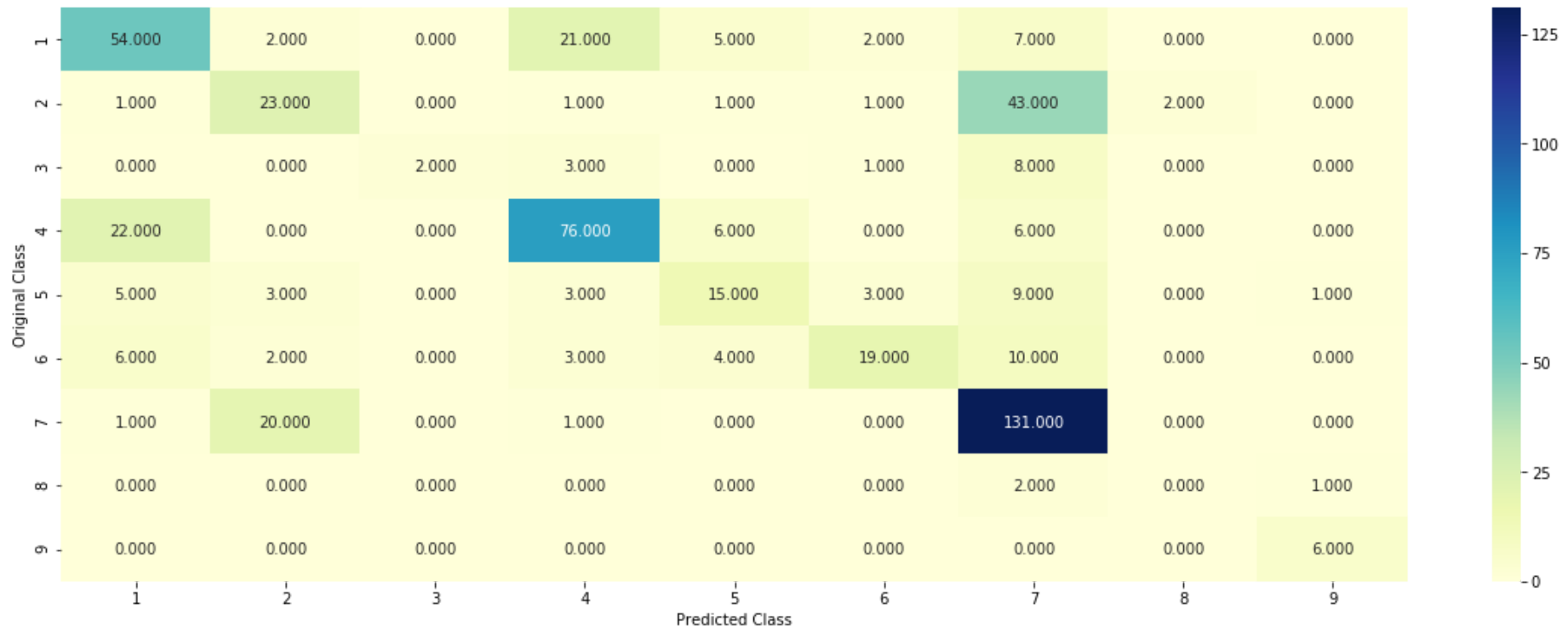
```
for alpha = 1e-05
Log Loss : 1.2097147069195262
for alpha = 0.0001
Log Loss : 1.210155739936705
for alpha = 0.001
Log Loss : 1.2107433672078416
for alpha = 0.1
Log Loss : 1.2650625428555367
for alpha = 1
Log Loss : 1.2949405607061206
for alpha = 10
Log Loss : 1.4939006058216622
for alpha = 100
Log Loss : 1.4932117679184662
for alpha = 1000
Log Loss : 1.4838817745933874
```



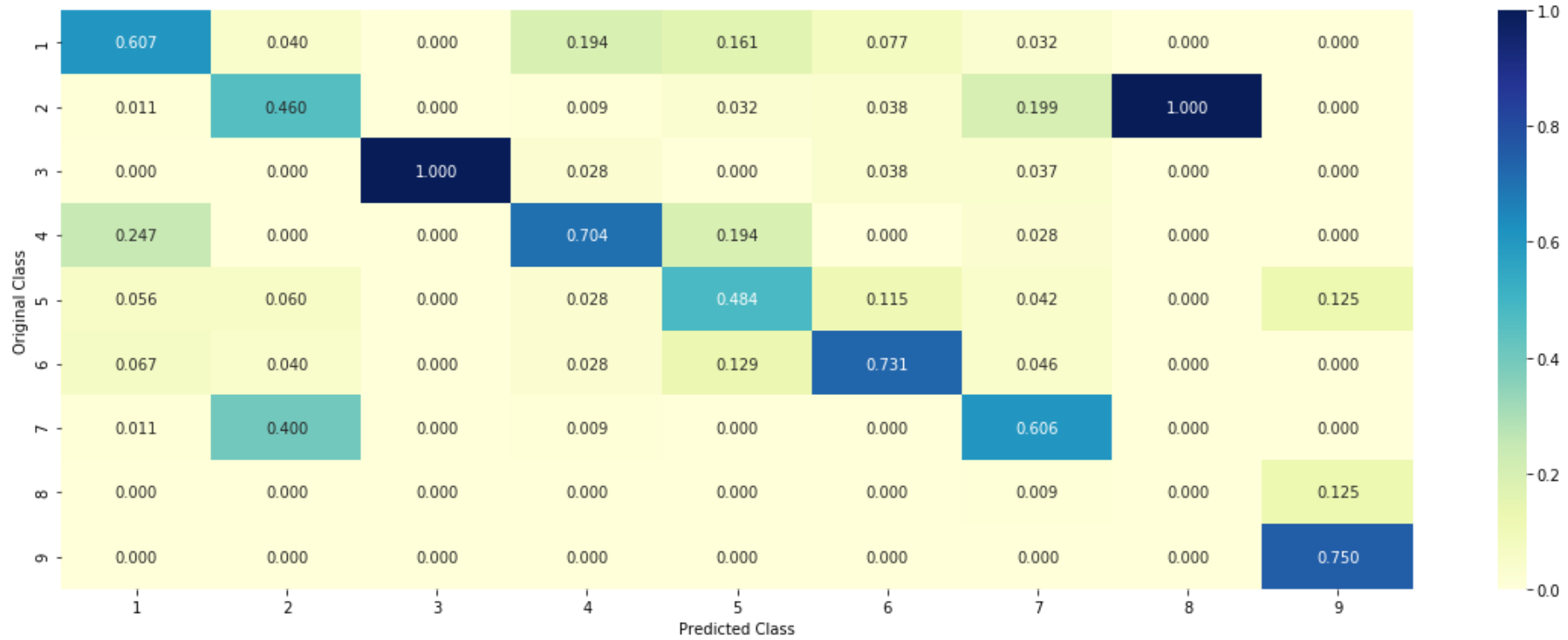

```
For values of best alpha =  1e-05 The train log loss is: 0.5374136557990257
For values of best alpha =  1e-05 The cross validation log loss is: 1.2097147069195262
For values of best alpha =  1e-05 The test log loss is: 1.2092797333197003
```

In [131]:
```
#Testing the model with best hyper paramters
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

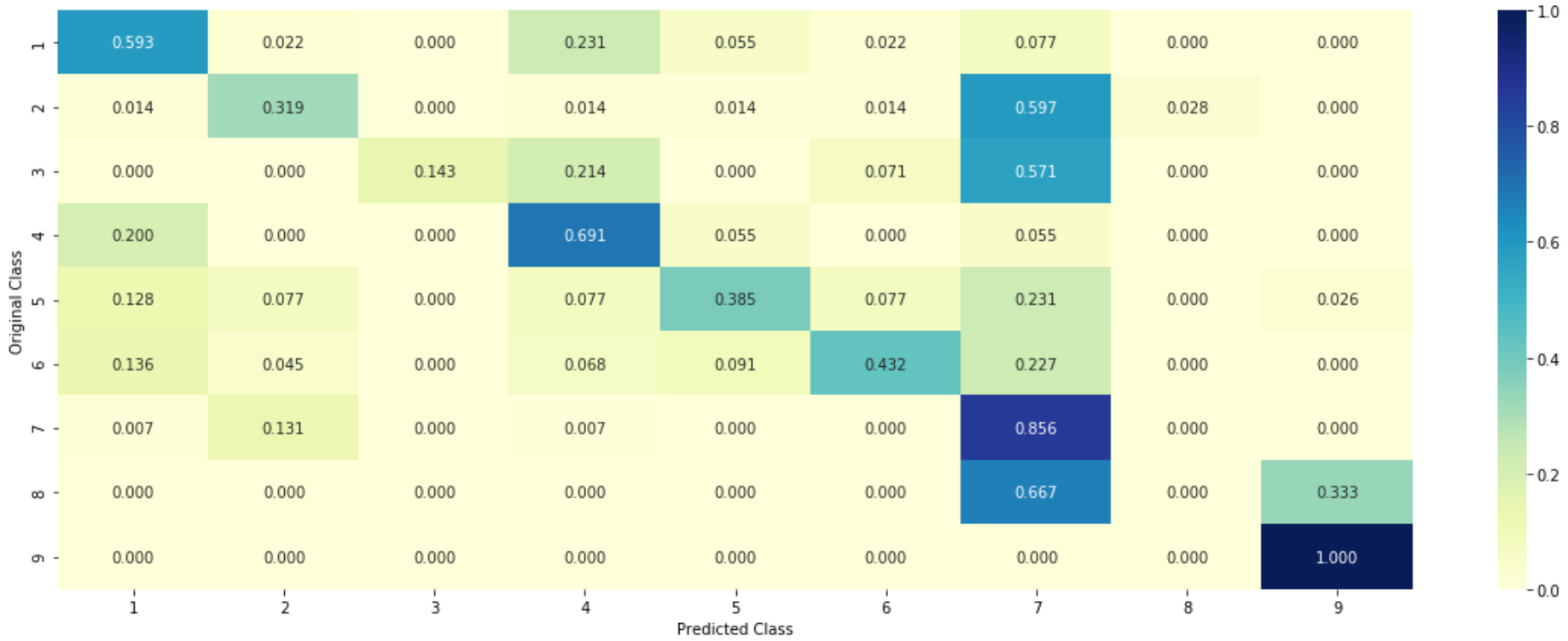
```
Log Loss : 1.2097147069195262
Number of missclassified point : 0.38721804511278196
-------------------- Confusion matrix --------------------
```




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




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



Feature Importance, Correctly classified point

In [134]:
```
test_point_index = 3
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.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['
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0722 0.6548 0.0121 0.0821 0.0391 0.0401 0.094  0.0028 0.0028]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Feature Importance, Incorrectly classified point

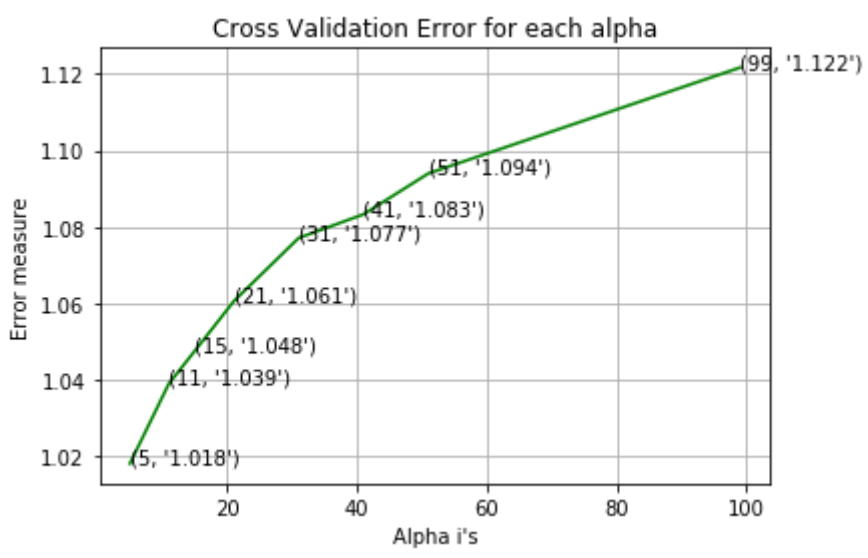
In [135]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0753 0.0466 0.0106 0.0723 0.0354 0.038  0.7169 0.0025 0.0025]]
Actual Class : 6
--------------------------------------------------
18 Text feature [000141] present in test data point [True]
62 Text feature [000132] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

K Nearest Neighbour Classification

In [136]:
```
#Hyper parameter tuning
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

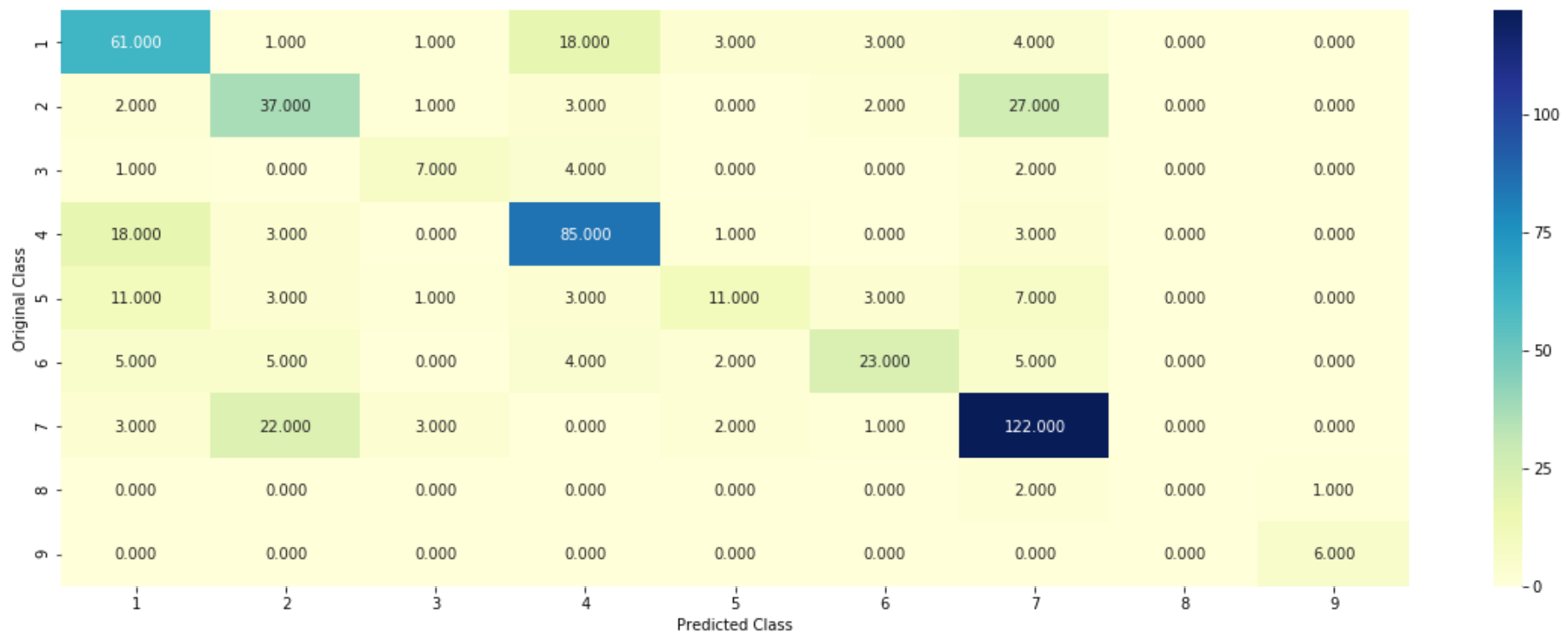
```
for alpha = 5
Log Loss : 1.018188204702352
for alpha = 11
Log Loss : 1.0390696264776218
for alpha = 15
Log Loss : 1.0476352290571909
for alpha = 21
Log Loss : 1.0605937382961506
for alpha = 31
Log Loss : 1.0771391456776076
for alpha = 41
Log Loss : 1.0833274114220066
for alpha = 51
Log Loss : 1.0940162465861756
for alpha = 99
Log Loss : 1.1216539446538119
```




```
For values of best alpha =  5 The train log loss is: 0.49230858704152763
For values of best alpha =  5 The cross validation log loss is: 1.018188204702352
For values of best alpha =  5 The test log loss is: 1.0475598954018752
```

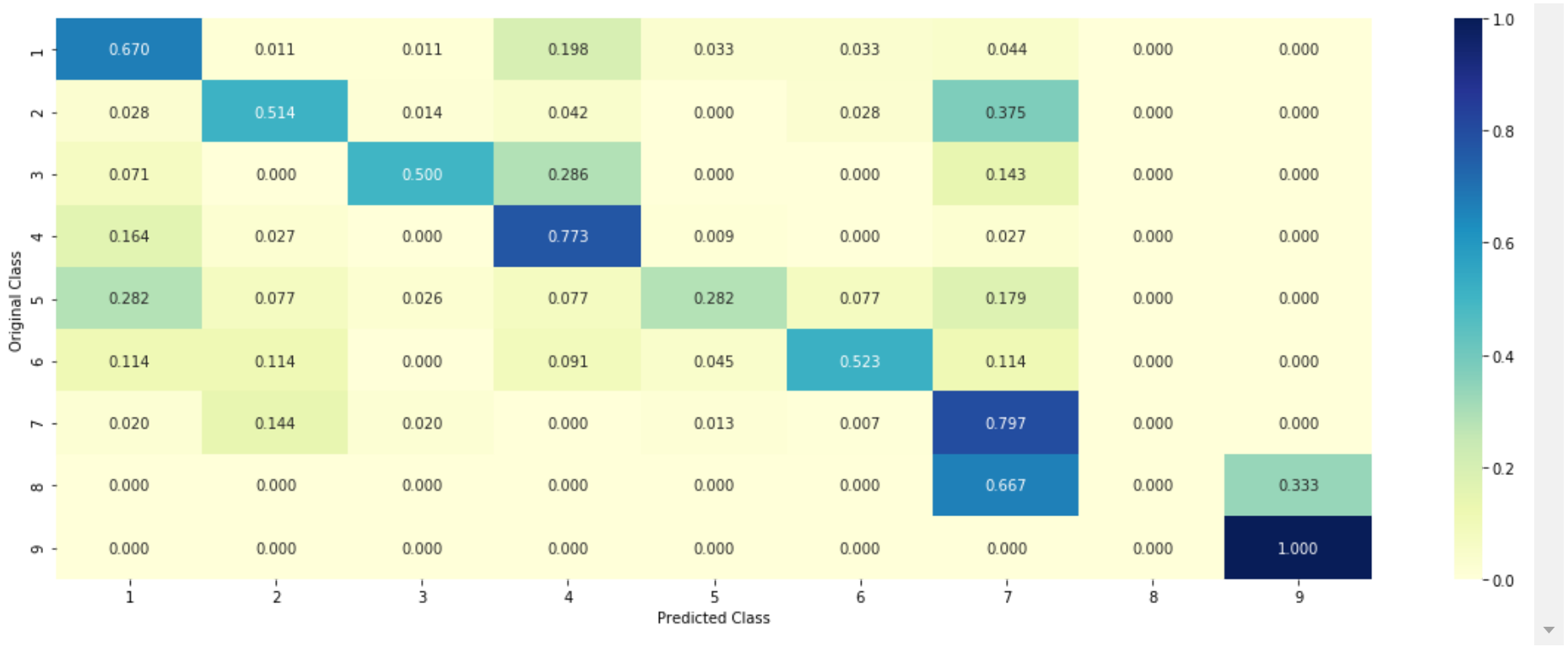
In [137]:
```
#Testing the model with best hyper paramters
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.018188204702352
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```




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


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



correctly classified points

In [141]:
```
#Sample Query point -1
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 4
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

incorrectly classified points

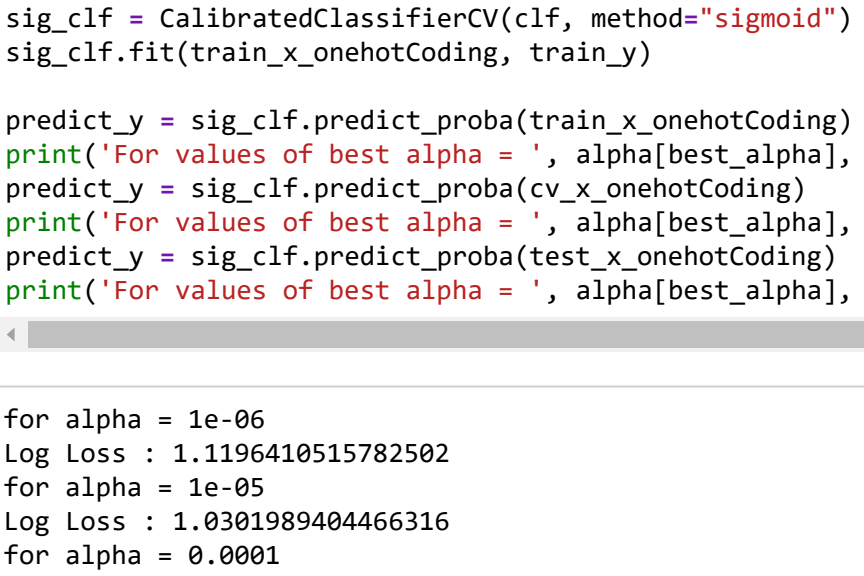
In [147]:
```
#Sample Query Point-2
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 5

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",trai
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 2
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [7 7 7 2 7]
Fequency of nearest points : Counter({7: 4, 2: 1})
```

Logistic Regression

In [148]:
```
#With Class balancing
####### Hyper paramter tuning
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

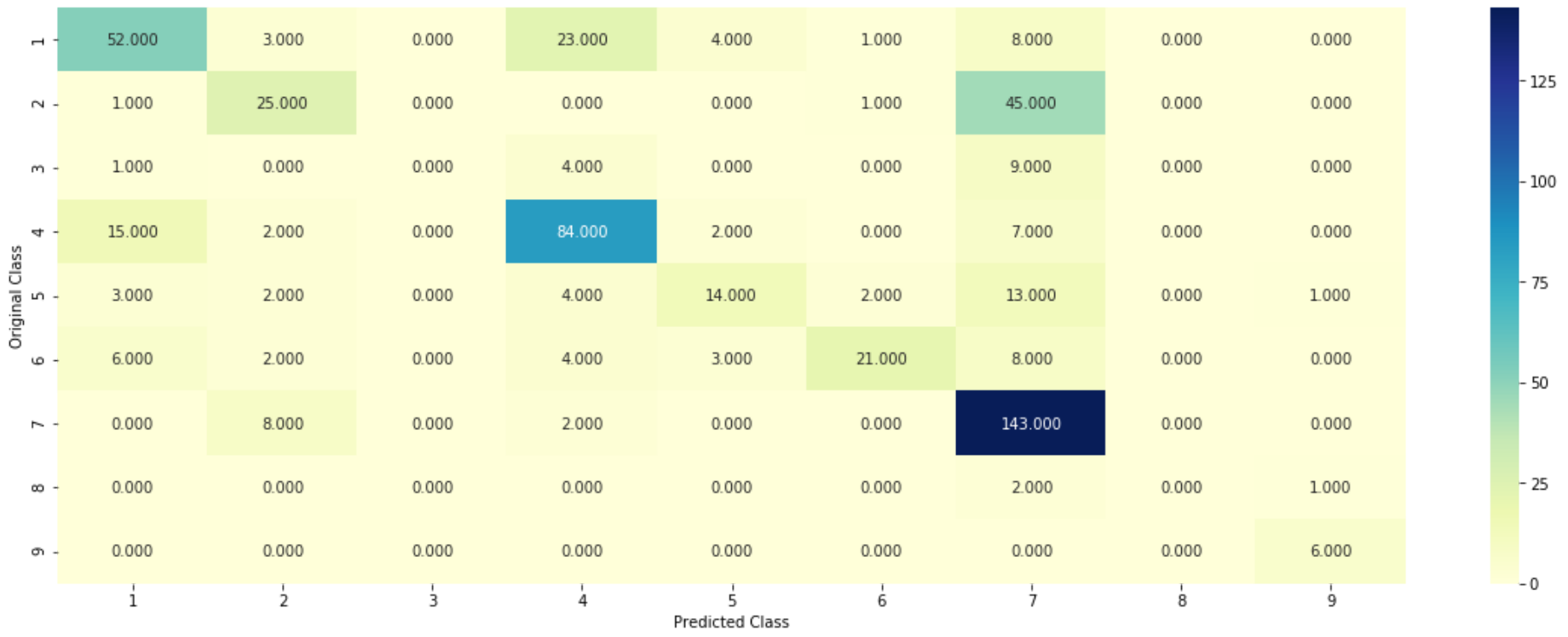
```
for alpha = 1e-06
Log Loss : 1.1196410515782502
for alpha = 1e-05
Log Loss : 1.0301989404466316
for alpha = 0.0001
Log Loss : 1.0006314727439851
for alpha = 0.001
Log Loss : 1.05019363614398
for alpha = 0.01
Log Loss : 1.2036844898786865
for alpha = 0.1
Log Loss : 1.549424971396928
for alpha = 1
Log Loss : 1.758213137313625
for alpha = 10
Log Loss : 1.78703339057697
for alpha = 100
Log Loss : 1.79035260029675
```



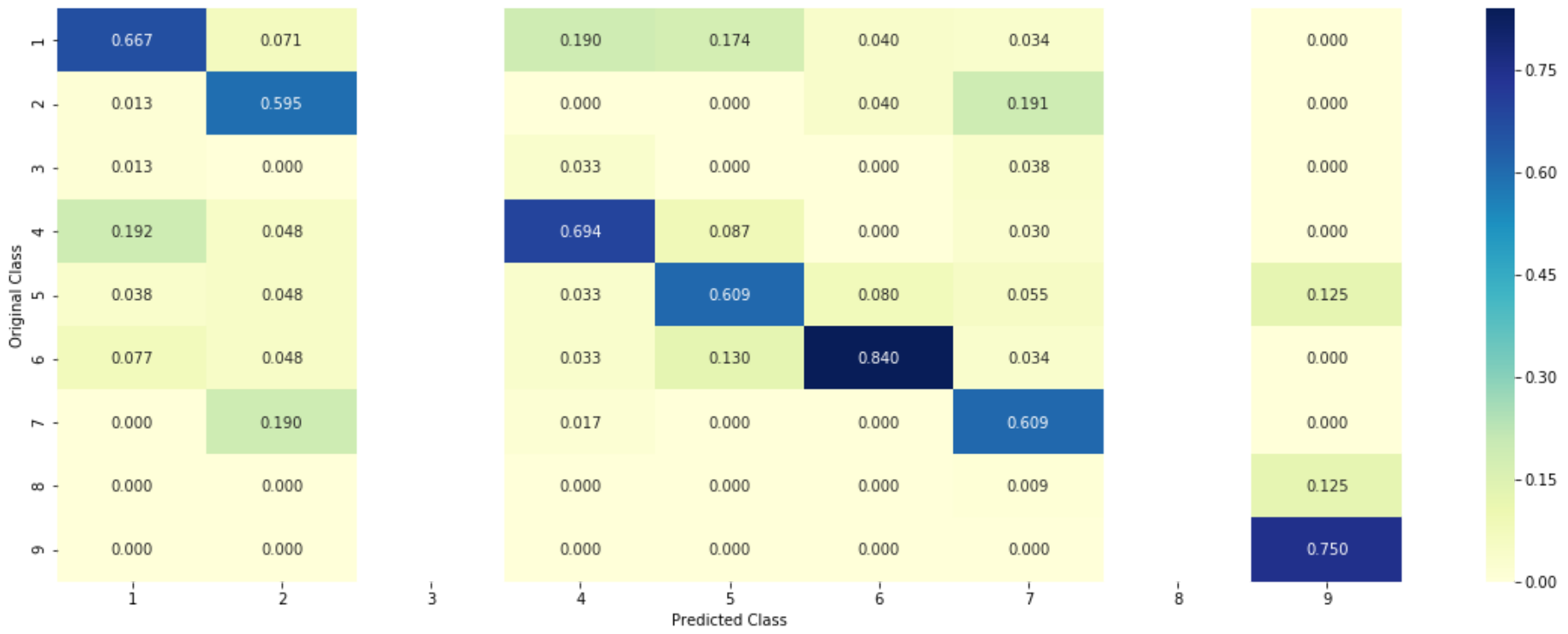
```
For values of best alpha =  0.0001 The train log loss is: 0.4503440286505412
For values of best alpha =  0.0001 The cross validation log loss is: 1.0006314727439851
For values of best alpha =  0.0001 The test log loss is: 1.0013156446178935
```

In [149]:
```
#Testing the model with best hyper paramters
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)
```

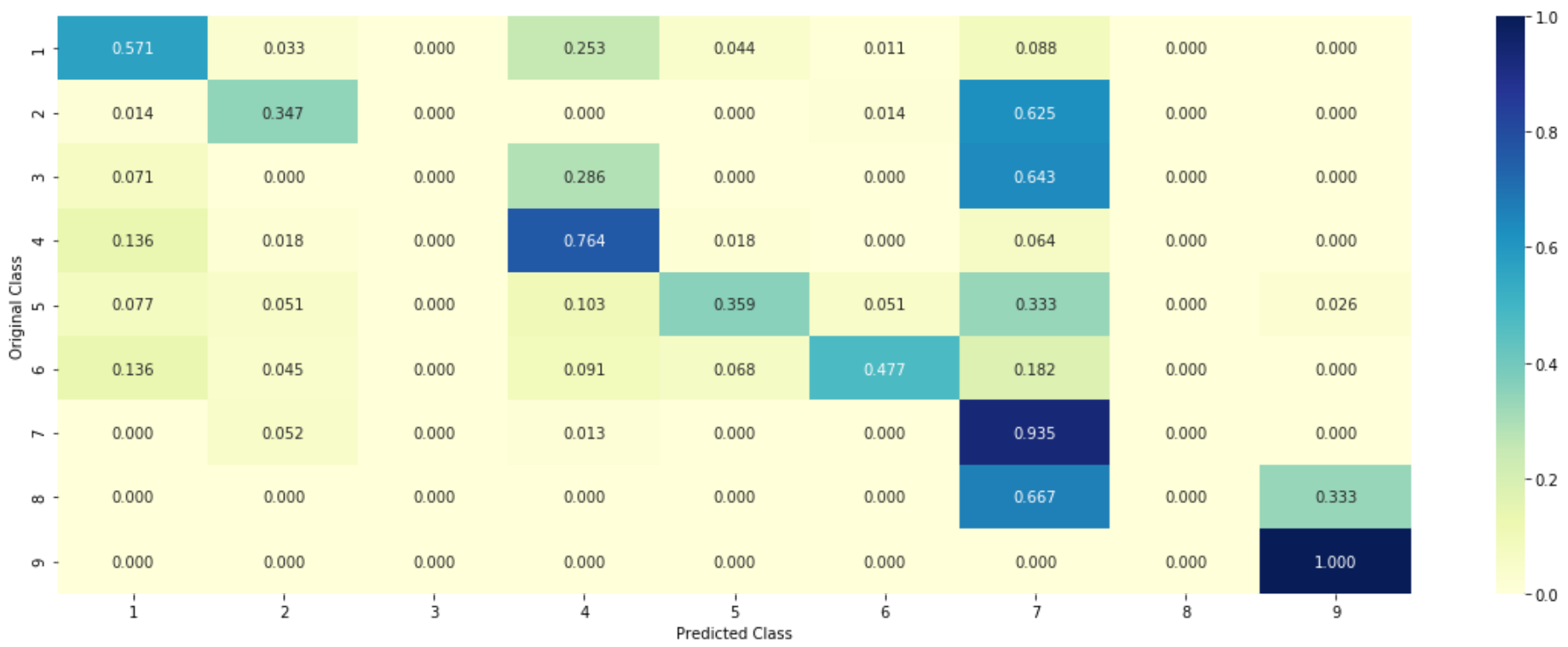
```
Log loss : 1.0006314727439851
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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




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



In [150]:
```
#Feature Importance
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

Correctly Classified point

In [152]:
```
# 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 = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[9.200e-03 8.400e-03 5.200e-03 7.400e-03 7.500e-03 1.040e-02 9.504e-01
  8.000e-04 7.000e-04]]
Actual Class : 7
--------------------------------------------------
213 Text feature [13] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

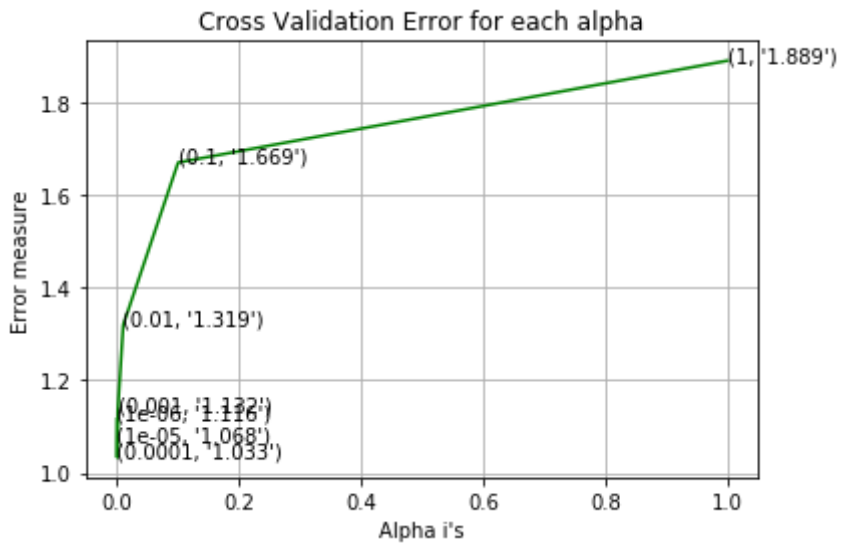
Incorrectly Classified point

In [153]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.080e-01 3.070e-02 1.100e-03 1.510e-02 2.800e-02 1.508e-01 5.653e-01
  7.000e-04 2.000e-04]]
Actual Class : 6
--------------------------------------------------
213 Text feature [13] present in test data point [True]
332 Text feature [000132] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

In [154]:
```
# Without Class balancing
#### Hyper paramter tuning
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

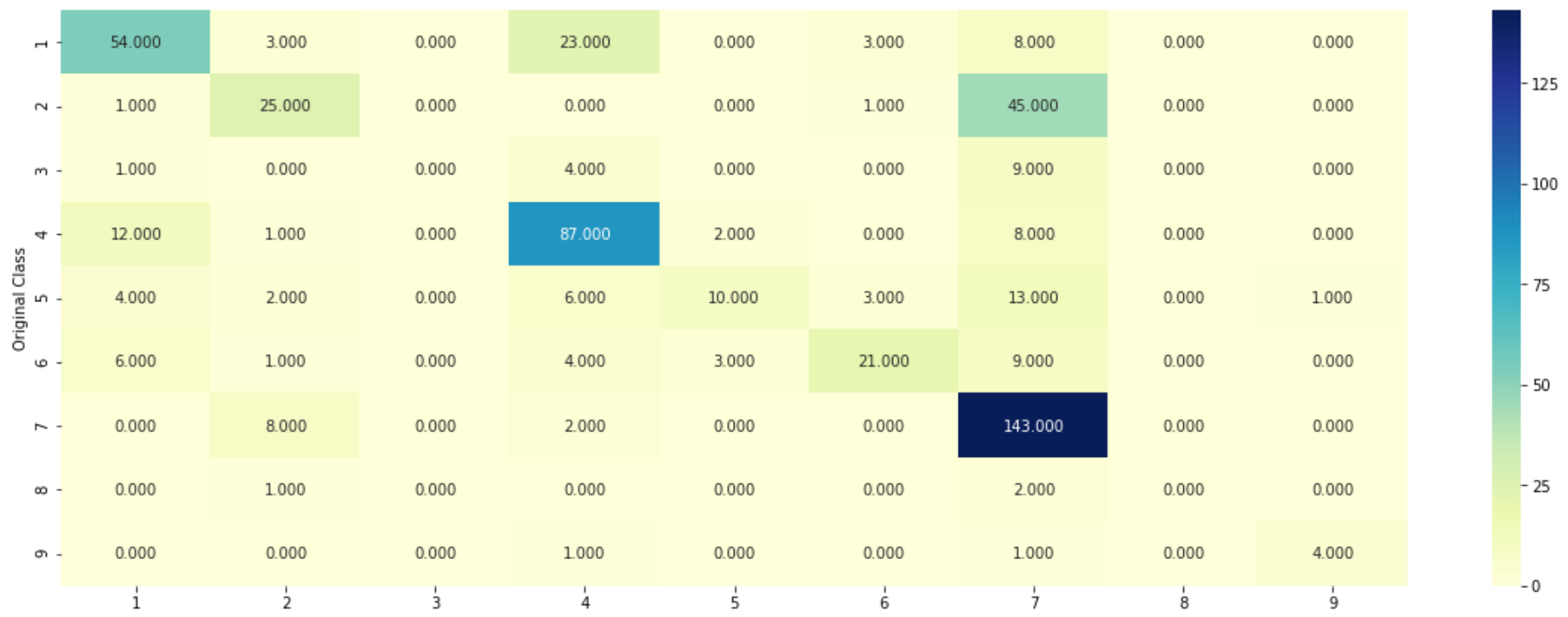
```
for alpha = 1e-06
Log Loss : 1.1156988751477417
for alpha = 1e-05
Log Loss : 1.0675288944273273
for alpha = 0.0001
Log Loss : 1.0328852907969186
for alpha = 0.001
Log Loss : 1.1316190131826194
for alpha = 0.01
Log Loss : 1.3189833319632513
for alpha = 0.1
Log Loss : 1.6694769027892542
for alpha = 1
Log Loss : 1.8892019785989902
```



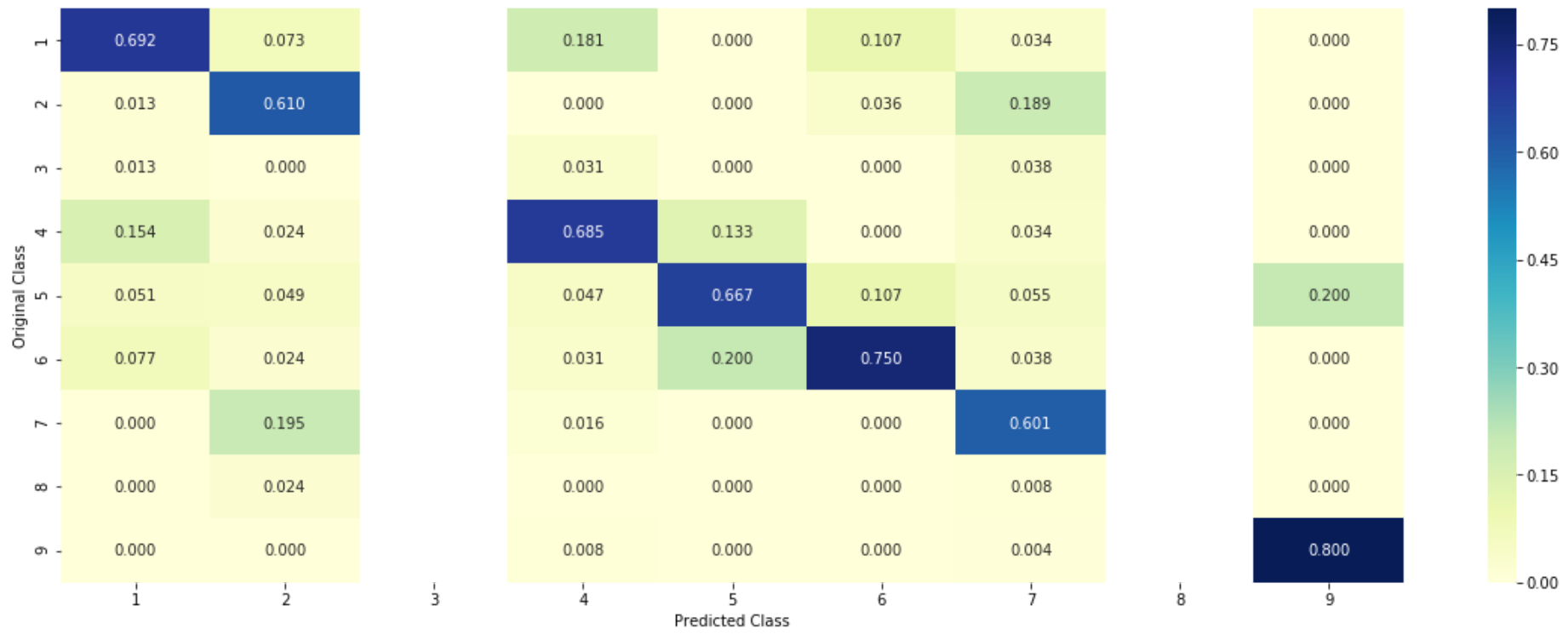

```
For values of best alpha =  0.0001 The train log loss is: 0.4388291368643059
For values of best alpha =  0.0001 The cross validation log loss is: 1.0328852907969186
For values of best alpha =  0.0001 The test log loss is: 1.0151847126764988
```

In [155]:
```
#Testing model with best hyper parameters
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 : 1.0328852907969186
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




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




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

Feature Importance, Correctly Classified point

In [158]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.000e-02 8.400e-03 3.900e-03 8.200e-03 6.800e-03 1.040e-02 9.517e-01
  4.000e-04 3.000e-04]]
Actual Class : 7
--------------------------------------------------
268 Text feature [13] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

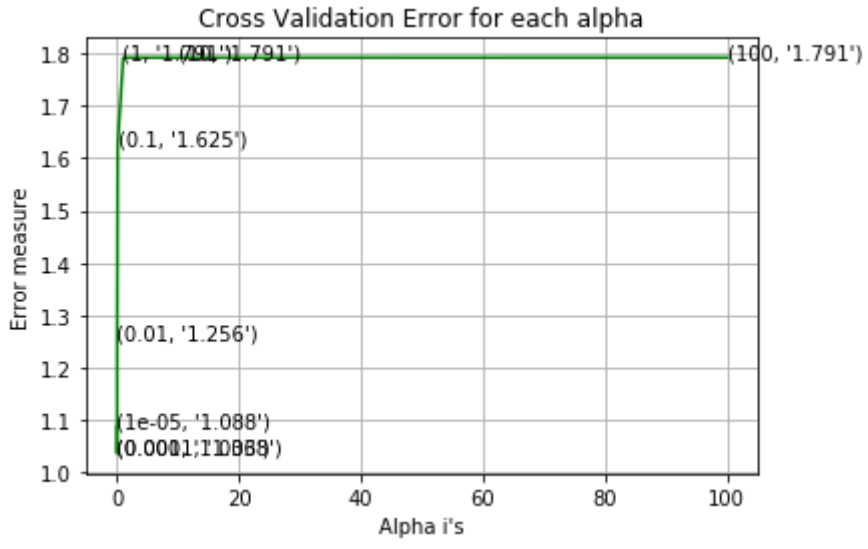
#Feature Importance, Inorrectly Classified point

In [159]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.342e-01 2.780e-02 7.000e-04 1.580e-02 2.250e-02 1.391e-01 5.598e-01
  1.000e-04 1.000e-04]]
Actual Class : 6
--------------------------------------------------
268 Text feature [13] present in test data point [True]
349 Text feature [000132] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

In [160]:
```
#Linear Support Vector Machines
#Hyper paramter tuning
alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for C = 1e-05
Log Loss : 1.0875247732702573
for C = 0.0001
Log Loss : 1.0383440452496118
for C = 0.001
Log Loss : 1.0363256112770094
for C = 0.01
Log Loss : 1.255933892885779
for C = 0.1
Log Loss : 1.6250490523251633
for C = 1
Log Loss : 1.7911898284448413
for C = 10
Log Loss : 1.7911592707031045
for C = 100
Log Loss : 1.7912226151506119
```

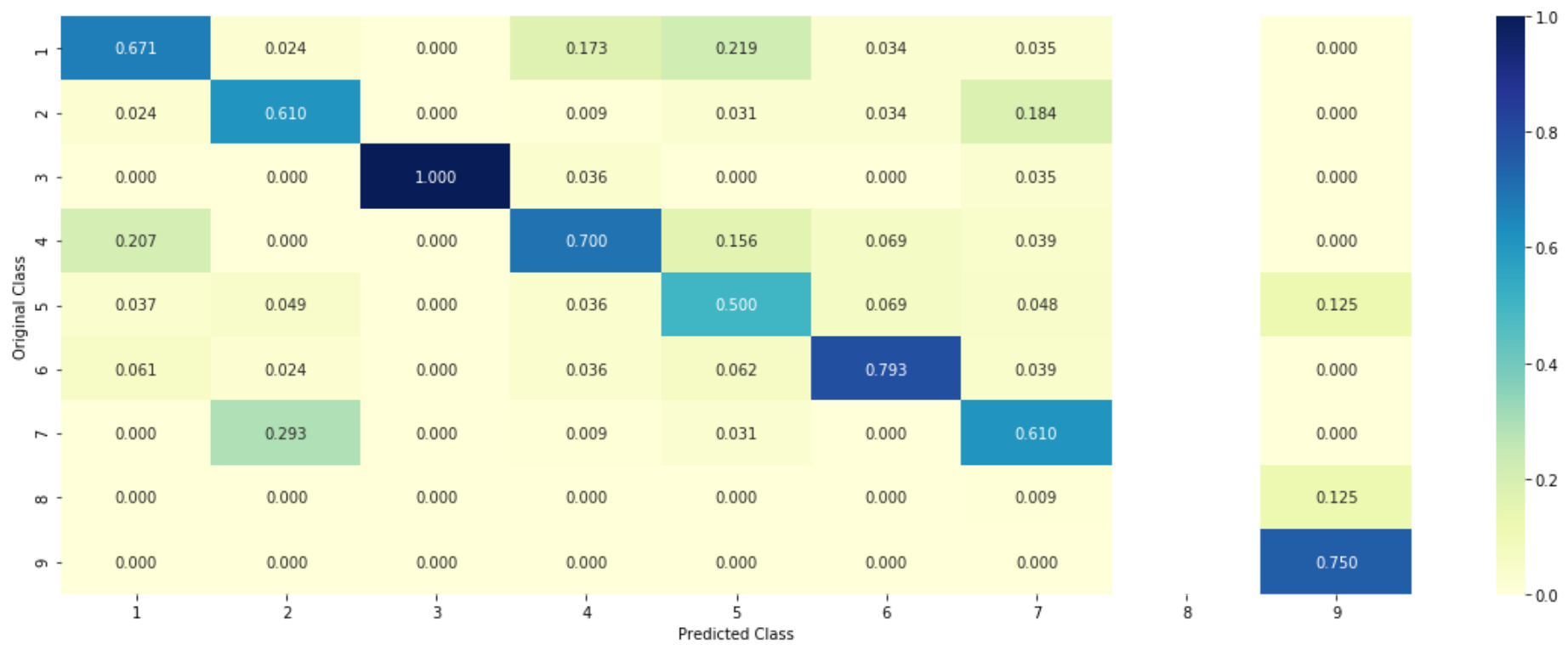



```
For values of best alpha =  0.001 The train log loss is: 0.5688702030152898
For values of best alpha =  0.001 The cross validation log loss is: 1.0363256112770094
For values of best alpha =  0.001 The test log loss is: 1.0688129687338648
```

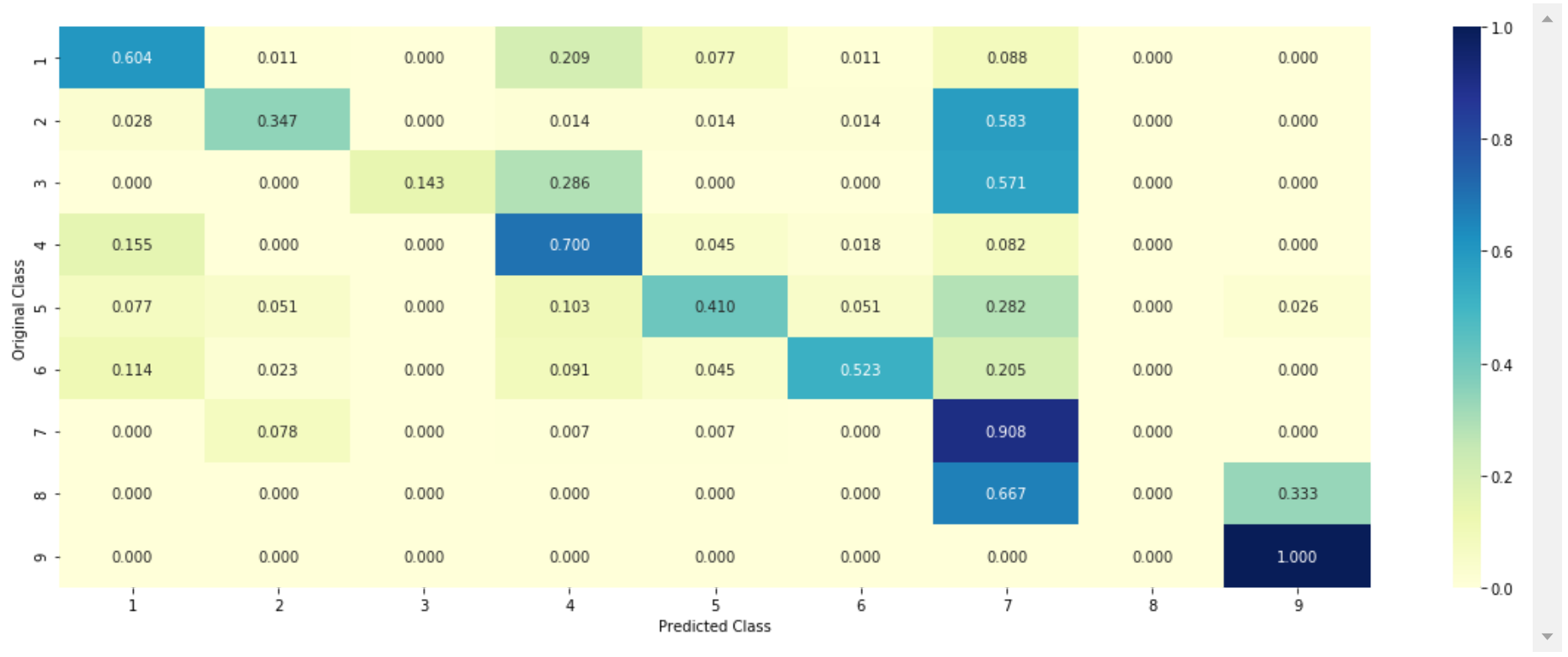
In [161]:
```
#Testing model with best hyper parameters
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.0363256112770094
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```


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




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



For Correctly classified point

In [163]:
```
#Feature Importance
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0492 0.017  0.0098 0.0452 0.0205 0.0229 0.8315 0.0016 0.0021]]
Actual Class : 7
--------------------------------------------------
202 Text feature [13] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

For Incorrectly classified point

In [164]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.2115 0.0517 0.0048 0.1231 0.0529 0.1757 0.376  0.0026 0.0017]]
Actual Class : 6
--------------------------------------------------
202 Text feature [13] present in test data point [True]
295 Text feature [000132] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

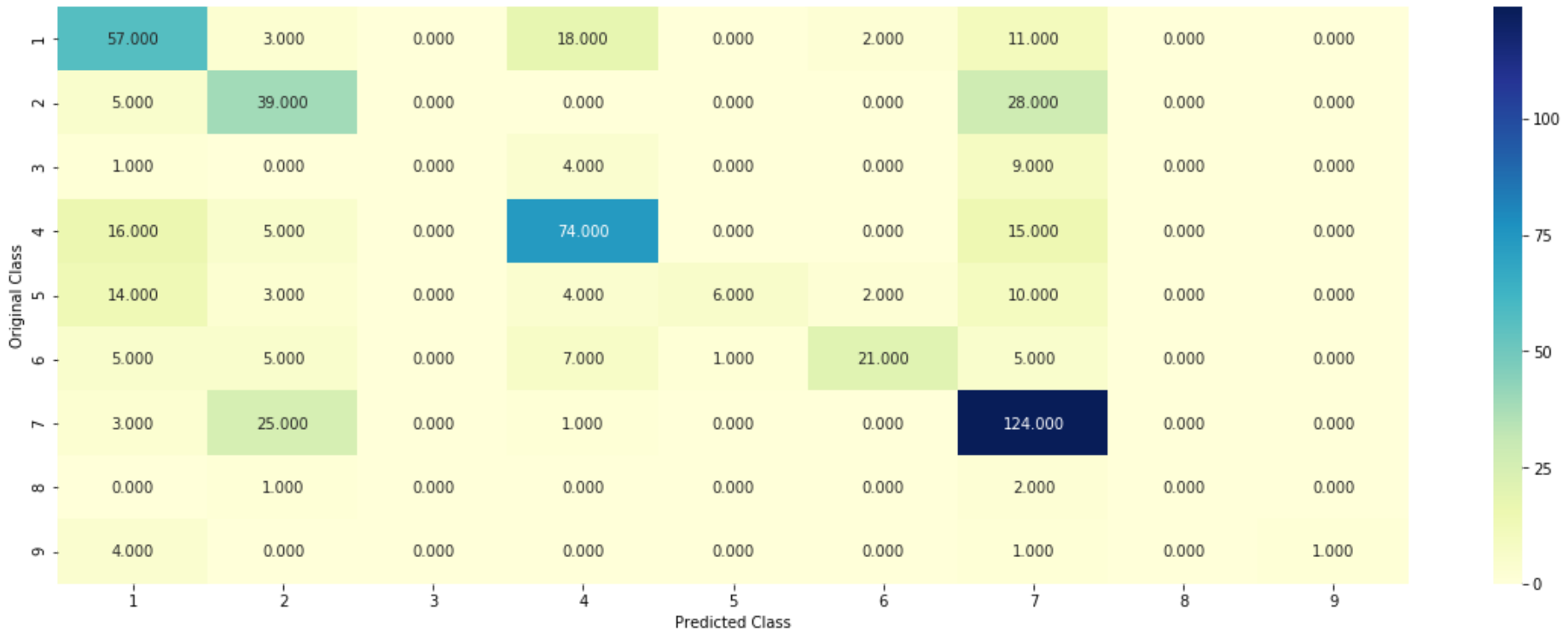
Random Forest Classifier

In [165]:
```
# Hyper paramter tuning (With One hot Encoding)
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train, predict_y,
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, pr
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(y_test, predict_y, l
```

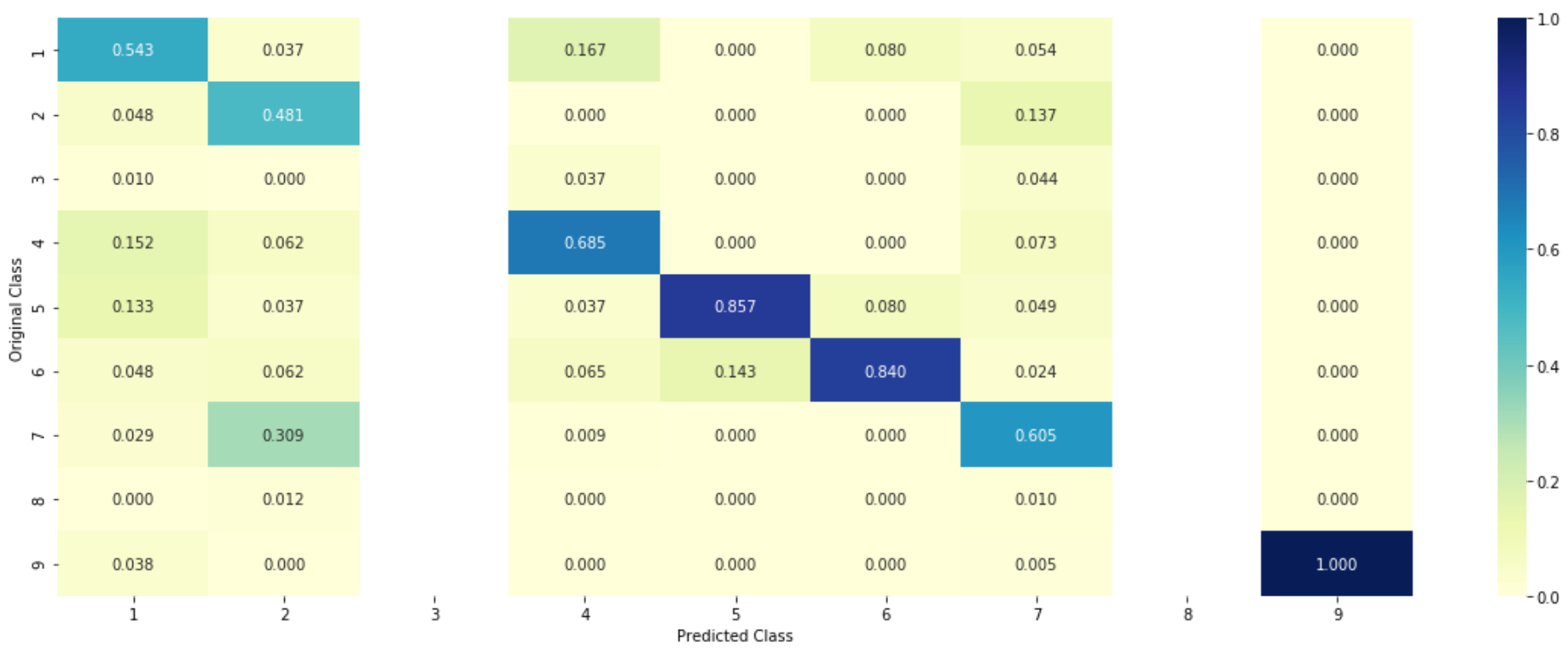
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2727182344262546
for n_estimators = 100 and max depth =  10
Log Loss : 1.2748545978745998
for n_estimators = 200 and max depth =  5
Log Loss : 1.2597886566448282
for n_estimators = 200 and max depth =  10
Log Loss : 1.2655489765566494
for n_estimators = 500 and max depth =  5
Log Loss : 1.2578265404144433
for n_estimators = 500 and max depth =  10
Log Loss : 1.2574475812612158
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2546143648157766
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2544747378227505
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2546245875062723
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2530082945054961
For values of best estimator =  2000 The train log loss is: 0.584982310595121
For values of best estimator =  2000 The cross validation log loss is: 1.2530082945054961
For values of best estimator =  2000 The test log loss is: 1.2113308838497308
```

In [166]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

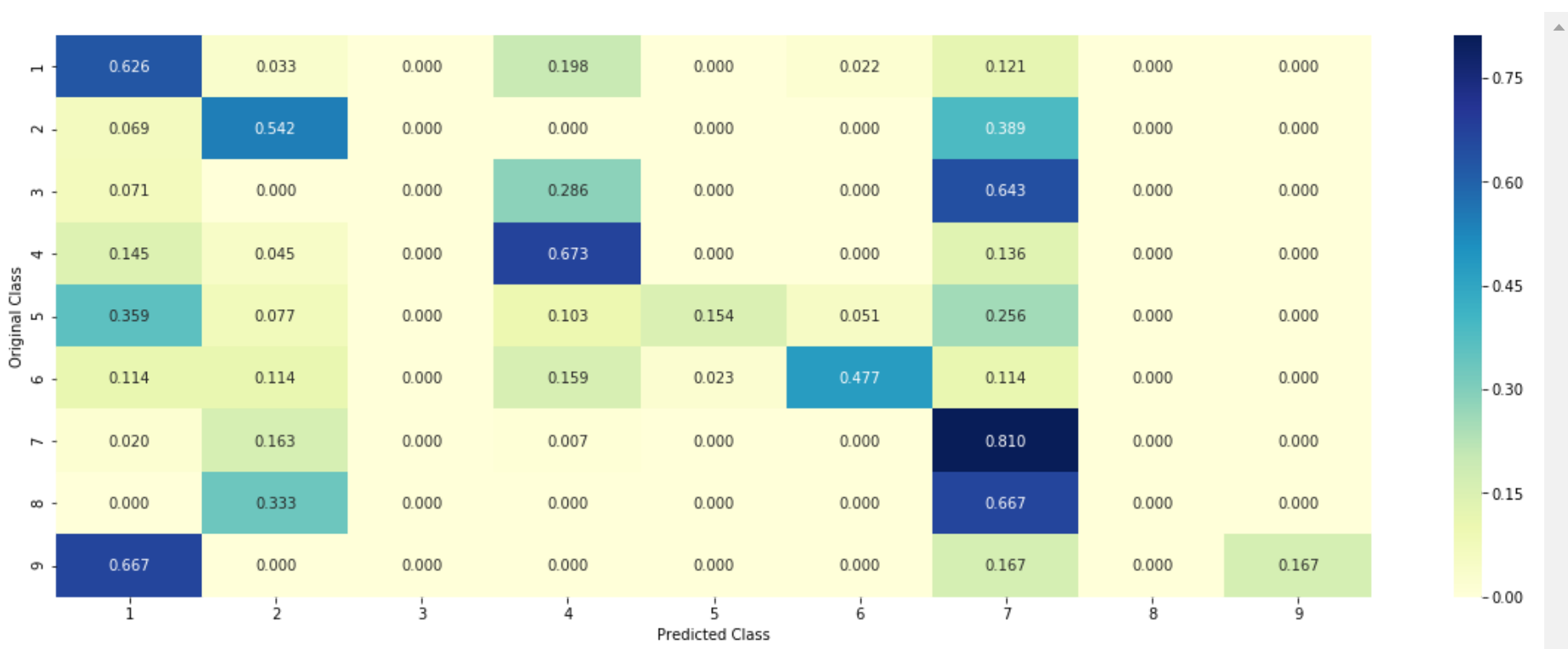
```
Log loss : 1.2530082945054961
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```




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




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



Correctly Classified point

In [167]:
```
#Feature Importance
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.346  0.0395 0.0256 0.1574 0.3014 0.078  0.0407 0.0056 0.0059]]
Actual Class : 1
--------------------------------------------------
58 Text feature [09] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

Inorrectly Classified point

In [168]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index]
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2544 0.1364 0.0274 0.1179 0.0679 0.1823 0.1983 0.0075 0.0079]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [000141] present in test data point [True]
42 Text feature [000] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

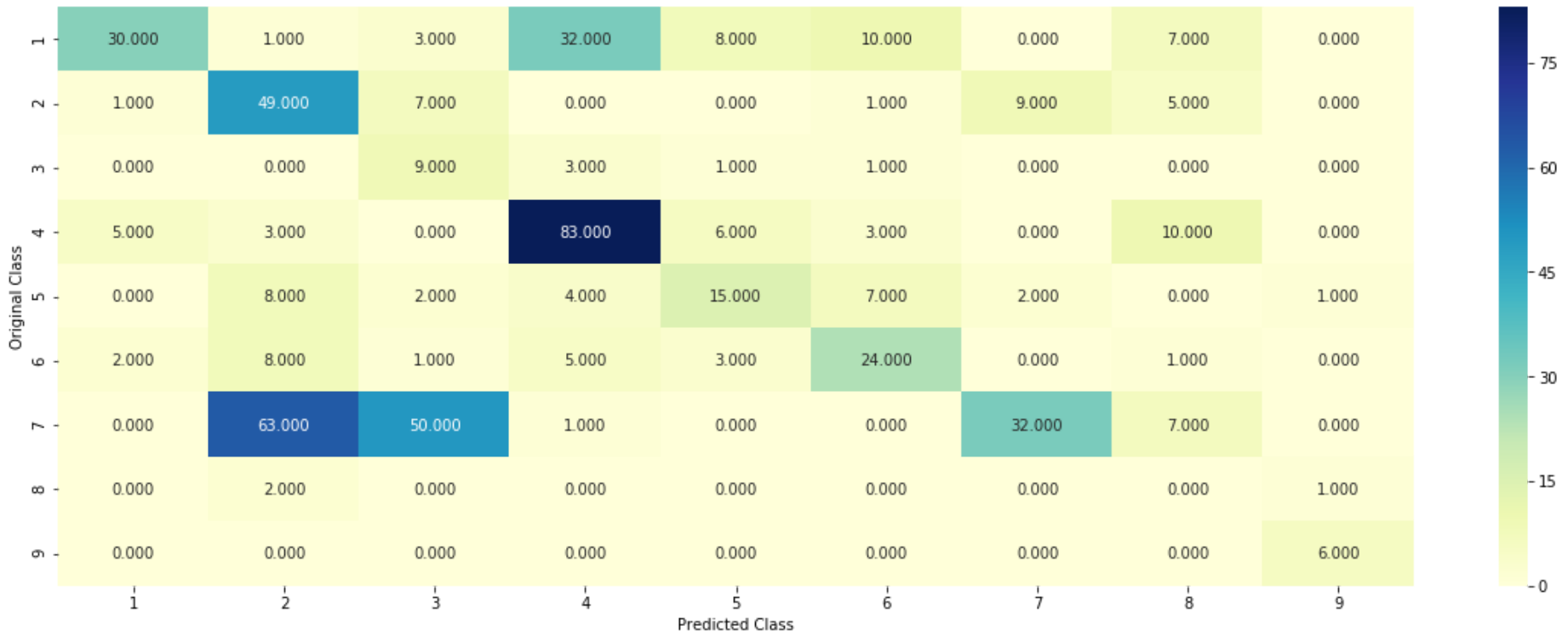
In [169]:
```
#Hyper paramter tuning (With Response Coding)
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha
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, lab
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, predic
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_test, predict_y, label
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0905612101237026
for n_estimators = 10 and max depth =  3
Log Loss : 1.573732191984886
for n_estimators = 10 and max depth =  5
Log Loss : 1.746112533979271
for n_estimators = 10 and max depth =  10
Log Loss : 1.639200638202826
for n_estimators = 50 and max depth =  2
Log Loss : 1.7632064527366265
for n_estimators = 50 and max depth =  3
Log Loss : 1.4986211006578782
for n_estimators = 50 and max depth =  5
Log Loss : 1.5310851900387812
for n_estimators = 50 and max depth =  10
Log Loss : 1.6432835724214305
for n_estimators = 100 and max depth =  2
Log Loss : 1.6454491237928843
for n_estimators = 100 and max depth =  3
Log Loss : 1.5258648117744968
for n_estimators = 100 and max depth =  5
Log Loss : 1.4716415375589946
for n_estimators = 100 and max depth =  10
Log Loss : 1.675139165629154
for n_estimators = 200 and max depth =  2
Log Loss : 1.723515334185217
for n_estimators = 200 and max depth =  3
Log Loss : 1.6082454956826098
for n_estimators = 200 and max depth =  5
Log Loss : 1.4411686111895414
for n_estimators = 200 and max depth =  10
Log Loss : 1.689651879445109
for n_estimators = 500 and max depth =  2
Log Loss : 1.753187269330325
for n_estimators = 500 and max depth =  3
Log Loss : 1.6368046410033554
for n_estimators = 500 and max depth =  5
Log Loss : 1.4169628772612128
for n_estimators = 500 and max depth =  10
Log Loss : 1.7132748891590908
for n_estimators = 1000 and max depth =  2
Log Loss : 1.748868541190876
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6166174234796715
for n_estimators = 1000 and max depth =  5
Log Loss : 1.407575385361305
```

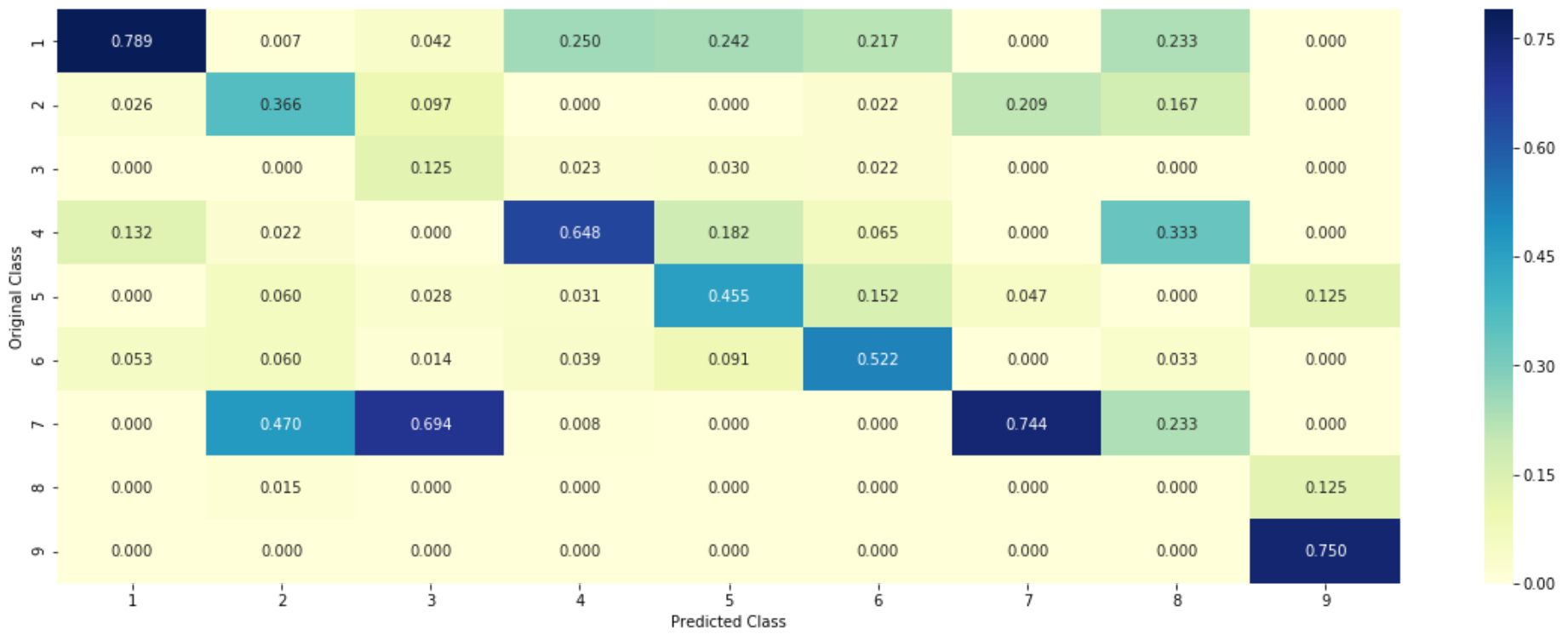
```
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6945463207603786
For values of best alpha =  1000 The train log loss is: 0.05266733231671452
For values of best alpha =  1000 The cross validation log loss is: 1.4075753853613053
For values of best alpha =  1000 The test log loss is: 1.3875976202293747
```

In [170]:
```
# Testing model with best hyper parameters (Response Coding)
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], criterion='g
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.407575385361305
Number of mis-classified points : 0.5338345864661654
-------------------- Confusion matrix --------------------
```




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




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

correctly classfied points

In [173]:
```
#Feature Importance
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 3
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

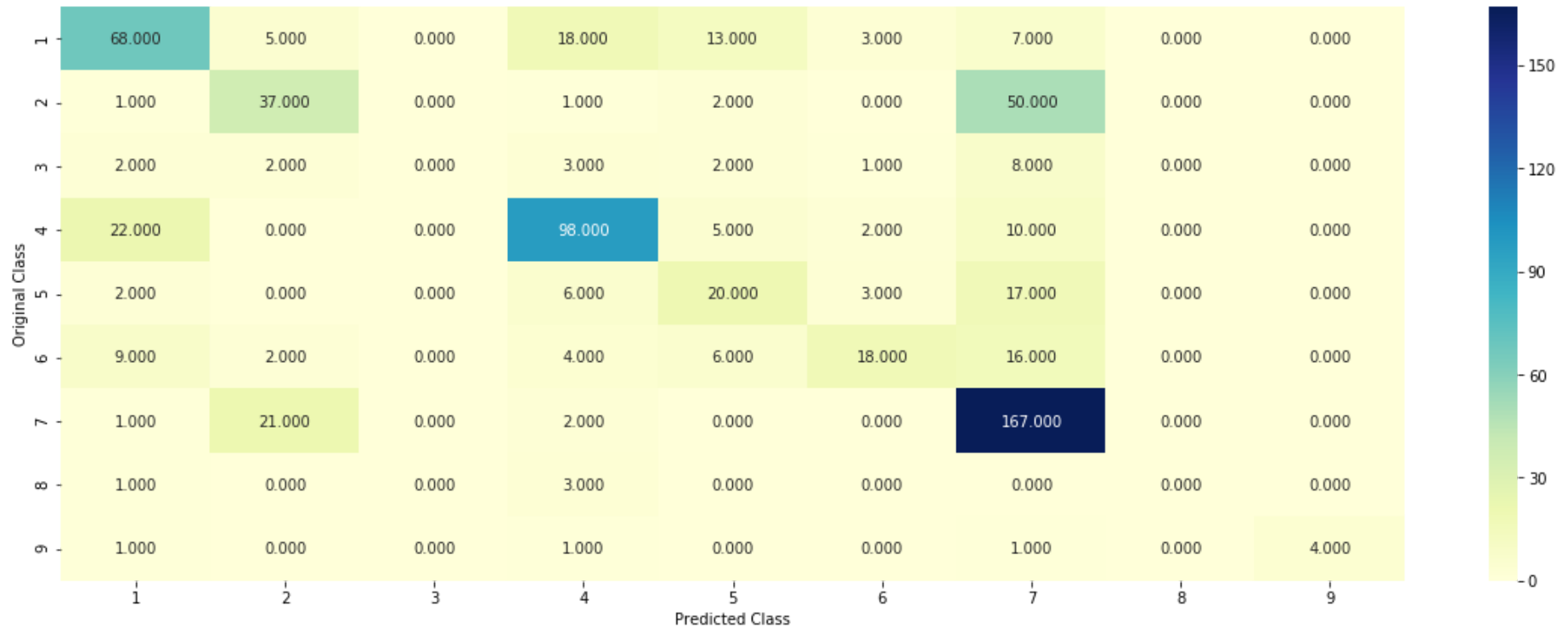
```
Predicted Class : 2
Predicted Class Probabilities: [[0.0155 0.7387 0.0322 0.0178 0.0145 0.0329 0.0378 0.0808 0.0298]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene 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
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

In [174]:
```
#Stack the models
##testing with hyper parameter tuning
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

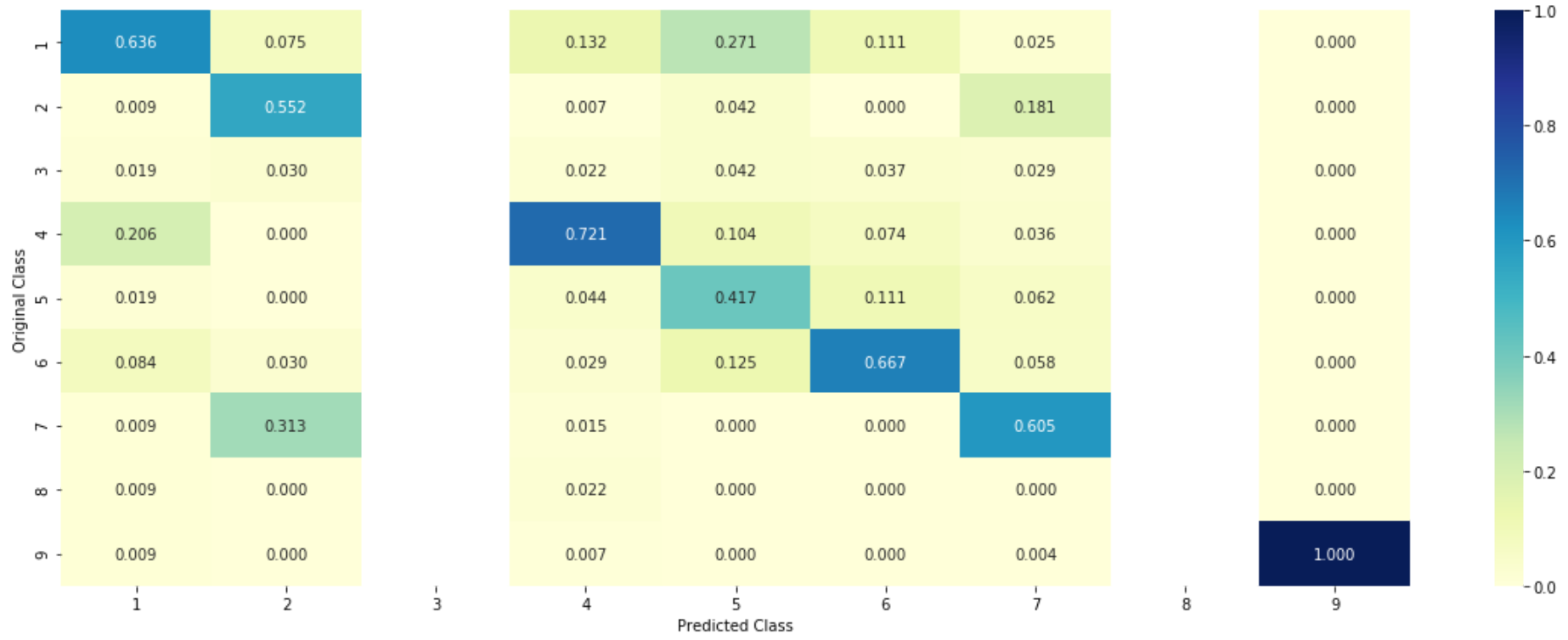
```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.21
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.508
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.192
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.409
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.873
```

In [175]:
```
#testing the model with the best hyper parameters
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

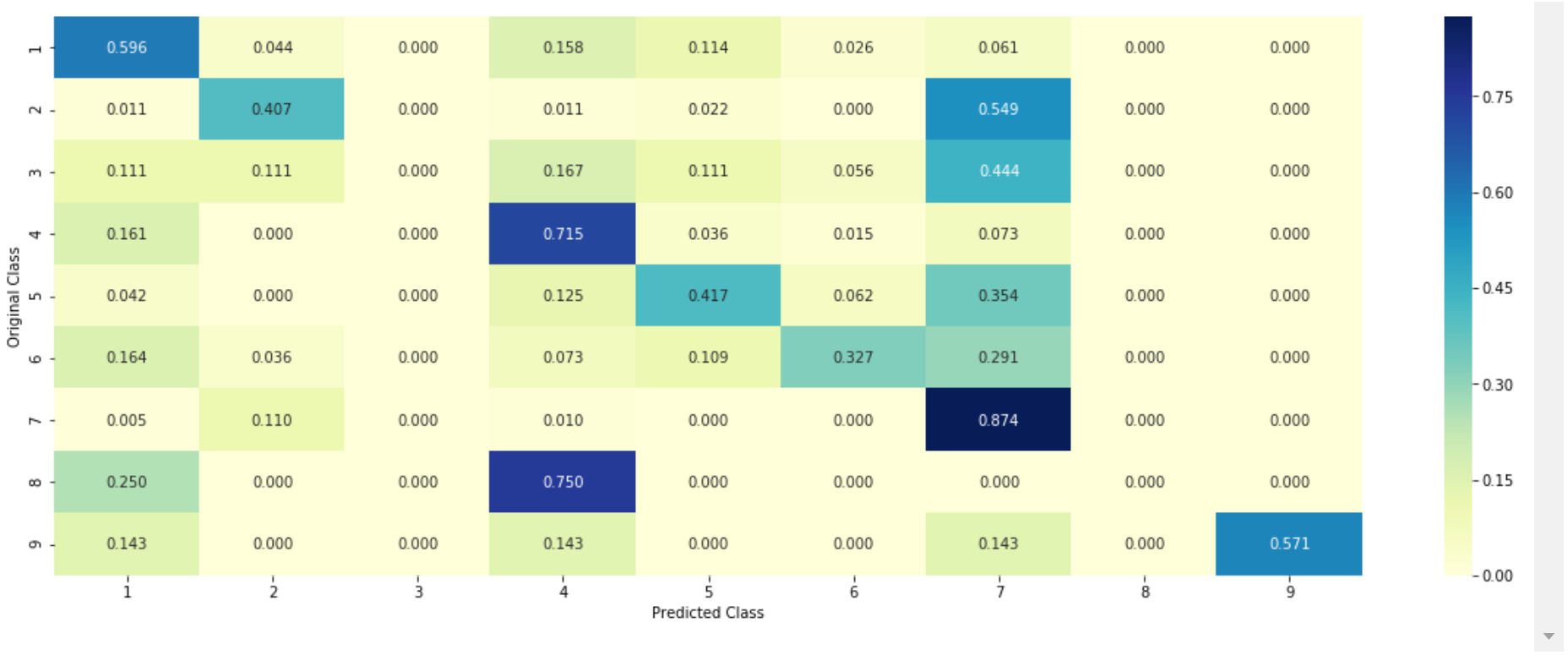
```
Log loss (train) on the stacking classifier : 0.5651416896640594
Log loss (CV) on the stacking classifier : 1.1922376042630183
Log loss (test) on the stacking classifier : 1.1737415558289126
Number of missclassified point : 0.3804511278195489
-------------------- Confusion matrix --------------------
```




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

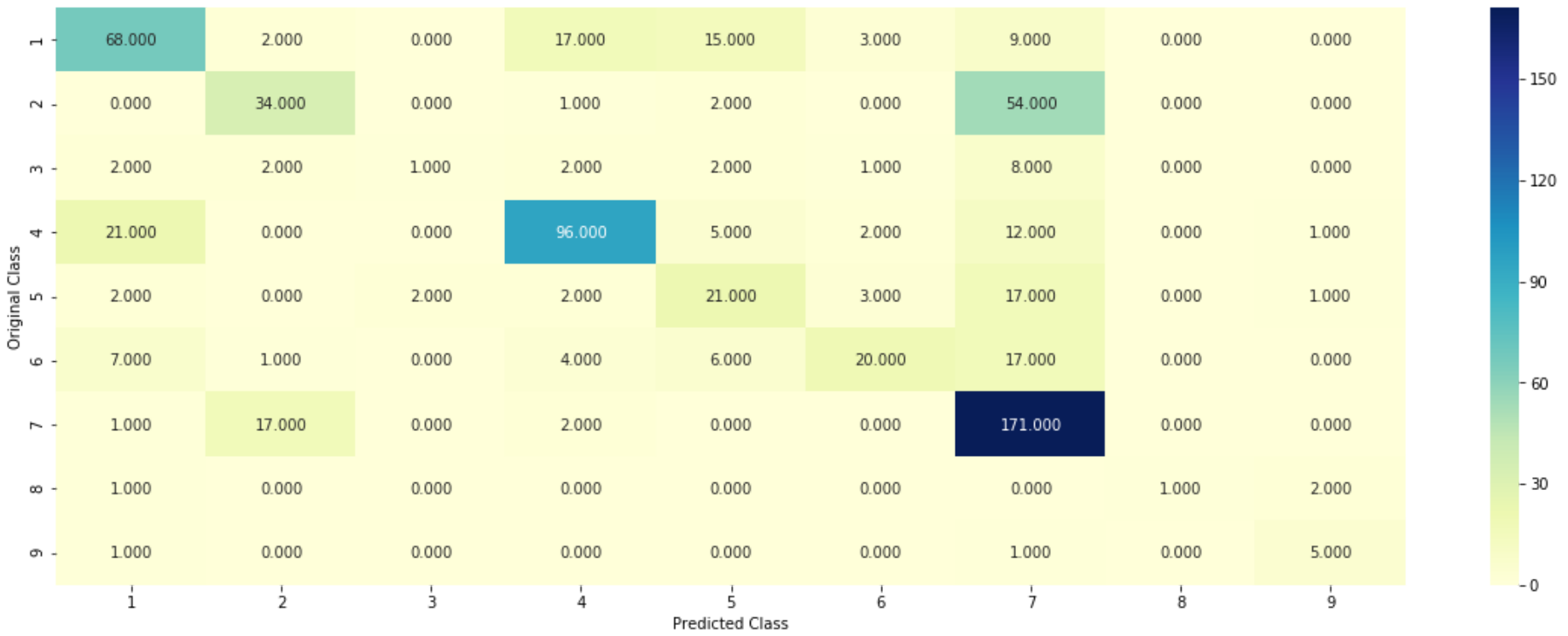



```
-------------------- Recall matrix (Row sum=1) --------------------
```


Original Class
1
2
3
4
5
6
7
8
9
0.596 0.044 0.000 0.158 0.114 0.026 0.061 0.000 0.000
0.011 0.407 0.000 0.011 0.022 0.000 0.549 0.000 0.000
0.111 0.111 0.000 0.167 0.111 0.056 0.444 0.000 0.000
0.161 0.000 0.000 0.715 0.036 0.015 0.073 0.000 0.000
0.042 0.000 0.000 0.125 0.417 0.062 0.354 0.000 0.000
0.164 0.036 0.000 0.073 0.109 0.327 0.291 0.000 0.000
0.005 0.110 0.000 0.010 0.000 0.000 0.874 0.000 0.000
0.250 0.000 0.000 0.750 0.000 0.000 0.000 0.000 0.000
0.143 0.000 0.000 0.143 0.000 0.000 0.143 0.000 0.571
1 2 3 4 5 6 7 8 9
Predicted Class
0.75
0.60
0.45
0.30
0.15
0.00

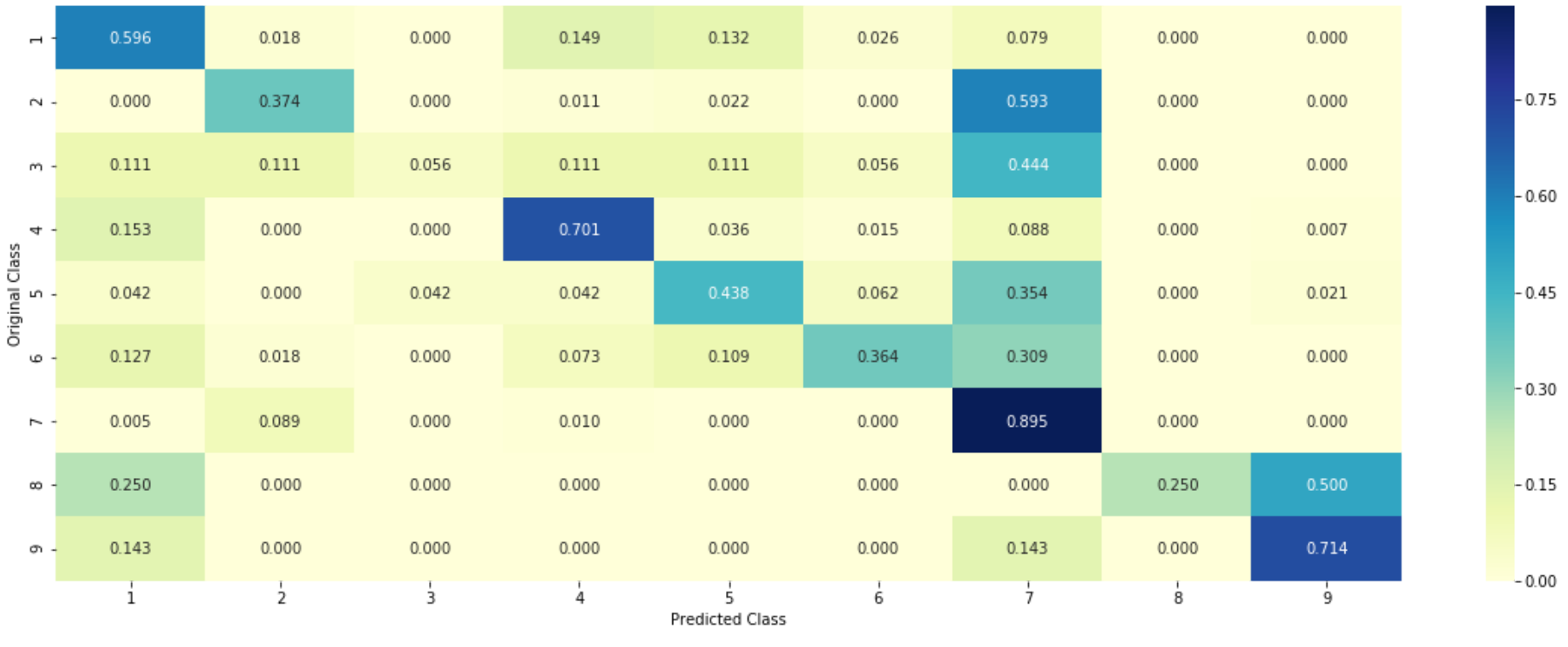
In [176]:
```
#Maximum Voting classifier
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.8419552910611676
Log loss (CV) on the VotingClassifier : 1.1833793832190374
Log loss (test) on the VotingClassifier : 1.1876046562251075
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```




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

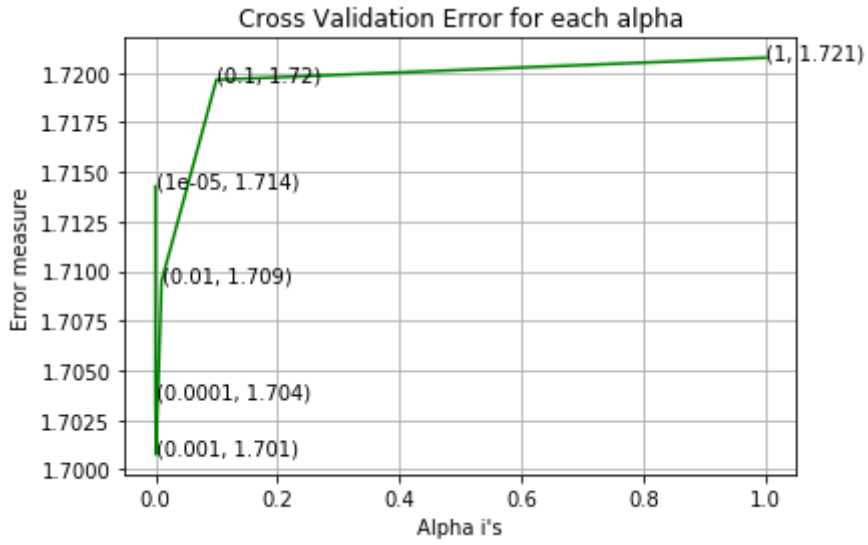

```
-------------------- Recall matrix (Row sum=1) --------------------
```



APPLYING LR WITH COUNTVECORIZER FEATURES INCLUDING UNIGRAMS & BIGRAMS

In [177]:
```
# Univariate Analysis on Variation Feature
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.714267082420089
For values of alpha =  0.0001 The log loss is: 1.7036434230177617
For values of alpha =  0.001 The log loss is: 1.7007619872095474
For values of alpha =  0.01 The log loss is: 1.7094933891604993
For values of alpha =  0.1 The log loss is: 1.7196512765864045
For values of alpha =  1 The log loss is: 1.7207625977363186
```




```
For values of best alpha =  0.001 The train log loss is: 1.0904906124055396
For values of best alpha =  0.001 The cross validation log loss is: 1.7007619872095474
For values of best alpha =  0.001 The test log loss is: 1.7063038942708804
```

In [178]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(ngram_range=(1, 2),min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = text_vectorizer.transform(test_df["TEXT"])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])

# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# final = train_text_feature_onehotCoding.toarray().sum(axis=0)
# top_1k_index = np.argsort(final)[::-1][0:1000]

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


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

```
Total number of unique words in train data : 786605
```

In [179]:
```
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 [180]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

# 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
# don't forget to normalize every feature
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
print("Number of words in train data = ", train_text_feature_onehotCoding.shape)
print("Number of words in test data = ", test_text_feature_onehotCoding.shape)
print("Number of words in cross validation data =", cv_text_feature_onehotCoding.shape)
```

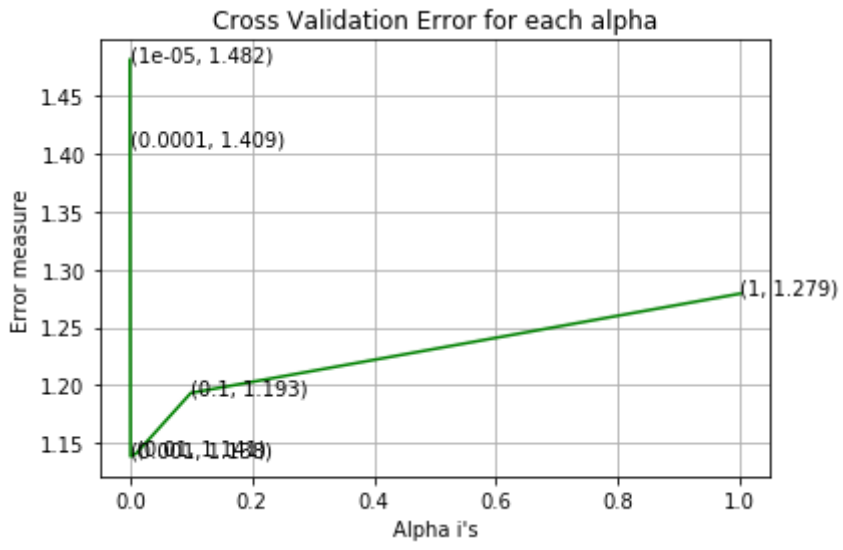
```
Number of words in train data =  (2124, 786605)
Number of words in test data =  (665, 786605)
Number of words in cross validation data = (532, 786605)
```

In [181]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))

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

```
Counter({3: 151006, 4: 116724, 5: 67111, 6: 57515, 7: 45778, 8: 42232, 9: 36638, 10: 23144, 12: 20888, 14: 17702, 11:
17332, 13: 15746, 15: 12787, 16: 11066, 18: 8499, 20: 7096, 17: 6838, 19: 6655, 24: 6068, 21: 5794, 25: 5047, 22: 434
2, 23: 3680, 26: 3572, 28: 3411, 27: 3285, 43: 3207, 34: 3180, 30: 2693, 29: 2470, 44: 2343, 32: 2256, 55: 2068, 31:
2026, 36: 1883, 33: 1850, 35: 1834, 38: 1498, 40: 1493, 37: 1473, 39: 1377, 42: 1322, 45: 1319, 75: 1295, 60: 1236, 4
8: 1224, 46: 1153, 41: 1139, 49: 953, 50: 945, 47: 932, 56: 898, 52: 862, 51: 834, 57: 779, 54: 763, 53: 754, 61: 66
7, 59: 662, 58: 660, 63: 628, 64: 593, 62: 587, 66: 540, 68: 534, 65: 526, 76: 520, 72: 517, 88: 492, 69: 450, 70: 44
7, 77: 445, 73: 442, 67: 441, 78: 439, 71: 424, 80: 375, 79: 361, 81: 352, 86: 349, 74: 346, 84: 345, 87: 340, 82: 32
8, 90: 320, 83: 313, 85: 305, 89: 304, 98: 297, 91: 275, 92: 273, 110: 270, 96: 257, 95: 248, 94: 247, 93: 243, 120:
216, 102: 215, 101: 214, 108: 212, 97: 209, 112: 209, 99: 207, 105: 200, 100: 196, 106: 193, 104: 191, 103: 189, 107:
183, 113: 172, 114: 170, 111: 165, 109: 163, 121: 161, 117: 159, 128: 151, 122: 150, 115: 149, 132: 147, 126: 144, 11
8: 143, 129: 143, 124: 142, 125: 142, 123: 137, 131: 132, 130: 131, 116: 127, 119: 127, 136: 123, 133: 123, 146: 119,
137: 118, 150: 118, 127: 116, 134: 114, 143: 114, 145: 114, 141: 113, 138: 110, 144: 109, 135: 108, 139: 108, 140: 10
7, 152: 105, 165: 105, 149: 101, 151: 98, 142: 97, 148: 96, 164: 90, 163: 88, 156: 87, 155: 86, 159: 86, 170: 85, 15
4: 82, 157: 81, 168: 81, 172: 80, 160: 79, 176: 79, 147: 78, 162: 77, 169: 77, 153: 76, 161: 75, 180: 75, 166: 73, 17
5: 73, 158: 72, 174: 72, 167: 71, 177: 69, 178: 68, 173: 67, 186: 66, 192: 66, 197: 63, 181: 62, 188: 62, 171: 61, 18
2: 61, 183: 59, 185: 59, 189: 59, 190: 58, 200: 58, 201: 58, 184: 56, 198: 56, 203: 56, 219: 55, 179: 54, 191: 54, 19
3: 53, 209: 53, 232: 53, 237: 53, 187: 51, 196: 51, 204: 51, 213: 51, 220: 51, 215: 50, 245: 50, 275: 50, 224: 49, 21
6: 48, 218: 47, 243: 47, 194: 46, 195: 46, 212: 46, 233: 46, 205: 45, 207: 45, 217: 45, 221: 45, 229: 45, 249: 45, 19
9: 44, 206: 44, 230: 44, 231: 43, 223: 42, 226: 42, 266: 42, 202: 41, 225: 41, 228: 40, 235: 40, 210: 39, 222: 39, 24
```

In [182]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.


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

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.4816175737897068
For values of alpha =  0.0001 The log loss is: 1.408611622138659
For values of alpha =  0.001 The log loss is: 1.1382302199905177
For values of alpha =  0.01 The log loss is: 1.140734667468016
For values of alpha =  0.1 The log loss is: 1.1932241031339472
For values of alpha =  1 The log loss is: 1.2791612554158605
```




```
For values of best alpha =  0.001 The train log loss is: 0.8181996713428853
For values of best alpha =  0.001 The cross validation log loss is: 1.1382302199905177
For values of best alpha =  0.001 The test log loss is: 1.2383244442196224
```

In [183]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(ngram_range=(1, 2),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

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

```
91.708 % of word of test data appeared in train data
95.199 % of word of Cross Validation appeared in train data
```

ML MODELS:

In [184]:
```
#Data preparation for ML models.

#Misc. functionns for ML models

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

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

In [185]:
```
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)

def get_impfeature_names(indices, text, gene, var, no_features,gene_vec,var_vec,text_vec):
    gene_vec = gene_vec.fit(train_df['Gene'])
    var_vec  = var_vec.fit(train_df['Variation'])
    text_vec = text_vec.fit(train_df['TEXT'])

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

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

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

In [186]:
```
#Stacking the three types of features
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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

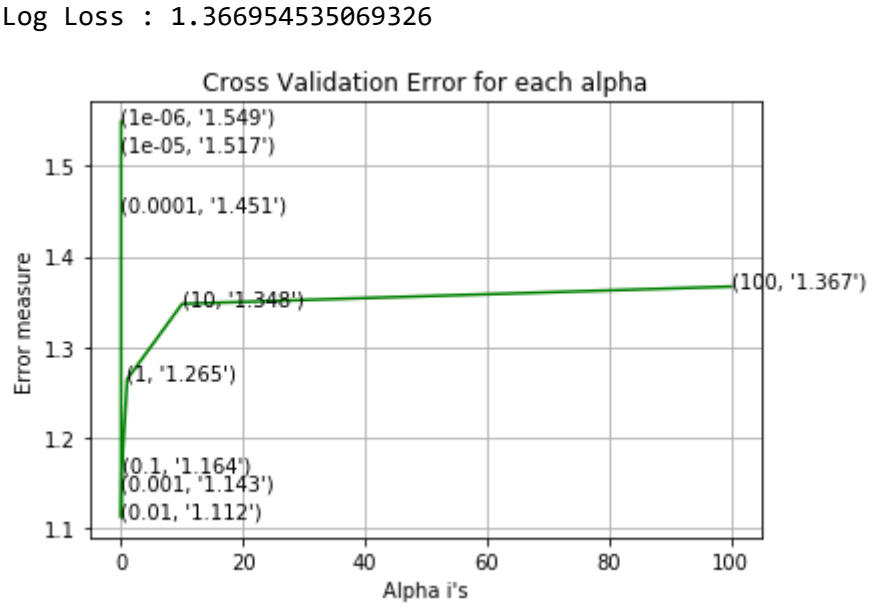
In [187]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 788785)
(number of data points * number of features) in test data =  (665, 788785)
(number of data points * number of features) in cross validation data = (532, 788785)
```

Logistic Regression:

In [188]:
```
#With Class balancing
###Hyper paramter tuning

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
for alpha = 1e-06
Log Loss : 1.5487297286395418
for alpha = 1e-05
Log Loss : 1.5166781141794752
for alpha = 0.0001
Log Loss : 1.4513088402635568
for alpha = 0.001
Log Loss : 1.1426481176759744
for alpha = 0.01
Log Loss : 1.1122477823437267
for alpha = 0.1
Log Loss : 1.1636317024096563
for alpha = 1
Log Loss : 1.2648545201965027
for alpha = 10
Log Loss : 1.3479310576659445
for alpha = 100
Log Loss : 1.366954535069326
```



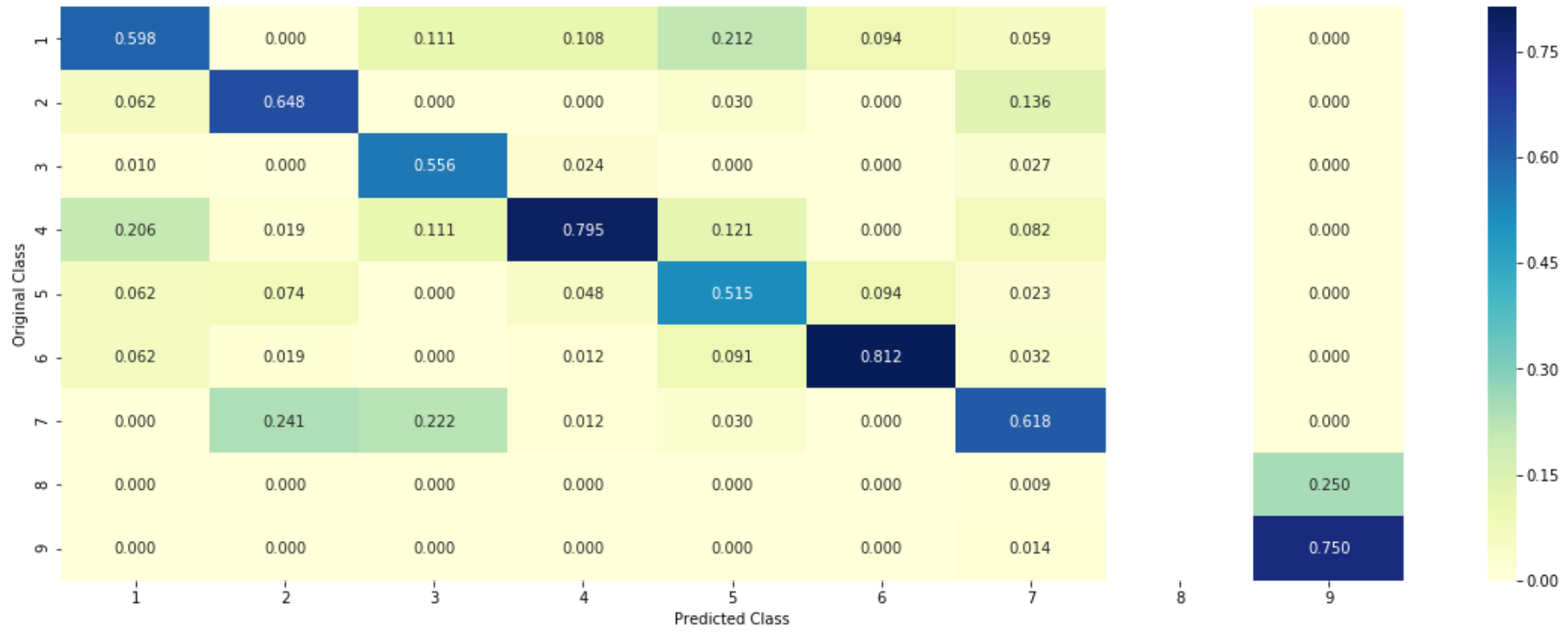

```
For values of best alpha =  0.01 The train log loss is: 0.7211447010120651
For values of best alpha =  0.01 The cross validation log loss is: 1.1122477823437267
For values of best alpha =  0.01 The test log loss is: 1.1950816589795243
```

In [189]:
```
#Testing the model with best hyper paramters
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 : 1.1122477823437267
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



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




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

Correctly classified

In [191]:
```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3643 0.028  0.025  0.173  0.3634 0.0112 0.0229 0.0065 0.0056]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

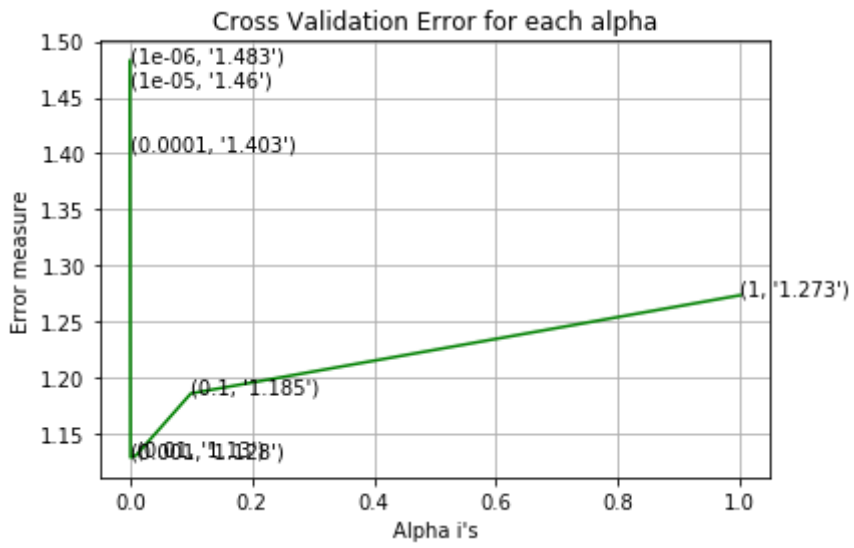
incorrectly classified

In [195]:
```
test_point_index = 5
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.195  0.1532 0.0238 0.1694 0.0731 0.0785 0.2902 0.0043 0.0124]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

In [196]:
```
#Without Balancing Hyper Parameter Tuning
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

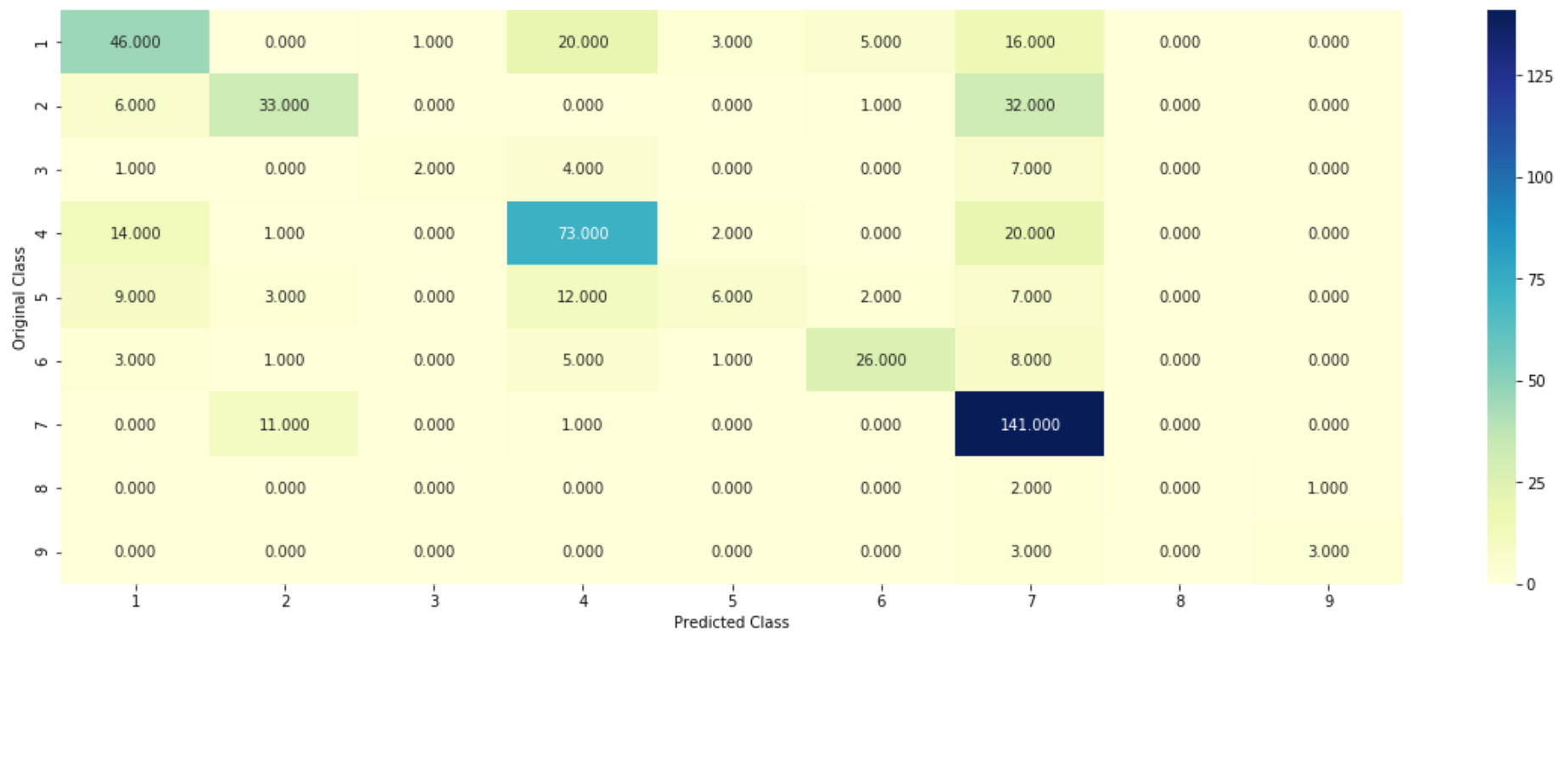
```
for alpha = 1e-06
Log Loss : 1.482762377047317
for alpha = 1e-05
Log Loss : 1.4604217266408148
for alpha = 0.0001
Log Loss : 1.4028550103125752
for alpha = 0.001
Log Loss : 1.1280746473709116
for alpha = 0.01
Log Loss : 1.1299098046269231
for alpha = 0.1
Log Loss : 1.1853258579819623
for alpha = 1
Log Loss : 1.2731070670750464
```



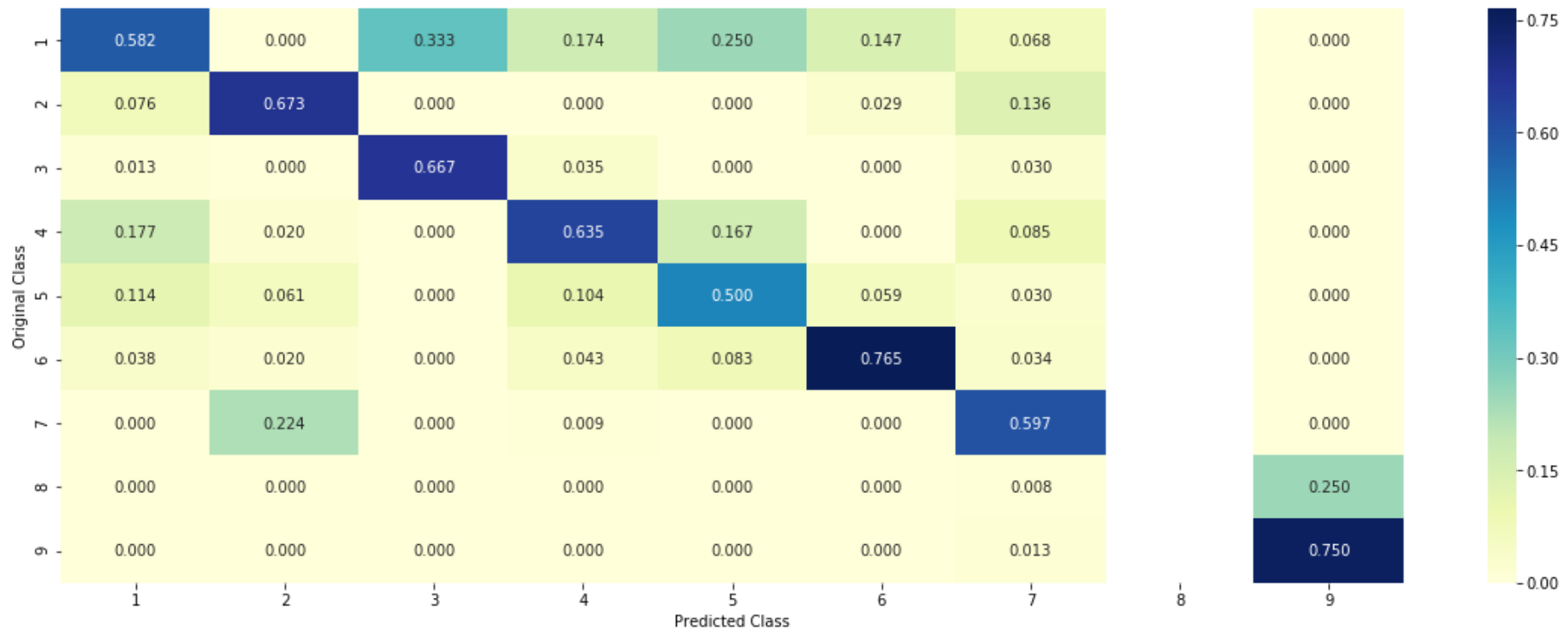

```
For values of best alpha =  0.001 The train log loss is: 0.7585039024232757
For values of best alpha =  0.001 The cross validation log loss is: 1.1280746473709116
For values of best alpha =  0.001 The test log loss is: 1.2172985897034123
```

In [197]:
```
#Testing model with best hyper parameters
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)
```

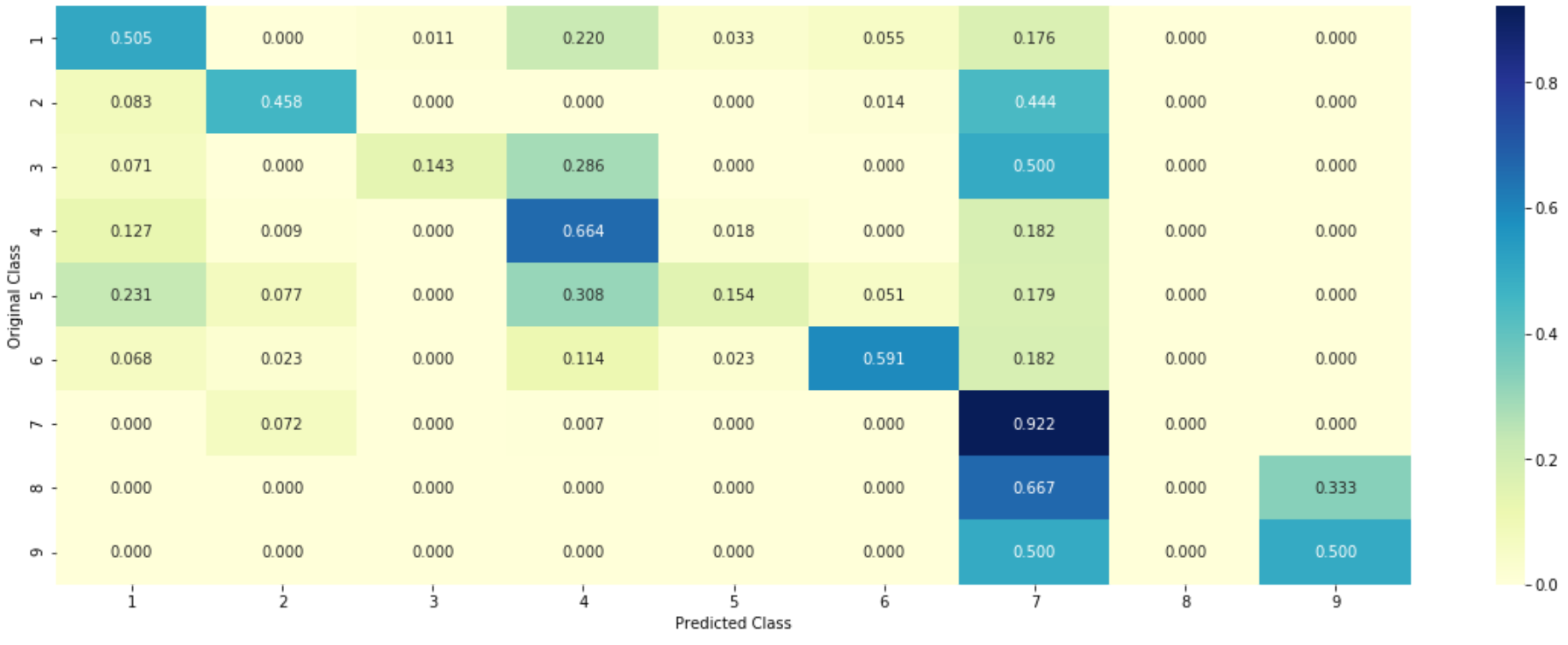
```
Log loss : 1.1280746473709116
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```




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




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



COrrectly classified

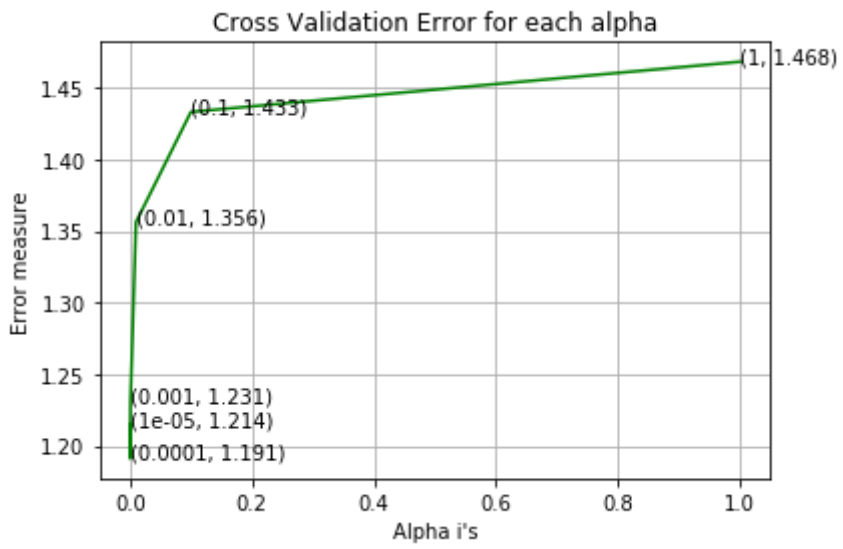
In [198]:
```
test_point_index = 1
no_feature = 200
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.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['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4047 0.051  0.0233 0.2567 0.1898 0.0195 0.0437 0.005  0.0062]]
Actual Class : 1
--------------------------------------------------
Out of the top  200  features  0 are present in query point
```

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

In [199]:
```
# hyperparam for SGD classifier.

alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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

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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.2143176195070209
For values of alpha =  0.0001 The log loss is: 1.1911127561743882
For values of alpha =  0.001 The log loss is: 1.2311929583261052
For values of alpha =  0.01 The log loss is: 1.3561045424768787
For values of alpha =  0.1 The log loss is: 1.4330643710636428
For values of alpha =  1 The log loss is: 1.4681123457028933
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0036741633733037
For values of best alpha =  0.0001 The cross validation log loss is: 1.1911127561743882
For values of best alpha =  0.0001 The test log loss is: 1.1738995614628518
```

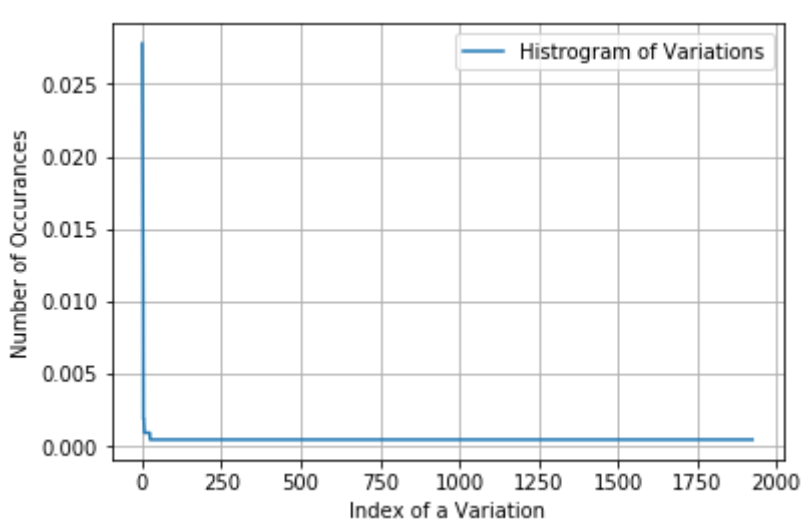
In [200]:
```
print("How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in train datase

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

```
How many data points in Test and CV datasets are covered by the  229  genes in train dataset?
 In test data 639 out of 665 : 96.09022556390977
 In cross validation data 509 out of  532 : 95.67669172932331
```

In [201]:
```
#Univariate Analysis on Variation Feature
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))
```

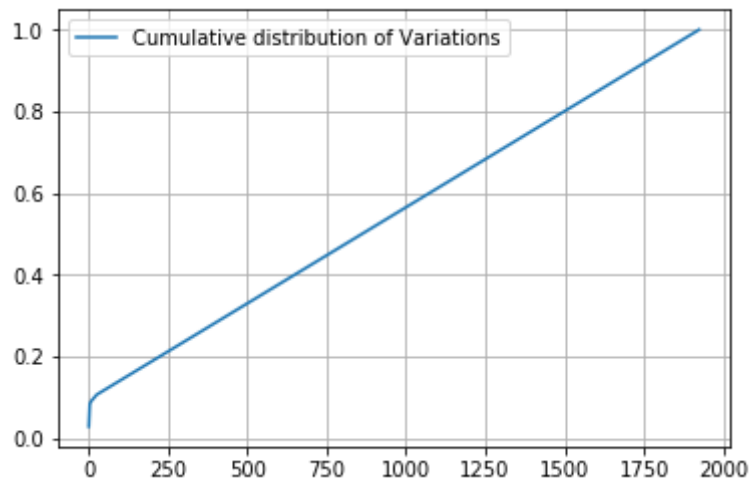
```
Number of Unique Variations : 1925
Truncating_Mutations    59
Deletion                48
Amplification           44
Fusions                 26
G12V                     4
Overexpression           4
T58I                     3
EWSR1-ETV1_Fusion        2
G12A                     2
T167A                    2
Name: Variation, dtype: int64
```

In [202]:
```
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 [203]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02777778 0.05037665 0.07109228 ... 0.99905838 0.99952919 1.        ]
```



In [204]:
```
#FEATURAIZATION
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [206]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Vari
```

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

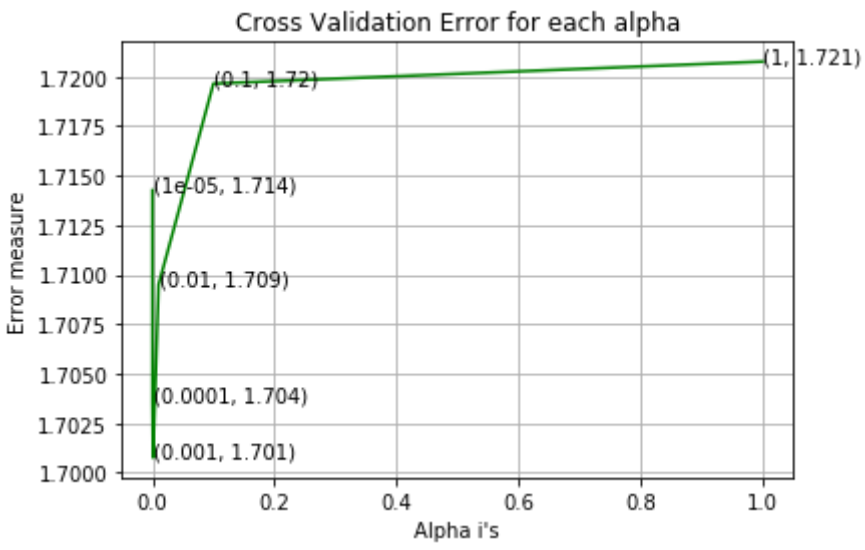
In [205]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [208]:
```
train_variation_feature_onehotCoding.shape
```

Out[208]:
```
(2124, 1951)
```

In [209]:
```
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.714267082420089
For values of alpha =  0.0001 The log loss is: 1.7036434230177617
For values of alpha =  0.001 The log loss is: 1.7007619872095474
For values of alpha =  0.01 The log loss is: 1.7094933891604993
For values of alpha =  0.1 The log loss is: 1.7196512765864045
For values of alpha =  1 The log loss is: 1.7207625977363186
```




```
For values of best alpha =  0.001 The train log loss is: 1.0904906124055396
For values of best alpha =  0.001 The cross validation log loss is: 1.7007619872095474
For values of best alpha =  0.001 The test log loss is: 1.7063038942708804
```

In [210]:
```
print("How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation dat
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('In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*100)
print('In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
How many data points are covered by total  1925  genes in test and cross validation data sets?
In test data 67 out of 665 : 10.075187969924812
In cross validation data 54 out of  532 : 10.150375939849624
```

building a TfidfVectorizer with ngram range(1,4) and max features 2000

In [212]:
```
text_vectorizer = TfidfVectorizer(ngram_range=(1,4),max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = text_vectorizer.transform(test_df["TEXT"])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])

# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

final = train_text_feature_onehotCoding.toarray().sum(axis=0)
top_1k_index = np.argsort(final)[::-1][0:1000]

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


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

```
Total number of unique words in train data : 2000
```

In [213]:
```
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 [214]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)

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

train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding[:,top_1k_index], axis=0)
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding[:,top_1k_index], axis=0)
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding[:,top_1k_index], axis=0)
```

In [215]:
```
print("Number of words in train data = ", train_text_feature_onehotCoding.shape)
print("Number of words in test data = ", test_text_feature_onehotCoding.shape)
print("Number of words in cross validation data =", cv_text_feature_onehotCoding.shape)
```

```
Number of words in train data =  (2124, 1000)
Number of words in test data =  (665, 1000)
Number of words in cross validation data = (532, 1000)
```

In [218]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
```

In [219]:
```
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

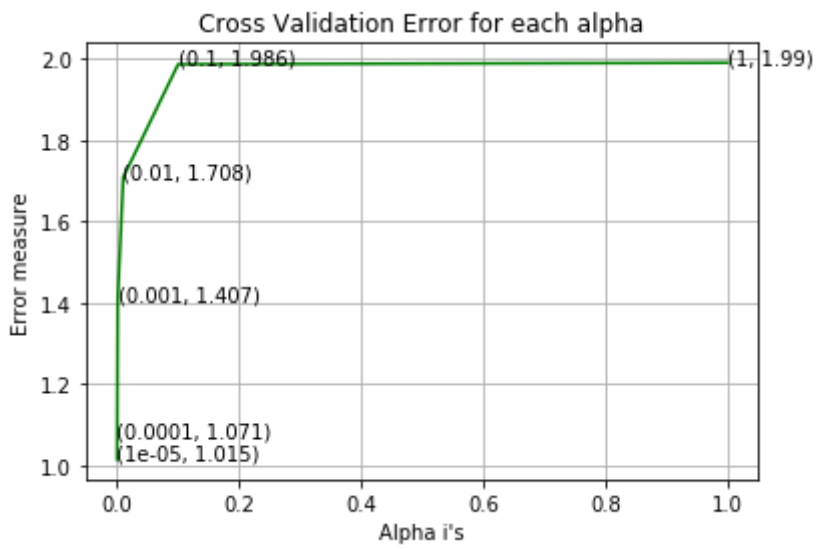
In [220]:
```
freq_dict=Counter(sorted_text_occur)
```

In [221]:
```
freq_dict
```

Out[221]:
```
Counter({4.160608239251734: 1,
         3.7277284805456947: 1,
         4.68037889789891: 1,
         5.883883407184095: 1,
         6.428139405037163: 1,
         7.793116363110858: 1,
         8.136851102026036: 1,
         9.222156380270588: 1,
         10.507996621998487: 1,
         11.242701488667079: 1,
         12.106844078302043: 1,
         13.131438034354948: 1,
         14.447424681513924: 1,
         15.684979538387: 1,
         16.646498333532975: 1,
         17.92030473610936: 1,
         18.12353054074064: 1,
         19.436544260794538: 1,
         20.47578881138638: 1,
```

In [222]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, y_train)

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

```
For values of alpha =  1e-05 The log loss is: 1.0150927057889856
For values of alpha =  0.0001 The log loss is: 1.0713946723738128
For values of alpha =  0.001 The log loss is: 1.4068430273057335
For values of alpha =  0.01 The log loss is: 1.7084433824187129
For values of alpha =  0.1 The log loss is: 1.9864612973776532
For values of alpha =  1 The log loss is: 1.9903128576849354
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7416981533130199
For values of best alpha =  1e-05 The cross validation log loss is: 1.0150927057889856
For values of best alpha =  1e-05 The test log loss is: 1.1172653310999838
```

In [223]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
0.598 % of word of test data appeared in train data
0.84 % of word of Cross Validation appeared in train data
```

ML Models

In [224]:
```
#Stacking the three types of features
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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

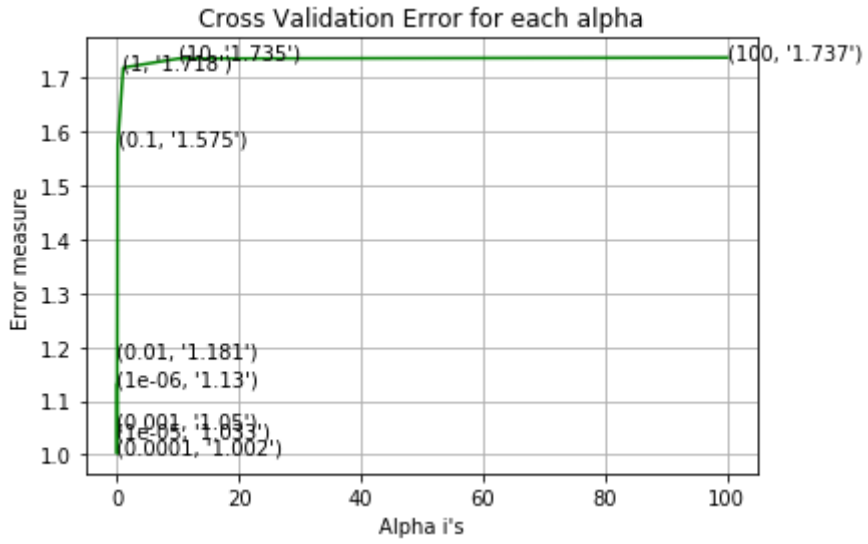
In [225]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
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)
```

```
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)
 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)
```

Logistic Regression:

In [226]:
```
#With Class balancing
####Hyper paramter tuning
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, la
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.c
```

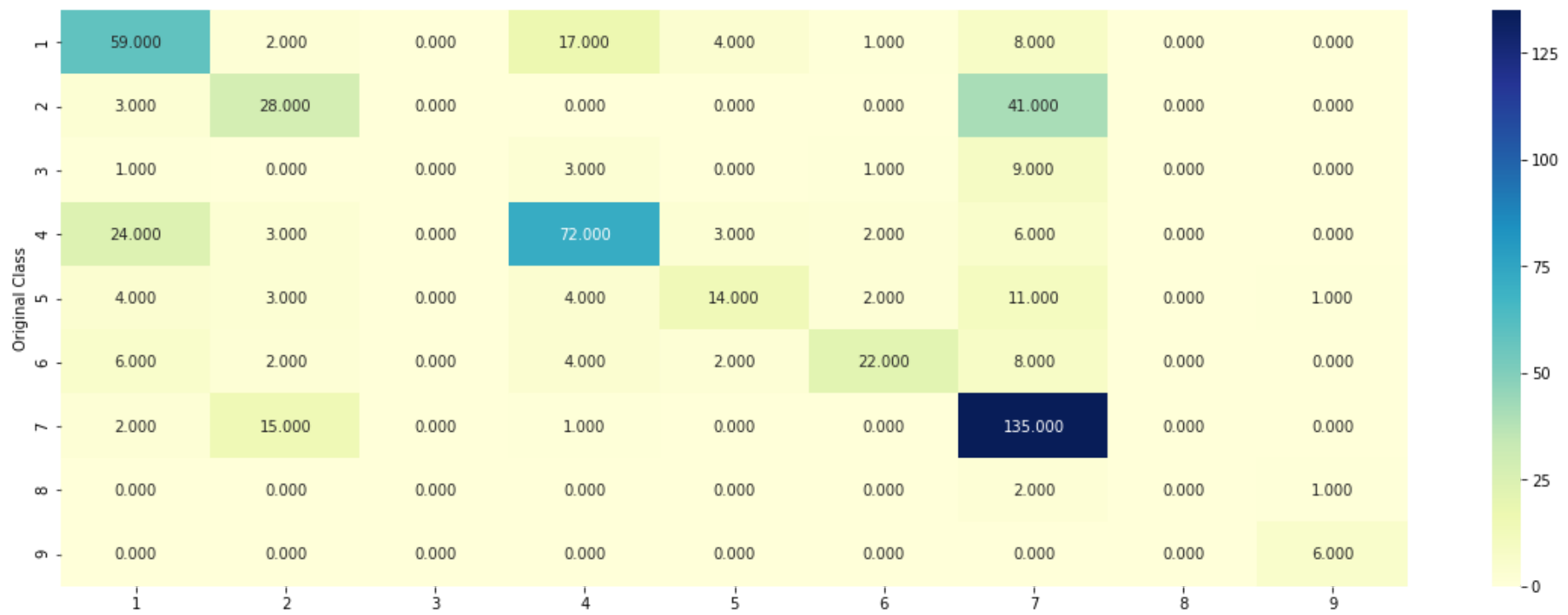
```
for alpha = 1e-06
Log Loss : 1.1302712419350265
for alpha = 1e-05
Log Loss : 1.033491679374652
for alpha = 0.0001
Log Loss : 1.0020514513985133
for alpha = 0.001
Log Loss : 1.0501083842618837
for alpha = 0.01
Log Loss : 1.1807531120019474
for alpha = 0.1
Log Loss : 1.5750492908735039
for alpha = 1
Log Loss : 1.71820039486223
for alpha = 10
Log Loss : 1.7349591944737266
for alpha = 100
Log Loss : 1.7368776102792969
```



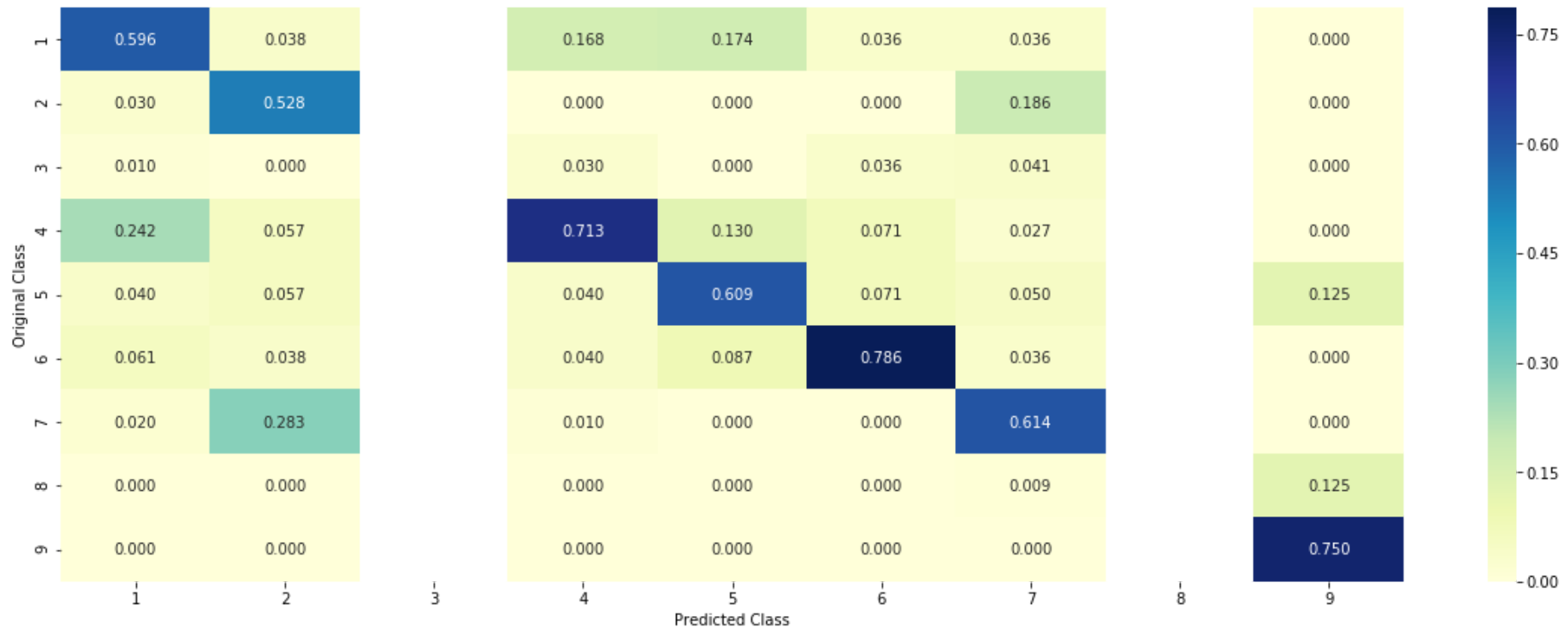

```
For values of best alpha =  0.0001 The train log loss is: 0.4600394143665047
For values of best alpha =  0.0001 The cross validation log loss is: 1.0020514513985133
For values of best alpha =  0.0001 The test log loss is: 0.9941297427991787
```

In [227]:
```
#Testing the model with best hyper paramters
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)
```

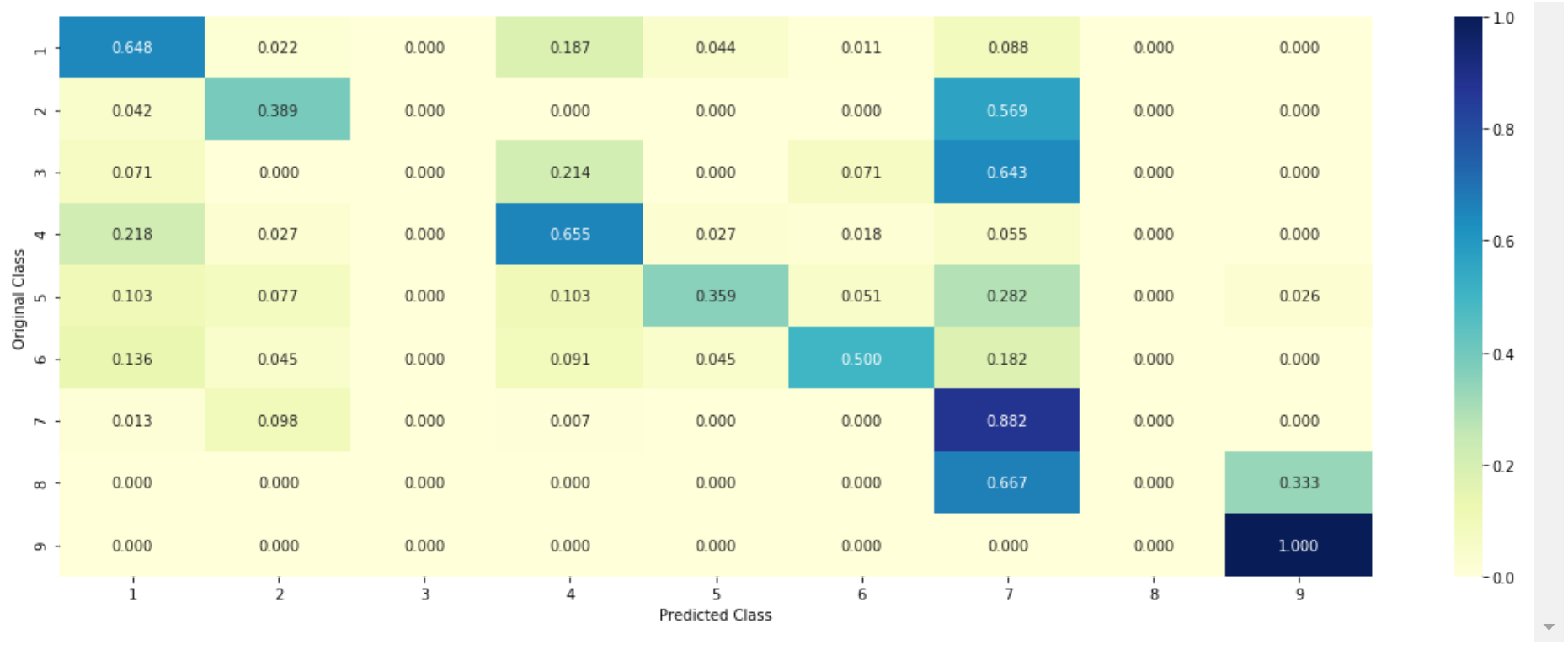
```
Log loss : 1.0020514513985133
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```




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




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



Correctly Classified point

In [ ]:

In [230]:
```
# 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 = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0177 0.0134 0.0095 0.0081 0.0166 0.0151 0.9164 0.0014 0.0017]]
Actual Class : 7
--------------------------------------------------
```

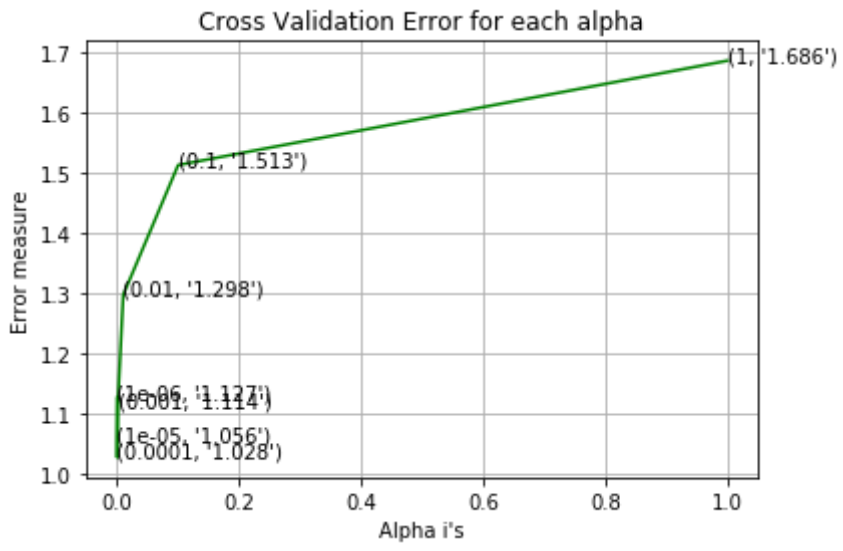
Incorrectly classified point s

In [231]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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
```

```
Predicted Class : 5
Predicted Class Probabilities: [[3.314e-01 4.100e-03 1.500e-03 1.826e-01 4.759e-01 2.700e-03 7.000e-04
  8.000e-04 3.000e-04]]
Actual Class : 1
--------------------------------------------------
```

In [232]:
```
g without Class balancing
x in range(-6, 1)]
[]

=", i)
er(alpha=i, penalty='l2', loss='log', random_state=42)
nehotCoding, train_y)
tedClassifierCV(clf, method="sigmoid")
_x_onehotCoding, train_y)
ig_clf.predict_proba(cv_x_onehotCoding)
y.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
",log_loss(cv_y, sig_clf_probs))

s()
_error_array,c='g')
te(np.round(cv_log_error_array,3)):
a[i],str(txt)), (alpha[i],cv_log_error_array[i]))

dation Error for each alpha")
")
sure")



n(cv_log_error_array)
lpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
tCoding, train_y)
lassifierCV(clf, method="sigmoid")
nehotCoding, train_y)

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

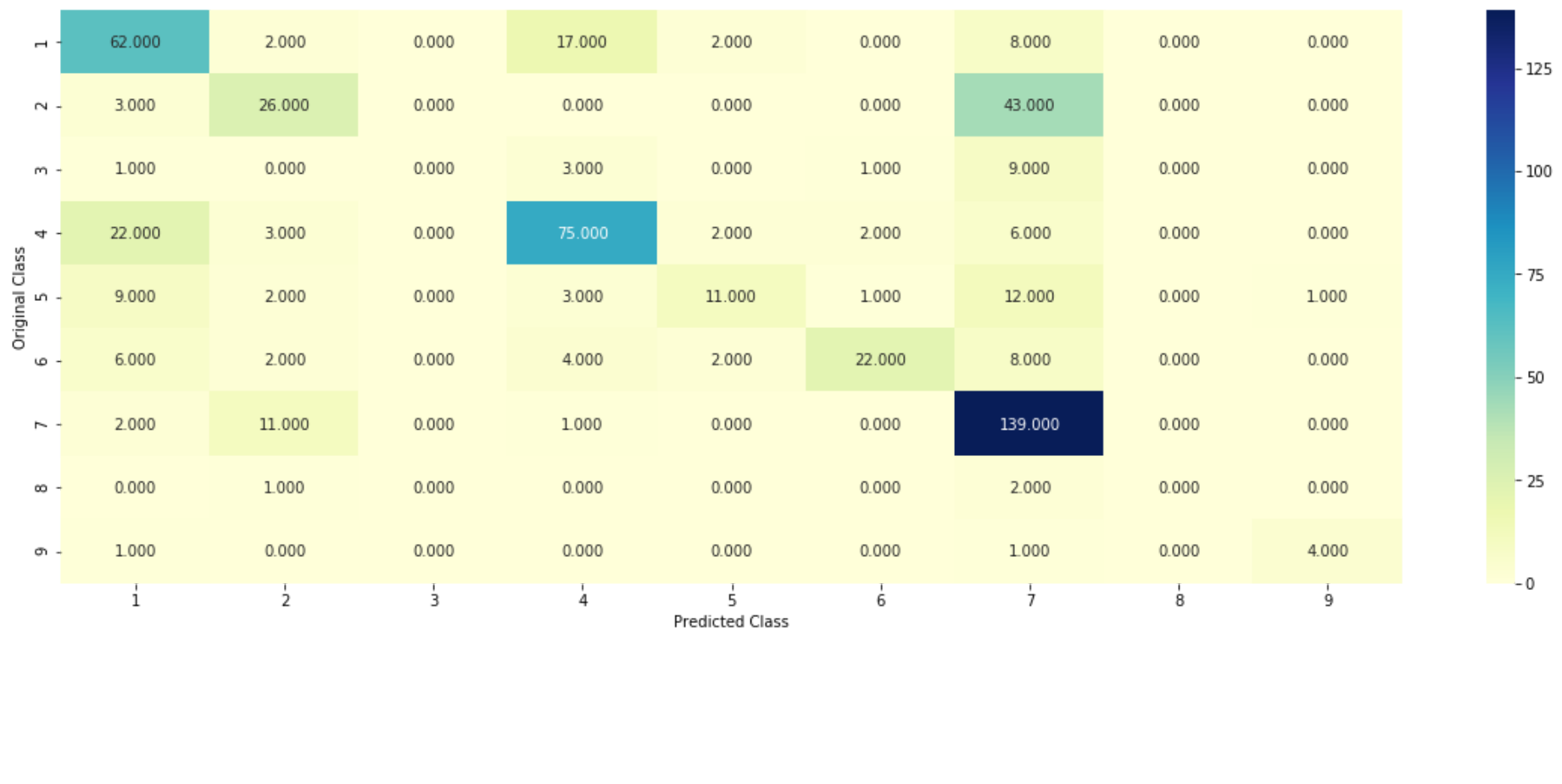
```
for alpha = 1e-06
Log Loss : 1.126791996175152
for alpha = 1e-05
Log Loss : 1.0555556117124851
for alpha = 0.0001
Log Loss : 1.0281926468722726
for alpha = 0.001
Log Loss : 1.1137250841925925
for alpha = 0.01
Log Loss : 1.2976075086341676
for alpha = 0.1
Log Loss : 1.512807968768255
for alpha = 1
Log Loss : 1.6857805848257632
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44640707409959784
For values of best alpha =  0.0001 The cross validation log loss is: 1.0281926468722726
For values of best alpha =  0.0001 The test log loss is: 1.014254354180853
```

In [233]:
```
#Testing model with best hyper parameters
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 : 1.0281926468722726
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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


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

In [236]:
```
#Feature Importance, Correctly Classified point
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.700e-02 1.310e-02 7.600e-03 8.500e-03 1.550e-02 1.480e-02 9.213e-01
  9.000e-04 1.100e-03]]
Actual Class : 7
--------------------------------------------------
```

In [238]:
```
#Feature Importance, inCorrectly Classified point
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[4.082e-01 4.200e-03 1.300e-03 1.453e-01 4.371e-01 2.700e-03 1.000e-03
  1.000e-04 1.000e-04]]
Actual Class : 1
--------------------------------------------------
```

Performance Comparison of Models

In [4]:
```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

model =['Naive Bayes','K-Nearest Neighbour','LR With Class Balancing','LR Without Class Balancing','Linear SVM','RF With


train = [0.938 , 0.492 , 0.558 , 0.547 , 0.601 , 0.678 , 0.0526 , 0.645 , 0.865]

cv_loss = [1.231 , 1.018 , 0.984 , 1.006 , 1.023 , 1.183 , 1.407, 1.077 , 1.107]

test = [1.22 , 1.047 , 1.049 , 1.0608 , 1.113 , 1.166 , 1.387 , 1.109 , 1.163]

misclass = [42.2 , 33.8 , 33.4 , 34.3 , 33.6 , 38.7 , 53.3, 36.8 , 36.4]

s_no = [1,2,3,4,5,6,7,8,9]
table = PrettyTable()

# Adding columns
table.add_column("s.no.",s_no)
table.add_column("model",model)
table.add_column("training loss",train)
table.add_column("cv loss",cv_loss)
table.add_column("test loss",test)
table.add_column("Missclassification (%)",misclass)

print(table)
```

```
+-------+---------------------------+---------------+---------+-----------+------------------------+
| s.no. |           model           | training loss | cv loss | test loss | Missclassification (%) |
+-------+---------------------------+---------------+---------+-----------+------------------------+
|   1   |        Naive Bayes        |     0.938     |  1.231  |    1.22   |          42.2          |
|   2   |    K-Nearest Neighbour    |     0.492     |  1.018  |   1.047   |          33.8          |
|   3   |  LR With Class Balancing  |     0.558     |  0.984  |   1.049   |          33.4          |
|   4   | LR Without Class Balancing |     0.547     |  1.006  |   1.0608  |          34.3          |
|   5   |         Linear SVM        |     0.601     |  1.023  |   1.113   |          33.6          |
|   6   |  RF With One hot Encoding |     0.678     |  1.183  |   1.166   |          38.7          |
|   7   |  RF With Response Coding  |     0.0526    |  1.407  |   1.387   |          53.3          |
|   8   |    Stacking Classifier    |     0.645     |  1.077  |   1.109   |          36.8          |
|   9   | Maximum Voting Classifier |     0.865     |  1.107  |   1.163   |          36.4          |
+-------+---------------------------+---------------+---------+-----------+------------------------+
```

performance table for models with top 1000 features

In [8]:
```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

model =['Naive Bayes','K-Nearest Neighbour','LR With Class Balancing','LR Without Class Balancing','Linear SVM','RF With


train = [0.534 , 0.492 , 0.450 , 0.438 , 0.568 , 0.584 , 0.0526 , 0.565 , 0.841]

cv_loss = [1.209 , 1.018 , 1.000 , 1.032 , 1.036 , 1.253 , 1.407, 1.192 , 1.183]

test = [1.209 , 1.047 , 1.001, 1.015 , 1.068 , 1.166 , 1.387 , 1.173 , 1.187]

misclass = [38.7 , 33.8 , 35.1 , 35.3 , 35.5 , 39.4 , 53.3, 38.0 , 37.2]

s_no = [1,2,3,4,5,6,7,8,9]
table = PrettyTable()

# Adding columns
table.add_column("s.no.",s_no)
table.add_column("model",model)
table.add_column("training loss",train)
table.add_column("cv loss",cv_loss)
table.add_column("test loss",test)
table.add_column("Missclassification (%)",misclass)
print(table)
```

```
+-------+---------------------------+---------------+---------+-----------+------------------------+
| s.no. |           model           | training loss | cv loss | test loss | Missclassification (%) |
+-------+---------------------------+---------------+---------+-----------+------------------------+
|   1   |        Naive Bayes        |     0.534     |  1.209  |   1.209   |          38.7          |
|   2   |    K-Nearest Neighbour    |     0.492     |  1.018  |   1.047   |          33.8          |
|   3   |  LR With Class Balancing  |      0.45     |   1.0   |   1.001   |          35.1          |
|   4   | LR Without Class Balancing |     0.438     |  1.032  |   1.015   |          35.3          |
|   5   |        Linear SVM         |     0.568     |  1.036  |   1.068   |          35.5          |
|   6   |  RF With One hot Encoding |     0.584     |  1.253  |   1.166   |          39.4          |
|   7   |  RF With Response Coding  |     0.0526    |  1.407  |   1.387   |          53.3          |
|   8   |    Stacking Classifier    |     0.565     |  1.192  |   1.173   |          38.0          |
|   9   | Maximum Voting Classifier |     0.841     |  1.183  |   1.187   |          37.2          |
+-------+---------------------------+---------------+---------+-----------+------------------------+
```

performance model for "APPLYING LR WITH COUNTVECORIZER FEATURES INCLUDING UNIGRAMS & BIGRAMS"

In [11]:
```
# Names of models
model =['LR- Class Balancing','LR -non Class Balancing']

train  = [0.721,0.758]

cv = [1.112,1.128]

test = [1.195,1.217]

# Percentage Misclassified points
misclassification = [34.9,37.9]

s_no = [1,2]
# Initializing prettytable
table = PrettyTable()
# Adding columns
table.add_column("ser no.",s_no)
table.add_column("models",model)
table.add_column("training loss",train)
table.add_column("CV loss",cv)
table.add_column("test loss",test)
table.add_column("Missclassification(%)",misclassification)
# Printing the Table
print(table)
```

```
+---------+-------------------------+---------------+---------+-----------+-----------------------+
| ser no. |          models         | training loss | CV loss | test loss | Missclassification(%) |
+---------+-------------------------+---------------+---------+-----------+-----------------------+
|    1    |   LR- Class Balancing   |     0.721     |  1.112  |   1.195   |          34.9         |
|    2    | LR -non Class Balancing |     0.758     |  1.128  |   1.217   |          37.9         |
+---------+-------------------------+---------------+---------+-----------+-----------------------+
```

performance comparision for models with FE

In [13]:
```
# Names of models
model =['LR- Class Balancing','LR -non Class Balancing']

train  = [0.460,0.446]

cv = [1.002,1.028]

test = [0.994,1.014]

# Percentage Misclassified points
misclassification = [36.8,36.2]

s_no = [1,2]
# Initializing prettytable
table = PrettyTable()
# Adding columns
table.add_column("ser no.",s_no)
table.add_column("models",model)
table.add_column("training loss",train)
table.add_column("CV loss",cv)
table.add_column("test loss",test)
table.add_column("Missclassification(%)",misclassification)
# Printing the Table
print(table)
```

```
+---------+-------------------------+---------------+---------+-----------+-----------------------+
| ser no. |          models         | training loss | CV loss | test loss | Missclassification(%) |
+---------+-------------------------+---------------+---------+-----------+-----------------------+
|    1    |   LR- Class Balancing   |      0.46     |  1.002  |   0.994   |          36.8         |
|    2    | LR -non Class Balancing |     0.446     |  1.028  |   1.014   |          36.2         |
+---------+-------------------------+---------------+---------+-----------+-----------------------+
```