Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.

- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [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.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
# from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression

from tqdm import tqdm_notebook
from sklearn.feature_selection import SelectKBest
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv('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[2]:

	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 [3]:

```
# note the seprator in this file
data_text =pd.read_csv("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[3]:

	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 [4]:

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))

def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""

        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)

        # converting all the chars into lower-case.
        total_text = total_text.lower()

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

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

In [5]:

```
#text processing stage.
start_time = time.clock()
for index, row in tqdm_notebook(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 : 225.40548280000002 seconds
```

In [6]:

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

Out[6]:

	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 [7]:

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

Out[7]:

	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 [8]:

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

In [9]:

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

Out[9]:

	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 [10]:

```
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' [stra
tify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2, ra
ndom_state=1)

# split the train data into train and cross validation by maintaining same distribution of output varai
ble 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2, ra
ndom_state=1)
```

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

In [11]:

```
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])
```

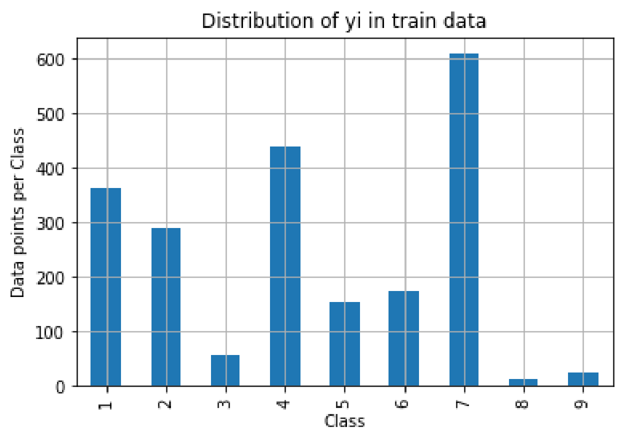
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

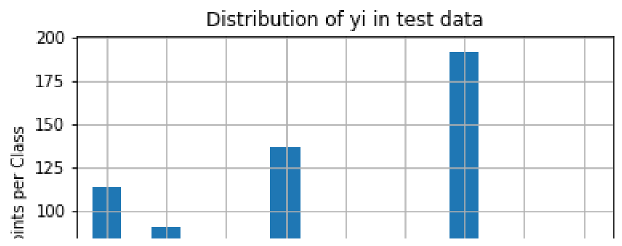
In [12]:

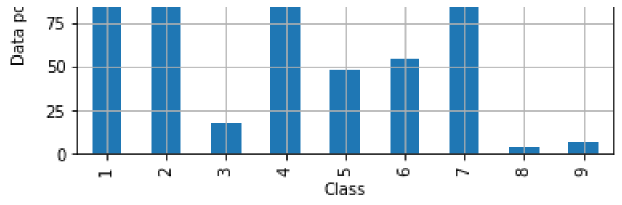
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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

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


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

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

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

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

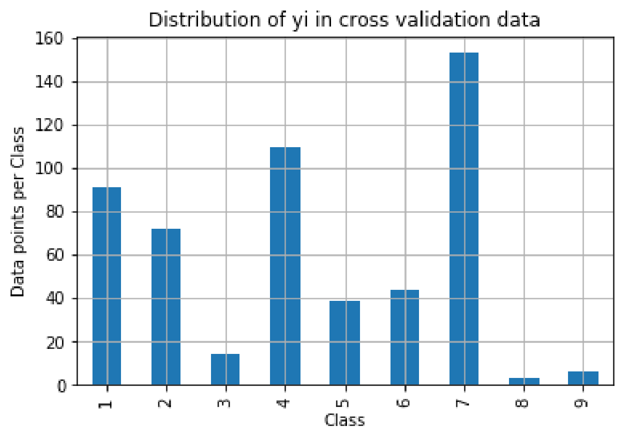


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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





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



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

3.2 Prediction using a 'Random' Model

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

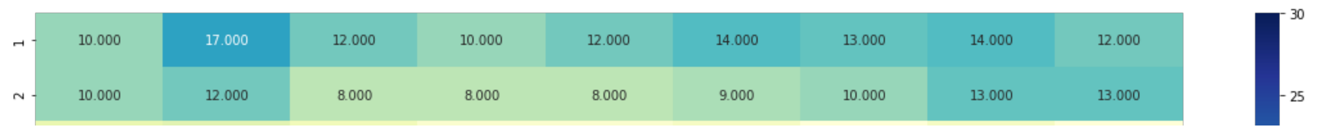
In [13]:

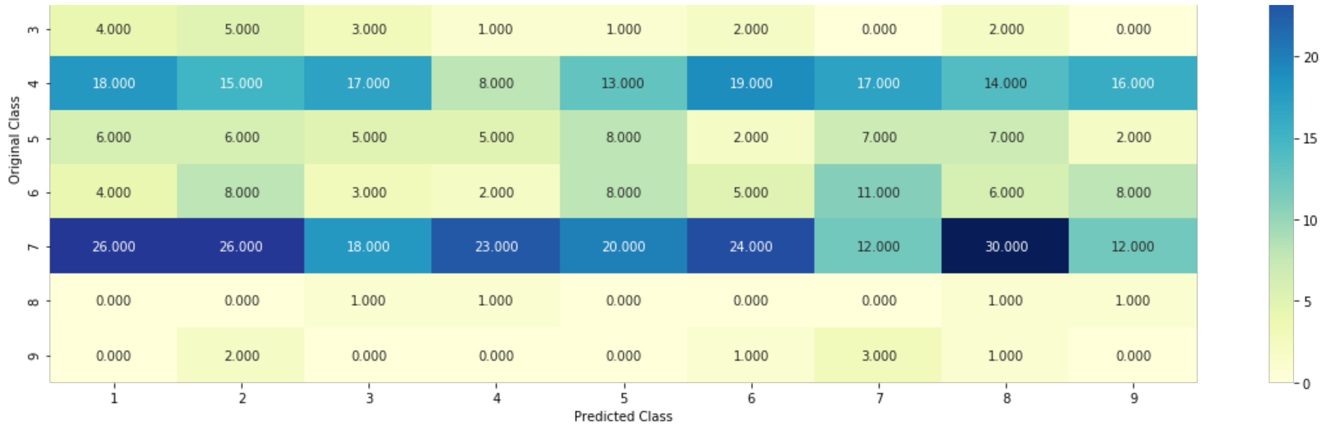
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

In [14]:

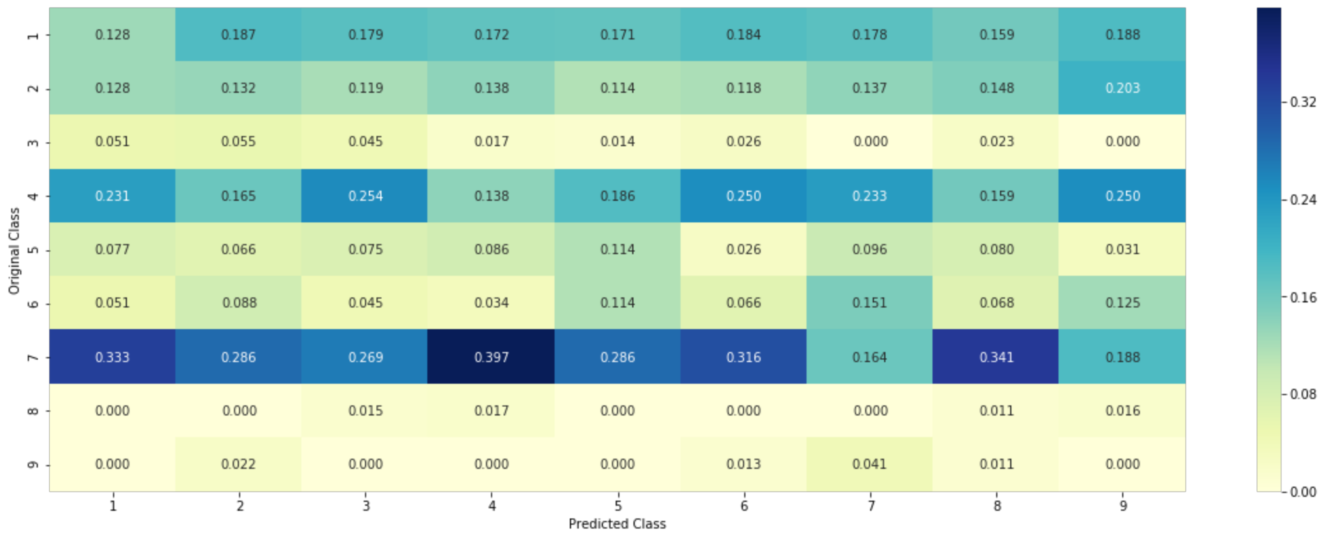
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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 2.527037114414166
Log loss on Test Data using Random Model 2.5714776111177153
-------------------- Confusion matrix --------------------
```

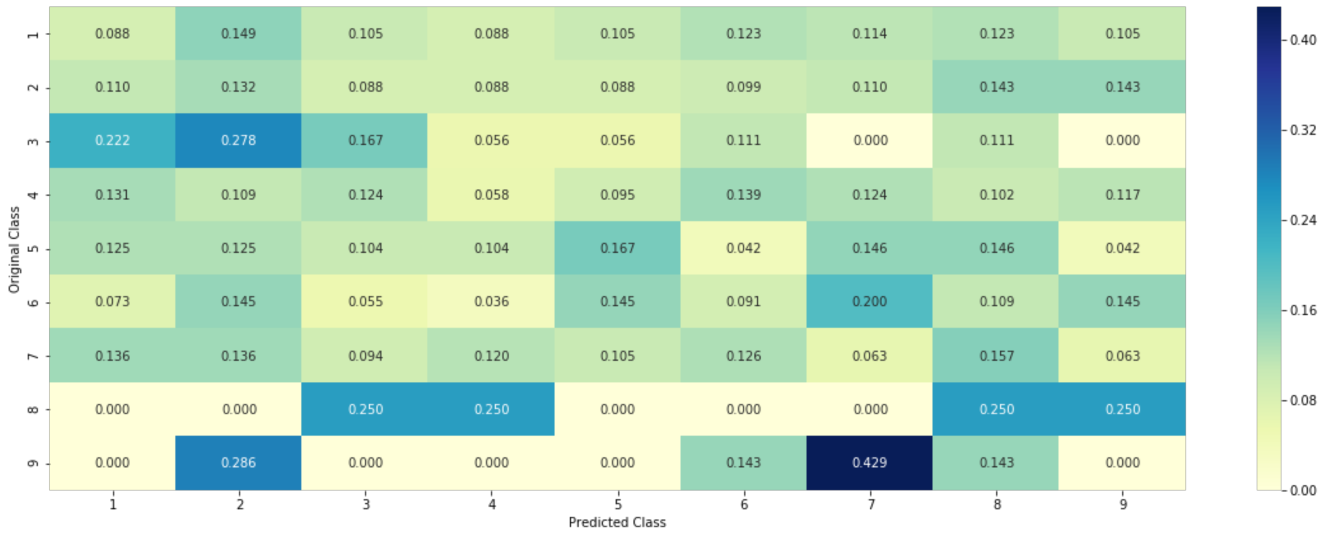




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



3.3 Univariate Analysis

In [15]:

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

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

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

    # denominator will contain the number of time that particular feature occured in 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 occur
ed 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):
```

```
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.037878787878787878, 0.068181818181818177, 0.1363636363636363
5, 0.25, 0.19318181818181818, 0.037878787878787878, 0.037878787878787878, 0.037878787878787878],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653061
5, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.0561224489
79591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818181
8181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878
782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.06060606060
6060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031446
55, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081
761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284768211920529
5, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556
2913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733333333333333
34, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.0666666666
66666666],
    #     ...
    #     }
    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 t
hen we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:

```
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 : 235
BRCA1     168
TP53      111
PTEN       86
EGFR       86
BRCA2      80
BRAF       65
KIT        60
ALK        44
ERBB2      39
PDGFRA     38
Name: Gene, dtype: int64
```

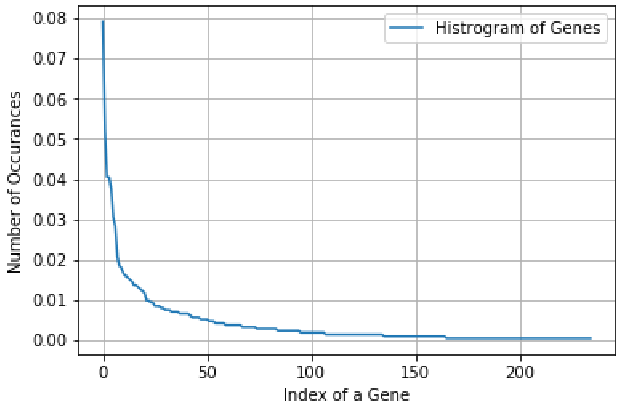
In [17]:

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

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

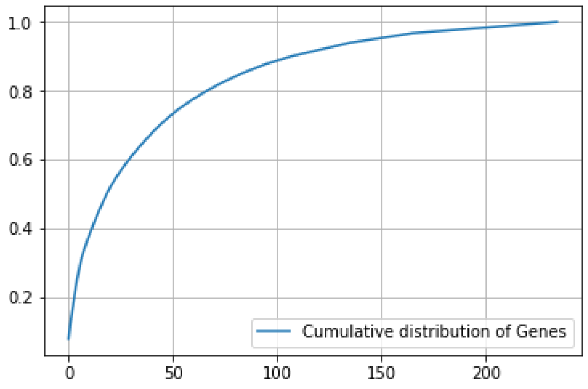
In [18]:

```
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 [19]:

```
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 [20]:

```
# 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 [21]:

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

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

In [22]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:

```
gene_vectorizer.get_feature_names()
```

Out[23]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11'
```

```
    'card11',
    'carm1',
    'casp8',
    'cbl',
    'ccnd1',
    'ccnd2',
    'ccnd3',
    'ccne1',
    'cdh1',
    'cdk12',
    'cdk4',
    'cdk6',
    'cdkn1a',
    'cdkn1b',
    'cdkn2a',
    'cdkn2b',
    'cdkn2c',
    'cebpa',
    'chek2',
    'cic',
    'crebbp',
    'ctcf',
    'ctla4',
    'ctnnb1',
    'ddr2',
    'dicer1',
    'dnmt3a',
    'dnmt3b',
    'egfr',
    'elf3',
    'ep300',
    'epas1',
    'epcam',
    'erbb2',
    'erbb3',
    'erbb4',
    'ercc2',
    'ercc3',
    'ercc4',
    'erg',
    'errfi1',
    'esr1',
    'etv1',
    'etv6',
    'ewsr1',
    'ezh2',
    'fanca',
    'fat1',
    'fbxw7',
    'fgf4',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'fgfr4',
    'flt1',
    'flt3',
    'foxa1',
    'foxl2',
    'foxp1',
    'fubp1',
    'gata3',
    'gli1',
    'gnaq',
    'gnas',
    'h3f3a',
    'hist1h1c',
    'hla',
    'hras',
    'idh1',
    'idh2',
    'igf1r',
    'ikbke',
    'ikzf1',
    'jak1',
    'jak2',
    'jun',
    'kdm5c',
    'kdm6a',
    'kdm6a',
    'kdr',
    'keap1',
    'kit',
    'kmt2a',
    'kmt2b',
    'kmt2c',
    'kmt2d',
    'knstrn',
    'kras',
    'lats2',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
    'mapk1',
    'mdm2',
    'mdm4',
    'med12',
    'mef2b',
    'men1',
    'met',
    'mlh1',
    'mpl',
    'msh2',
    'msh6',
    'mtor',
    'myc',
    'mycn',
    'myd88',
    'ncor1',
    'nf1',
    'nf2',
    'nfe2l2',
    'nfkbia',
    'nkx2',
    'notch1',
    'notch2',
    'nras',
    'ntrk1',
    'ntrk2',
    'ntrk3',
    'nup93',
    'pak1',
    'pax8',
    'pbrm1',
    'pdgfra',
    'pdgfrb',
    'pik3ca',
    'pik3cb',
    'pik3cd',
    'pik3r1',
    'pik3r2',
    'pik3r3',
    'pim1',
    'pms1',
    'pms2',
    'pole',
    'ppm1d',
    'ppp2r1a',
    'ppp6c',
    'prdm1',
    'ptch1',
    'pten',
    'ptpn11',
    'ptprd',
    'ptprt',
    'rab35',
    'rac1',
    'rad21',
    'rad50',
    'rad51b',
    'rad51d',
    'raf1',
    'rara',
    'rasa1',
    'rb1',
    'rbm10'
```

```
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [24]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene f
eature: (2124, 235)
```

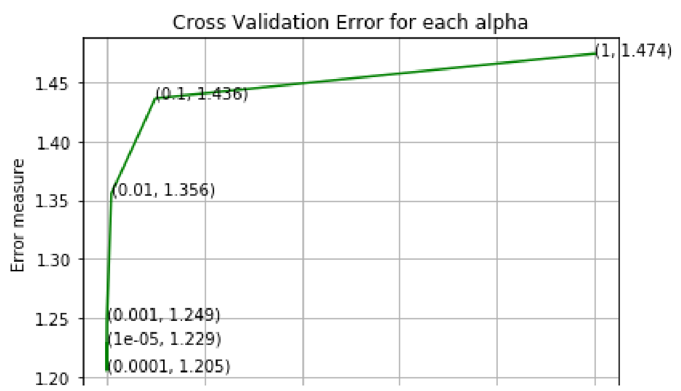
Q4. How good is this gene feature in predicting y_i?

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 [25]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_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, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2285570397154246
For values of alpha =  0.0001 The log loss is: 1.205397678433504
For values of alpha =  0.001 The log loss is: 1.2491650009150361
For values of alpha =  0.01 The log loss is: 1.3556362010078462
For values of alpha =  0.1 The log loss is: 1.4361429435939435
For values of alpha =  1 The log loss is: 1.4740301432719325
```



```
For values of best alpha =  0.0001 The train log loss is: 1.004691587658098
For values of best alpha =  0.0001 The cross validation log loss is: 1.205397678433504
For values of best alpha =  0.0001 The test log loss is: 1.1962093819056512
```

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 [26]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  235  genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 511 out of  532 : 96.05263157894737
```

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 [27]:

```
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 : 1920
Truncating_Mutations    58
Amplification           51
Deletion                45
Fusions                 25
G12V                     4
Overexpression           3
T167A                    2
R173C                    2
T58I                     2
F28L                     2
Name: Variation, dtype: int64
```

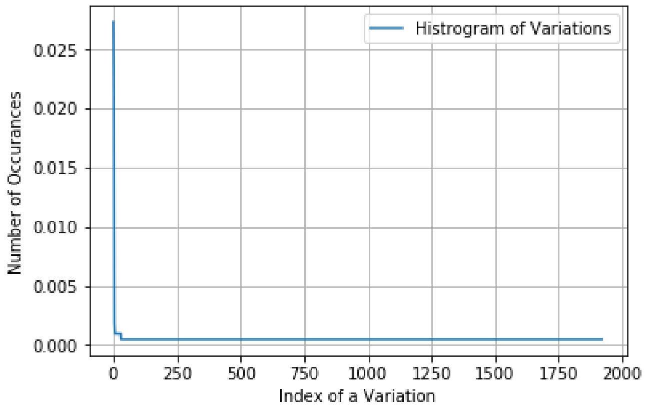
In [28]:

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

```
Ans: There are 1920 different categories of variations in the train data, and they are distibuted as fo
llows
```

In [29]:

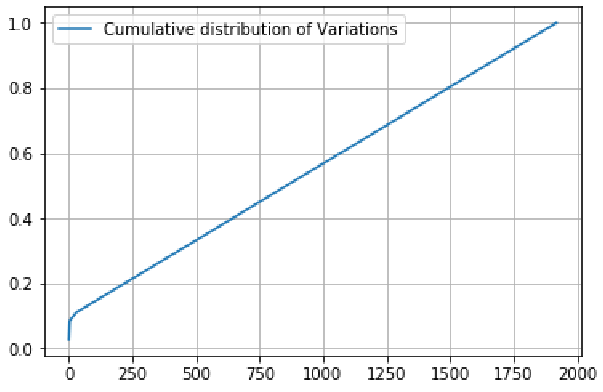
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [30]:

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

```
[0.02730697 0.05131827 0.07250471 ... 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 [31]:

```
# Response coding on 'Variation'

# 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 [32]:

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

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

In [33]:

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

In [34]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shap
e of Variation feature: (2124, 1948)
```

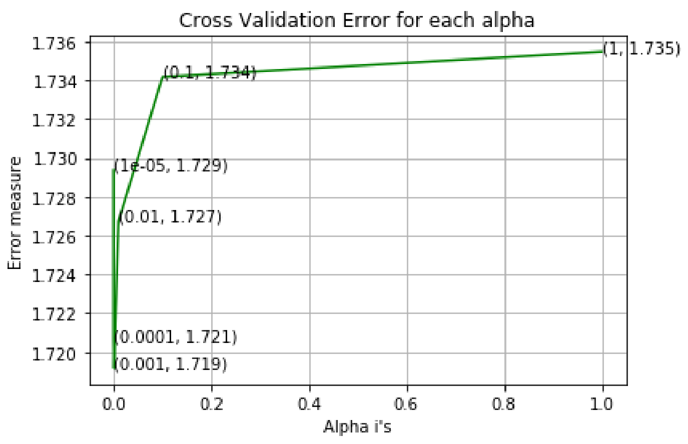
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [35]:

```
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.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7293899332490221
For values of alpha =  0.0001 The log loss is: 1.7205800598416228
For values of alpha =  0.001 The log loss is: 1.719154795433691
For values of alpha =  0.01 The log loss is: 1.7267443626734562
For values of alpha =  0.1 The log loss is: 1.7341851235193924
For values of alpha =  1 The log loss is: 1.7354863846993358
```



```
For values of best alpha =  0.001 The train log loss is: 1.0984554907261728
For values of best alpha =  0.001 The cross validation log loss is: 1.719154795433691
For values of best alpha =  0.001 The test log loss is: 1.7066671376163376
```

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 [36]:

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

```
Q12. How many data points are covered by total  1920  genes in test and cross validation data sets?
Ans
1. In test data 61 out of 665 : 9.172932330827068
2. In cross validation data 58 out of  532 : 10.902255639097744
```

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 [37]:

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

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

In [38]:

```
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[i+1]+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [67]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3, ngram_range=(1,4), max_features=100000)
# text_vectorizer = TfidfVectorizer(min_df=3, ngram_range=(1,4), max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# Select 1000 top feature
# selector = SelectKBest(k=2000).fit(train_text_feature_onehotCoding, y_train)
# train_text_feature_onehotCoding = selector.transform(train_text_feature_onehotCoding)

# getting all the feature names (words)
# train_text_features= [text_vectorizer.get_feature_names()[i] for i in selector.get_support(indices=Tr
ue)]
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 : 100000
```

In [40]:

```
# # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
# tfidf_model = TfidfVectorizer(min_df=3)
# tfidf_model.fit(train_df['TEXT'])
# # we are converting a dictionary with word as a key, and the idf as a value
# dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
# tfidf_words = set(tfidf_model.get_feature_names())
```

In [41]:

```
# # stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-
and-load-variables-in-python/
# # make sure you have the glove_vectors file
# import pickle

# with open('glove_vectors', 'rb') as f:
#     model = pickle.load(f)
#     glove_words =  set(model.keys())
```

In [42]:

```
# # average Word2Vec
# # compute average word2vec for each review.
# tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
# for sentence in tqdm_notebook(train_df['TEXT'].values): # for each review/sentence
#     vector = np.zeros(300) # as word vectors are of zero length
#     tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#     for word in sentence.split(): # for each word in a review/sentence
#         if (word in glove_words) and (word in train_text_features):
#             vec = model[word] # getting the vector for each word
#             # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(wo
rd)/len(sentence.split())))
#             tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfid
f value for each word
#             vector += (vec * tf_idf) # calculating tfidf weighted w2v
#             tf_idf_weight += tf_idf
#     if tf_idf_weight != 0:
#         vector /= tf_idf_weight
#     tfidf_w2v_vectors.append(vector)

# print(len(tfidf_w2v_vectors))
# print(len(tfidf_w2v_vectors[0]))
```

In [43]:

```
# train_text_feature_onehotCoding = np.array(tfidf_w2v_vectors)
```

In [44]:

```
# # average Word2Vec
# # compute average word2vec for each review.
# tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
# for sentence in tqdm_notebook(test_df['TEXT'].values): # for each review/sentence
#     vector = np.zeros(300) # as word vectors are of zero length
#     tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#     for word in sentence.split(): # for each word in a review/sentence
#         if (word in glove_words) and (word in train_text_features):
#             vec = model[word] # getting the vector for each word
#             # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(wo
rd)/len(sentence.split())))
#             tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfid
f value for each word
#             vector += (vec * tf_idf) # calculating tfidf weighted w2v
#             tf_idf_weight += tf_idf
#     if tf_idf_weight != 0:
#         vector /= tf_idf_weight
#     tfidf_w2v_vectors.append(vector)

# print(len(tfidf_w2v_vectors))
# print(len(tfidf_w2v_vectors[0]))
```

In [45]:

```
# test_text_feature_onehotCoding = np.array(tfidf_w2v_vectors)
```

In [46]:

```
# # average Word2Vec
# # compute average word2vec for each review.
# tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
# for sentence in tqdm_notebook(cv_df['TEXT'].values): # for each review/sentence
#     vector = np.zeros(300) # as word vectors are of zero length
#     tf_idf_weight =0; # num of words with a valid vector in the sentence/review
#     for word in sentence.split(): # for each word in a review/sentence
#         if (word in glove_words) and (word in train_text_features):
#             vec = model[word] # getting the vector for each word
#             # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(wo
rd)/len(sentence.split())))
#             tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfid
f value for each word
#             vector += (vec * tf_idf) # calculating tfidf weighted w2v
#             tf_idf_weight += tf_idf
#     if tf_idf_weight != 0:
#         vector /= tf_idf_weight
#     tfidf_w2v_vectors.append(vector)

# print(len(tfidf_w2v_vectors))
# print(len(tfidf_w2v_vectors[0]))
```

In [47]:

```
# cv_text_feature_onehotCoding = np.array(tfidf_w2v_vectors)
```

In [48]:

```
# # Ref : https://stackabuse.com/reading-and-writing-csv-files-in-python/
# # save this tfidf w2v for train, text and cv into csv time to save time for further processing.

# import csv

# tr_File = open('tr_tfidf_w2v.csv', 'w', newline='')
# ts_File = open('ts_tfidf_w2v.csv', 'w', newline='')
# cv_File = open('cv_tfidf_w2v.csv', 'w', newline='')

# with tr_File:
#     writer = csv.writer(tr_File)
#     writer.writerows(train_text_feature_onehotCoding)

# with ts_File:
#     writer = csv.writer(ts_File)
#     writer.writerows(test_text_feature_onehotCoding)

# with cv_File:
#     writer = csv.writer(cv_File)
#     writer.writerows(cv_text_feature_onehotCoding)
```

In [49]:

```
# train_text_feature_onehotCoding = pd.read_csv('tr_tfidf_w2v.csv', header=None)
# train_text_feature_onehotCoding = train_text_feature_onehotCoding.values
```

In [50]:

```
# test_text_feature_onehotCoding = pd.read_csv('ts_tfidf_w2v.csv', header=None)
# test_text_feature_onehotCoding = test_text_feature_onehotCoding.values
```

In [51]:

```
# cv_text_feature_onehotCoding = pd.read_csv('cv_tfidf_w2v.csv', header=None)
# cv_text_feature_onehotCoding = cv_text_feature_onehotCoding.values
```

In [52]:

```
# from sklearn.preprocessing import MinMaxScaler
# train_text_feature_onehotCoding = MinMaxScaler().fit_transform(train_text_feature_onehotCoding)
# test_text_feature_onehotCoding = MinMaxScaler().fit_transform(test_text_feature_onehotCoding)
# cv_text_feature_onehotCoding = MinMaxScaler().fit_transform(cv_text_feature_onehotCoding)
```

In [53]:

```
# np.amax(train_text_feature_onehotCoding)
```

In [68]:

```
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 = train_df.groupby('Class').count().to_dict().get('ID')


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[j+1]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

In [57]:

```
#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 [58]:

```
# 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_responseCod
ing.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding
.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(a
xis=1)).T
```

In [69]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
```

In [70]:

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

In [71]:

```
train_text_feature_onehotCoding.shape, test_text_feature_onehotCoding.shape, cv_text_feature_onehotCodi
ng.shape
```

Out[71]:

```
((2124, 100000), (665, 100000), (532, 100000))
```

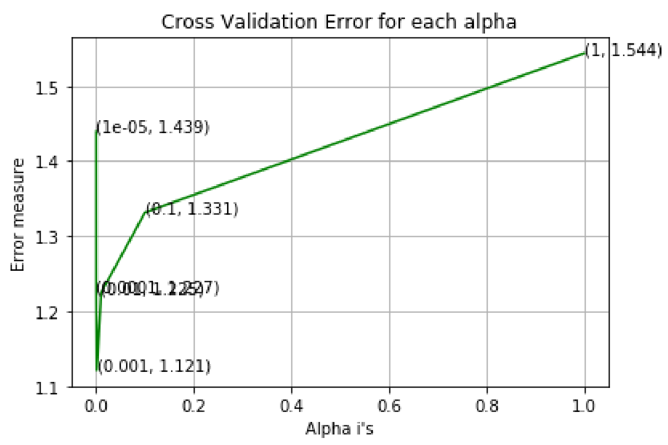
In [72]:

```
#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 [74]:

```
# 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_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.4393007857241316
For values of alpha =  0.0001 The log loss is: 1.2268882010555906
For values of alpha =  0.001 The log loss is: 1.1209445408199195
For values of alpha =  0.01 The log loss is: 1.2245940853991435
For values of alpha =  0.1 The log loss is: 1.3313327895671498
For values of alpha =  1 The log loss is: 1.5437178667632947
```



```
For values of best alpha =  0.001 The train log loss is: 0.6464015334502864
For values of best alpha =  0.001 The cross validation log loss is: 1.1209445408199195
For values of best alpha =  0.001 The test log loss is: 1.1267116418063567
```

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

Ans. Yes, it seems like!

In [75]:

```
def get_intersec_text(df,y):
    df_text_vec = TfidfVectorizer(min_df=3, ngram_range=(1,4), max_features=100000)
#     df_text_vec = TfidfVectorizer(min_df=3, ngram_range=(4,4))
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
#     selector = SelectKBest(k=10000).fit(df_text_fea, y)
#     df_text_fea = selector.transform(df_text_fea)
#     df_text_features = [df_text_vec.get_feature_names()[i] for i in selector.get_support(indices=True
)]
    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 [76]:

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

```
78.242 % of word of test data appeared in train data
76.64 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [77]:

```
#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 [78]:

```
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 [79]:

```
# generate ngram from sentence
# Ref: http://www.albertauyeung.com/post/generating-ngrams-python/
def generate_ngrams(s, n):
    # Convert to lowercases
    s = s.lower()

    # Replace all none alphanumeric characters with spaces
    s = re.sub(r'[^a-zA-Z0-9\s]', ' ', s)

    # Break sentence in the token, remove empty tokens
    tokens = [token for token in s.split(" ") if token != ""]

    # Use the zip function to help us generate n-grams
    # Concatentate the tokens into ngrams and return
    ngrams = zip(*[tokens[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]
```

In [80]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3, ngram_range=(1,4), max_features=100000)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    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'])
#     selector = SelectKBest(k=1000).fit(text_vec,y_train)
#     text_fea = [text_count_vec.get_feature_names()[i] for i in selector.get_support(indices=True)]
    text_fea = text_count_vec.get_feature_names()

    all_features = gene_count_vec.get_feature_names() + var_count_vec.get_feature_names() + text_fea

    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 = all_features[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 = all_features[v]
            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 = all_features[v]
            text_1 = generate_ngrams(text,1)
            text_2 = generate_ngrams(text,2)
            text_3 = generate_ngrams(text,3)
            text_4 = generate_ngrams(text,4)
            yes_no = True if word in text_1 else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

            yes_no = True if word in text_2 else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

            yes_no = True if word in text_3 else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

            yes_no = True if word in text_4 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 [81]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCod
ing))
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_re
sponseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_respo
nseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCod
ing))

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 [82]:

```
train_text_feature_onehotCoding.shape, train_gene_feature_onehotCoding.shape, train_variation_feature_o
nehotCoding.shape
```

Out[82]:

```
((2124, 100000), (2124, 235), (2124, 1948))
```

In [83]:

```
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.shap
e)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 102183)
(number of data points * number of features) in test data =  (665, 102183)
(number of data points * number of features) in cross validation data = (532, 102183)
```

In [84]:

```
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.sh
ape)
```

```
 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

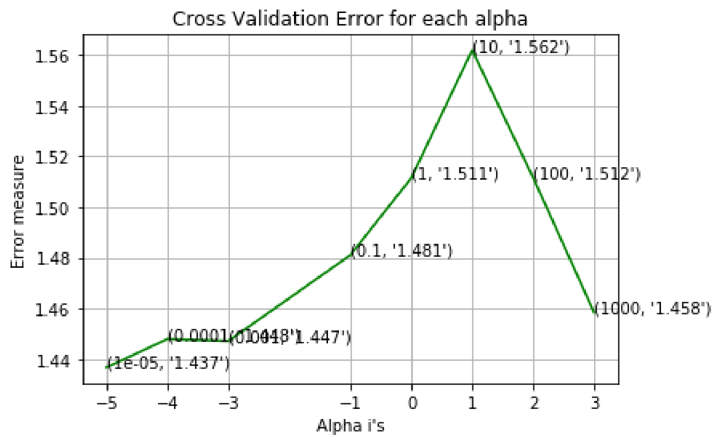
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [85]:

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

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)

    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.4366699393940812
for alpha = 0.0001
Log Loss : 1.4478349068507887
for alpha = 0.001
Log Loss : 1.4470496629539584
for alpha = 0.1
Log Loss : 1.4809637056066733
for alpha = 1
Log Loss : 1.5114095592450187
for alpha = 10
Log Loss : 1.5619387615255806
for alpha = 100
Log Loss : 1.5116467139359115
for alpha = 1000
Log Loss : 1.4583896977863007
```



```
For values of best alpha =  1e-05 The train log loss is: 1.064104554655866
For values of best alpha =  1e-05 The cross validation log loss is: 1.4366699393940812
For values of best alpha =  1e-05 The test log loss is: 1.4139971999299759
```

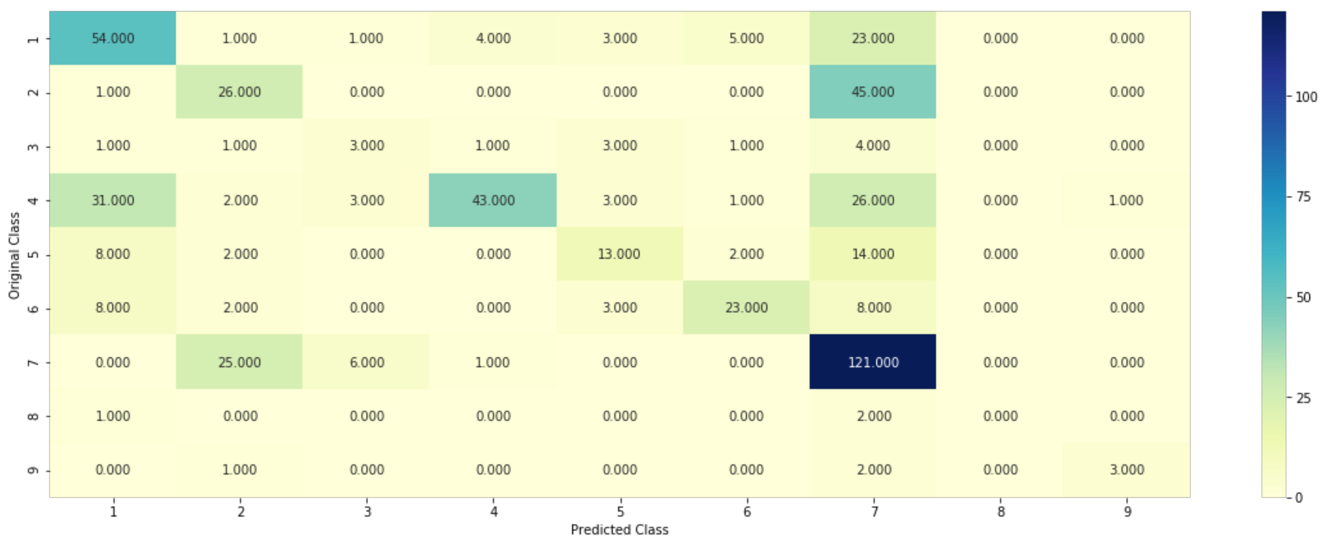
4.1.1.2. Testing the model with best hyper paramters

In [86]:

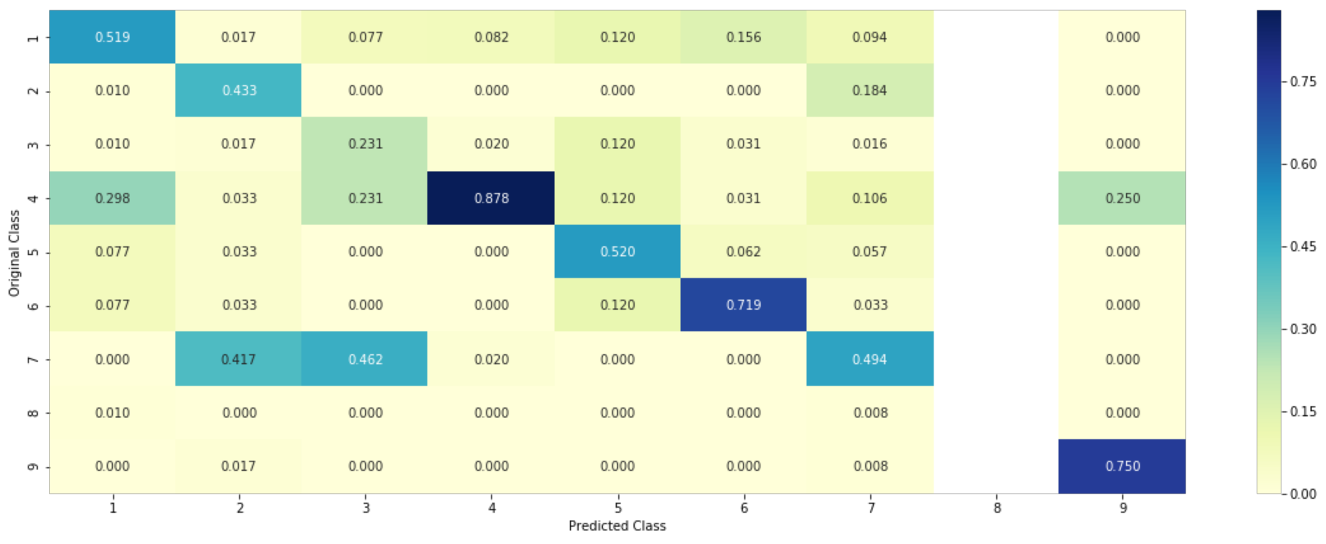
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated
/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

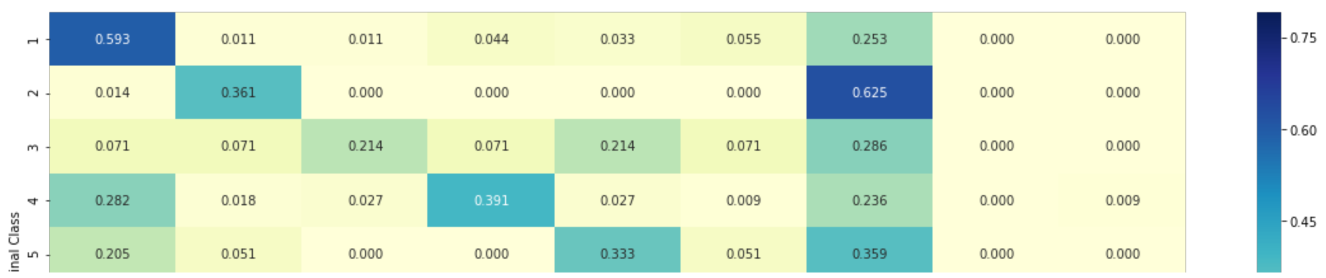
```
Log Loss : 1.4366699393940812
Number of missclassified point : 0.462406015037594
-------------------- Confusion matrix --------------------
```

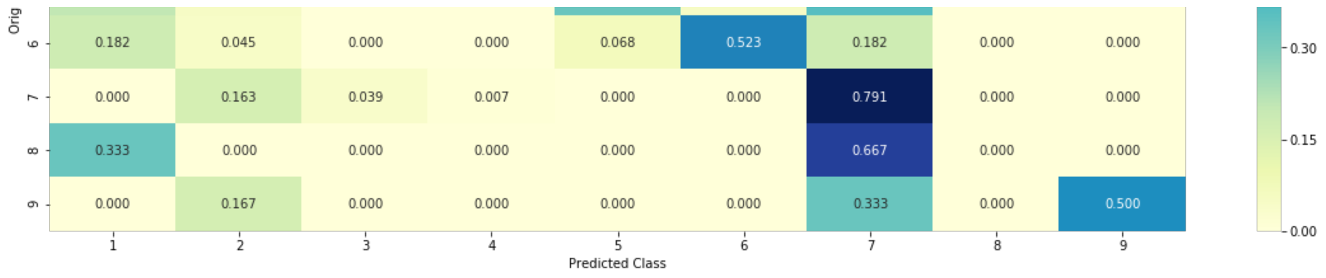


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.1.1.3. Feature Importance, Correctly classified point

In [87]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0724 0.0634 0.0114 0.1076 0.0485 0.6075 0.0798 0.0049 0.0045]]
Actual Class : 6
--------------------------------------------------
4 Text feature [majority variants] present in test data point [True]
6 Text feature [family personal] present in test data point [True]
8 Text feature [model brca1] present in test data point [True]
9 Text feature [relevance brca1] present in test data point [True]
10 Text feature [70 000] present in test data point [True]
11 Text feature [basis mutation] present in test data point [True]
12 Text feature [deleterious mutations] present in test data point [True]
13 Text feature [i124v] present in test data point [True]
14 Text feature [000 individuals] present in test data point [True]
15 Text feature [trans deleterious mutation] present in test data point [True]
16 Text feature [trans deleterious] present in test data point [True]
17 Text feature [probability deleterious] present in test data point [True]
18 Text feature [conservative substitution] present in test data point [True]
19 Text feature [odds] present in test data point [True]
20 Text feature [clinical interpretation] present in test data point [True]
21 Text feature [personal history] present in test data point [True]
22 Text feature [deleterious mutation] present in test data point [True]
23 Text feature [brca2 dna binding domain] present in test data point [True]
26 Text feature [brca testing] present in test data point [True]
27 Text feature [brca] present in test data point [True]
28 Text feature [ivs20] present in test data point [True]
29 Text feature [ivs5] present in test data point [True]
30 Text feature [brca2 dna binding] present in test data point [True]
31 Text feature [ivs13] present in test data point [True]
32 Text feature [18 brca1] present in test data point [True]
33 Text feature [57] present in test data point [True]
34 Text feature [ethnic populations] present in test data point [True]
35 Text feature [personal family history] present in test data point [True]
36 Text feature [favor deleterious] present in test data point [True]
37 Text feature [individuals known deleterious] present in test data point [True]
38 Text feature [variants 33] present in test data point [True]
39 Text feature [also useful classification] present in test data point [True]
40 Text feature [vus observed] present in test data point [True]
41 Text feature [provide overall] present in test data point [True]
42 Text feature [carry variants] present in test data point [True]
43 Text feature [carry deleterious mutation] present in test data point [True]
44 Text feature [i2285v] present in test data point [True]
45 Text feature [carriers known deleterious] present in test data point [True]
47 Text feature [additional brca] present in test data point [True]
48 Text feature [prevention therapeutic] present in test data point [True]
49 Text feature [known deleterious truncating] present in test data point [True]
50 Text feature [minority ethnic populations] present in test data point [True]
51 Text feature [predicted affect splicing] present in test data point [True]
```

```
51 Text feature [predicted affect splicing] present in test data point [True]
52 Text feature [70 000 individuals] present in test data point [True]
53 Text feature [variant classified deleterious] present in test data point [True]
54 Text feature [brca protein] present in test data point [True]
55 Text feature [carry deleterious] present in test data point [True]
56 Text feature [three sources] present in test data point [True]
57 Text feature [sequence alignments] present in test data point [True]
58 Text feature [factors combined] present in test data point [True]
59 Text feature [known deleterious] present in test data point [True]
60 Text feature [brca1 brca2 deleterious] present in test data point [True]
61 Text feature [useful classification] present in test data point [True]
63 Text feature [family members carry] present in test data point [True]
64 Text feature [brca mutation] present in test data point [True]
65 Text feature [members carry] present in test data point [True]
66 Text feature [deleterious] present in test data point [True]
68 Text feature [personal] present in test data point [True]
69 Text feature [deleterious brca1] present in test data point [True]
70 Text feature [dna binding domain amino] present in test data point [True]
71 Text feature [personal family] present in test data point [True]
72 Text feature [used prediction] present in test data point [True]
73 Text feature [binding domain amino acids] present in test data point [True]
74 Text feature [brca2 dna] present in test data point [True]
75 Text feature [different deleterious] present in test data point [True]
76 Text feature [minority ethnic] present in test data point [True]
77 Text feature [individuals known] present in test data point [True]
79 Text feature [basis] present in test data point [True]
80 Text feature [final data] present in test data point [True]
82 Text feature [binding domain amino] present in test data point [True]
83 Text feature [classified deleterious] present in test data point [True]
84 Text feature [using data] present in test data point [True]
85 Text feature [brca2 deleterious] present in test data point [True]
86 Text feature [falling] present in test data point [True]
87 Text feature [also useful] present in test data point [True]
88 Text feature [neutral vus] present in test data point [True]
89 Text feature [deleterious variants] present in test data point [True]
90 Text feature [favor] present in test data point [True]
91 Text feature [brca1 mutation] present in test data point [True]
92 Text feature [predicted] present in test data point [True]
93 Text feature [carry] present in test data point [True]
94 Text feature [myriad genetic laboratories] present in test data point [True]
95 Text feature [genetic laboratories] present in test data point [True]
96 Text feature [myriad genetic] present in test data point [True]
97 Text feature [brca1 brca2] present in test data point [True]
98 Text feature [intronic variant] present in test data point [True]
99 Text feature [risk family] present in test data point [True]
Out of the top  100  features  87 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [88]:

```python
test_point_index = 2
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_i
ndex]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point
_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0944 0.0826 0.0149 0.1392 0.0631 0.0449 0.5486 0.0063 0.0059]]
Actual Class : 6
--------------------------------------------------
16 Text feature [cells] present in test data point [True]
17 Text feature [activation] present in test data point [True]
21 Text feature [presence] present in test data point [True]
22 Text feature [kinase] present in test data point [True]
23 Text feature [downstream] present in test data point [True]
24 Text feature [contrast] present in test data point [True]
25 Text feature [activated] present in test data point [True]
```

```
26 Text feature [cell] present in test data point [True]
27 Text feature [shown] present in test data point [True]
28 Text feature [also] present in test data point [True]
30 Text feature [however] present in test data point [True]
31 Text feature [phosphorylation] present in test data point [True]
32 Text feature [inhibitor] present in test data point [True]
33 Text feature [factor] present in test data point [True]
35 Text feature [addition] present in test data point [True]
36 Text feature [well] present in test data point [True]
37 Text feature [suggest] present in test data point [True]
38 Text feature [compared] present in test data point [True]
39 Text feature [growth] present in test data point [True]
40 Text feature [10] present in test data point [True]
41 Text feature [increased] present in test data point [True]
42 Text feature [previously] present in test data point [True]
43 Text feature [independent] present in test data point [True]
45 Text feature [signaling] present in test data point [True]
46 Text feature [similar] present in test data point [True]
47 Text feature [found] present in test data point [True]
48 Text feature [mechanism] present in test data point [True]
49 Text feature [1a] present in test data point [True]
50 Text feature [higher] present in test data point [True]
51 Text feature [treated] present in test data point [True]
52 Text feature [mutations] present in test data point [True]
53 Text feature [showed] present in test data point [True]
54 Text feature [kinase domain] present in test data point [True]
56 Text feature [interestingly] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
58 Text feature [consistent] present in test data point [True]
59 Text feature [may] present in test data point [True]
60 Text feature [3b] present in test data point [True]
61 Text feature [figure] present in test data point [True]
63 Text feature [total] present in test data point [True]
64 Text feature [observed] present in test data point [True]
65 Text feature [followed] present in test data point [True]
68 Text feature [obtained] present in test data point [True]
69 Text feature [mutant] present in test data point [True]
70 Text feature [3a] present in test data point [True]
71 Text feature [activating] present in test data point [True]
72 Text feature [demonstrated] present in test data point [True]
73 Text feature [including] present in test data point [True]
76 Text feature [absence] present in test data point [True]
77 Text feature [treatment] present in test data point [True]
78 Text feature [inhibited] present in test data point [True]
79 Text feature [reported] present in test data point [True]
80 Text feature [tyrosine] present in test data point [True]
81 Text feature [fig] present in test data point [True]
83 Text feature [sensitive] present in test data point [True]
85 Text feature [suggesting] present in test data point [True]
86 Text feature [growth factor] present in test data point [True]
88 Text feature [mutation] present in test data point [True]
89 Text feature [without] present in test data point [True]
91 Text feature [expression] present in test data point [True]
92 Text feature [respectively] present in test data point [True]
93 Text feature [together] present in test data point [True]
94 Text feature [expressed] present in test data point [True]
95 Text feature [inhibition] present in test data point [True]
96 Text feature [increase] present in test data point [True]
98 Text feature [two] present in test data point [True]
99 Text feature [performed] present in test data point [True]
Out of the top  100  features  67 are present in query point
```

4.2. K Nearest Neighbour Classification

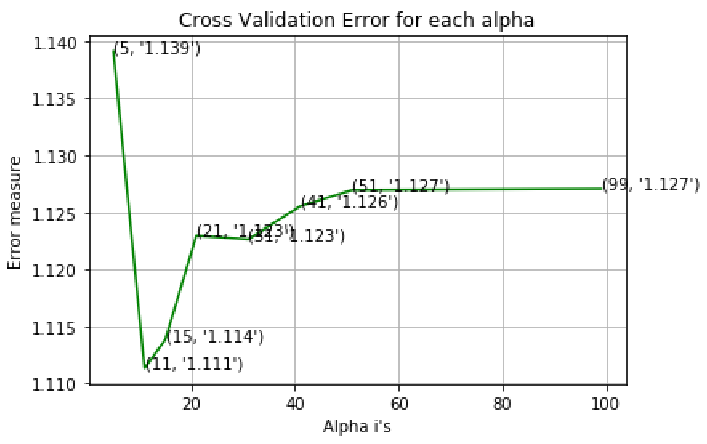
4.2.1. Hyper parameter tuning

In [89]:

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

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    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, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.139099679856894
for alpha = 11
Log Loss : 1.111344495549769
for alpha = 15
Log Loss : 1.1138335999233595
for alpha = 21
Log Loss : 1.1229670014029776
```

```
for alpha = 31
Log Loss : 1.1226455024911766
for alpha = 41
Log Loss : 1.1255304330843283
for alpha = 51
Log Loss : 1.1269446985863898
for alpha = 99
Log Loss : 1.127050761823645
```



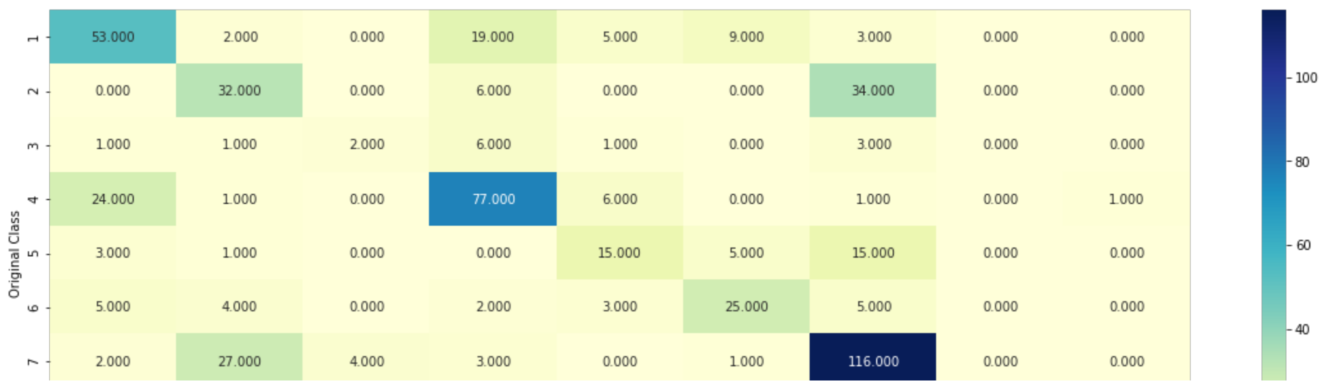
```
For values of best alpha =  11 The train log loss is: 0.6153065208699499
For values of best alpha =  11 The cross validation log loss is: 1.111344495549769
For values of best alpha =  11 The test log loss is: 1.0357413240294981
```

4.2.2. Testing the model with best hyper paramters

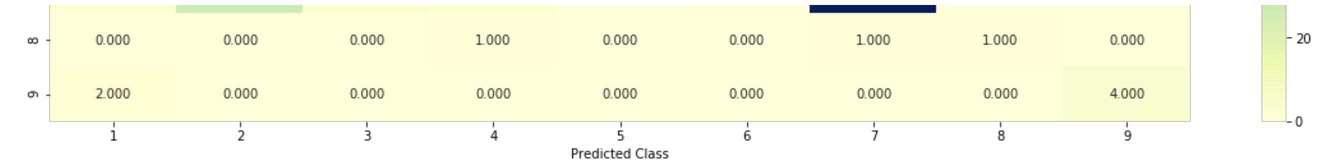
In [90]:

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

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

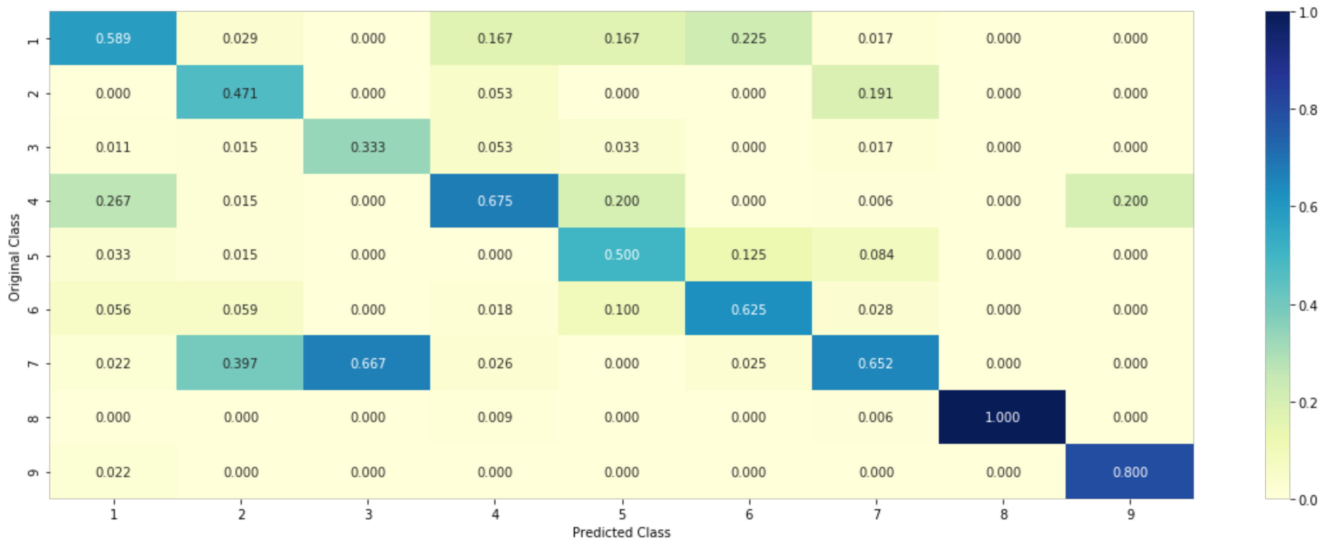
```
Log loss : 1.111344495549769
Number of mis-classified points : 0.38909774436090223
-------------------- Confusion matrix --------------------
```



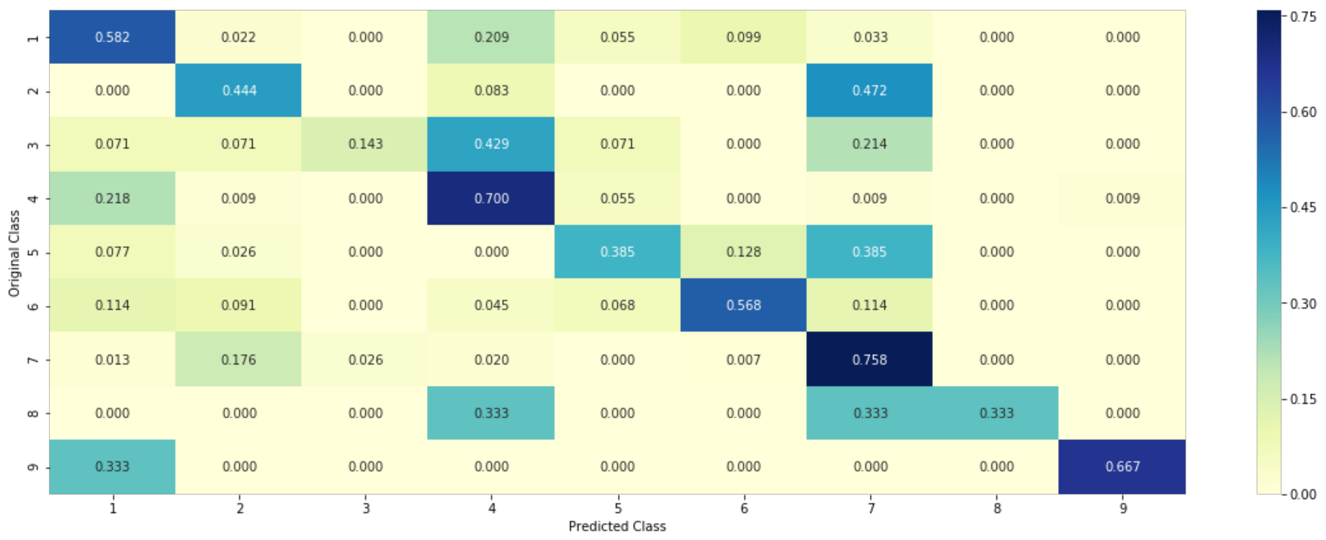
1	53.000	2.000	0.000	19.000	5.000	9.000	3.000	0.000	0.000
2	0.000	32.000	0.000	6.000	0.000	0.000	34.000	0.000	0.000
3	1.000	1.000	2.000	6.000	1.000	0.000	3.000	0.000	0.000
4	24.000	1.000	0.000	77.000	6.000	0.000	1.000	0.000	1.000
5	3.000	1.000	0.000	0.000	15.000	5.000	15.000	0.000	0.000
6	5.000	4.000	0.000	2.000	3.000	25.000	5.000	0.000	0.000
7	2.000	27.000	4.000	3.000	0.000	1.000	116.000	0.000	0.000



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



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



4.2.3.Sample Query point -1

In [91]:

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

```
Predicted Class : 4
Actual Class : 6
The  11  nearest neighbours of the test points belongs to classes [6 6 6 6 6 6 6 6 6 6 6]
Fequency of nearest points : Counter({6: 11})
```

4.2.4. Sample Query Point-2

In [92]:

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

test_point_index = 100

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

```
Predicted Class : 1
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [1 1 1 1 4 1
1 1 1 4 1]
Fequency of nearest points : Counter({1: 9, 4: 2})
```

4.3. Logistic Regression

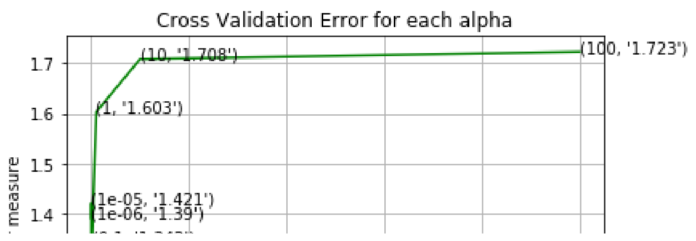
4.3.1. With Class balancing

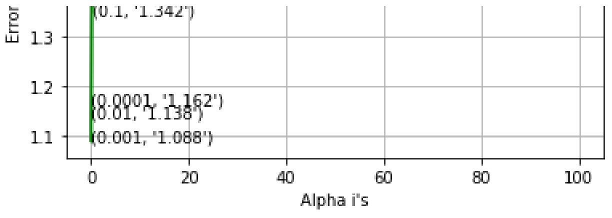
4.3.1.1. Hyper paramter tuning

In [93]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.3900065883111294
for alpha = 1e-05
Log Loss : 1.4206220986558065
for alpha = 0.0001
Log Loss : 1.1620651416742773
for alpha = 0.001
Log Loss : 1.0883268311625434
for alpha = 0.01
Log Loss : 1.138384437668513
for alpha = 0.1
Log Loss : 1.3415326319447602
for alpha = 1
Log Loss : 1.6025843280837841
for alpha = 10
Log Loss : 1.7083069388533076
for alpha = 100
Log Loss : 1.7229433345155563
```





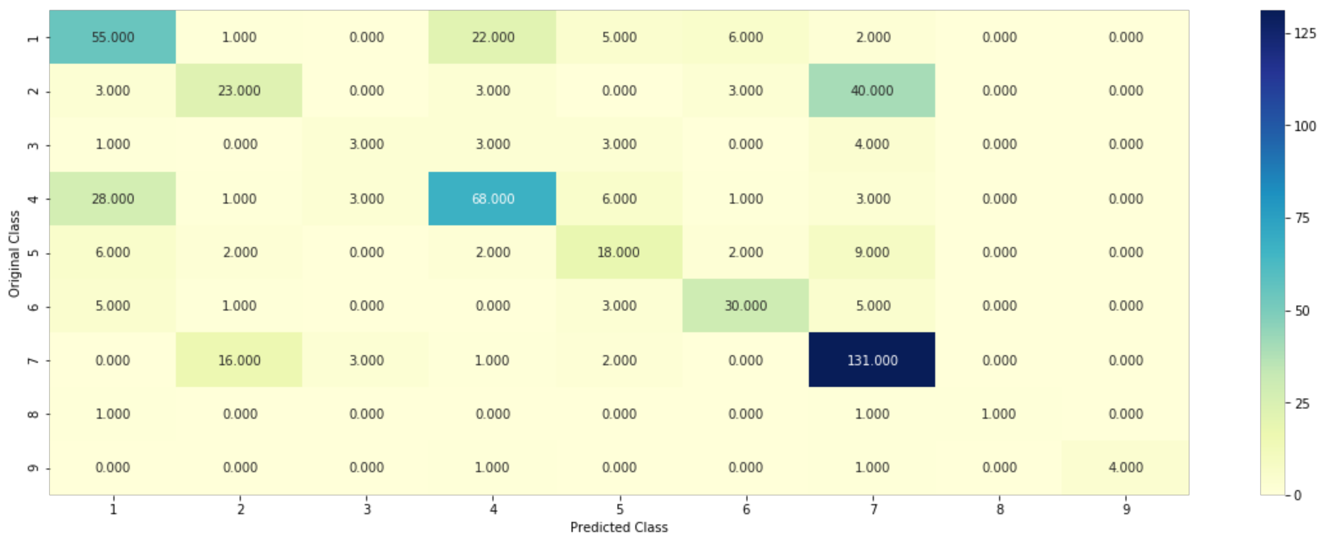
```
For values of best alpha =  0.001 The train log loss is: 0.5667340417941494
For values of best alpha =  0.001 The cross validation log loss is: 1.0883268311625434
For values of best alpha =  0.001 The test log loss is: 1.089275401933807
```

4.3.1.2. Testing the model with best hyper paramters

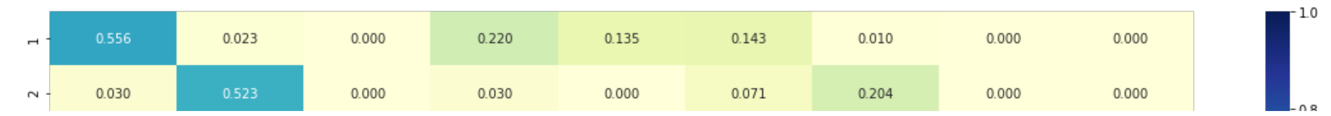
In [94]:

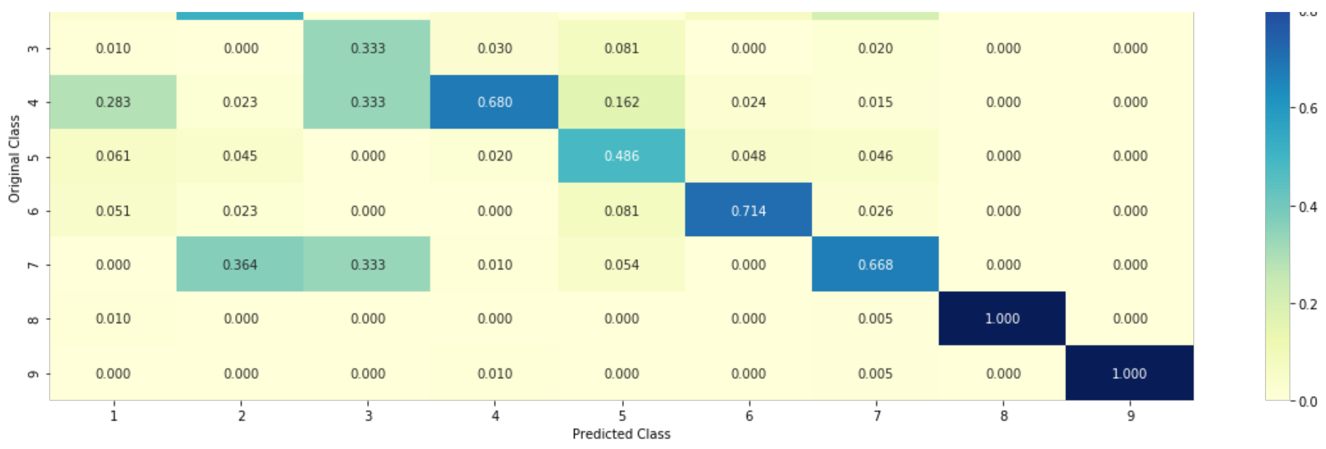
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.0883268311625434
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```

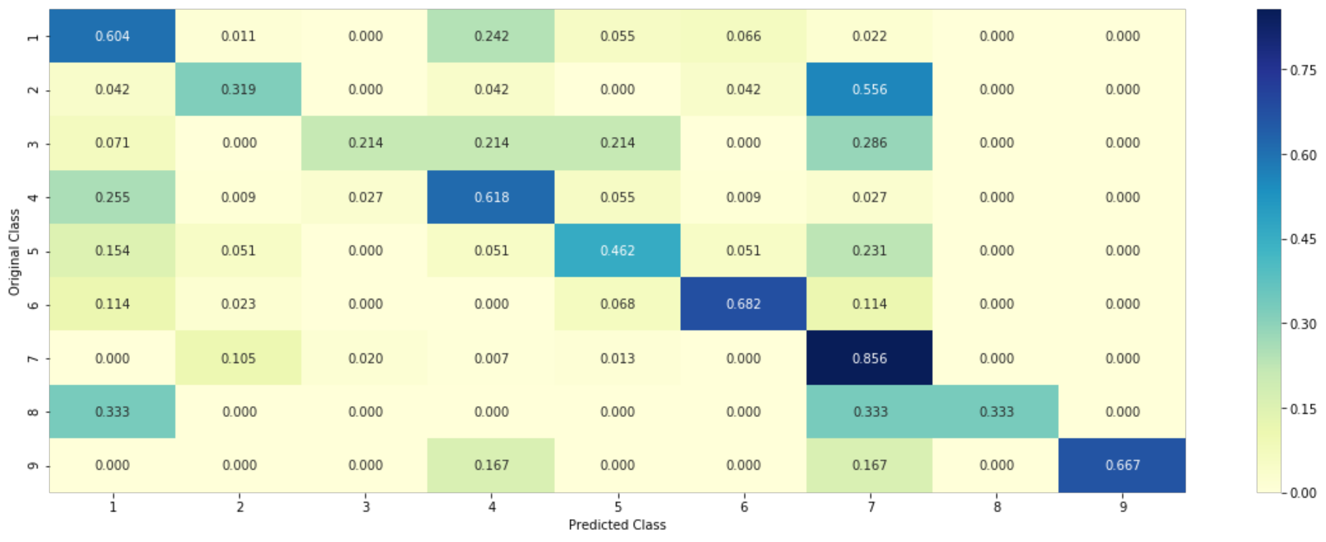


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.1.3. Feature Importance

In [95]:

```
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 [96]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[1.000e-03 2.000e-04 9.000e-04 2.000e-03 5.680e-02 9.385e-01 0.000e+00
  5.000e-04 0.000e+00]]
Actual Class : 6
--------------------------------------------------
96 Text feature [amplification] present in test data point [True]
187 Text feature [women] present in test data point [True]
225 Text feature [african] present in test data point [True]
336 Text feature [ethnic groups] present in test data point [True]
347 Text feature [homozygotes] present in test data point [True]
448 Text feature [ancestry] present in test data point [True]
475 Text feature [ductal carcinoma] present in test data point [True]
476 Text feature [given family] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [97]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.048  0.0492 0.0103 0.0633 0.0356 0.0096 0.7793 0.0032 0.0015]]
Actual Class : 6
--------------------------------------------------
48 Text feature [domain abd] present in test data point [True]
71 Text feature [binding domain abd] present in test data point [True]
139 Text feature [levels phosphorylated] present in test data point [True]
159 Text feature [activated] present in test data point [True]
190 Text feature [tumor recurrence] present in test data point [True]
202 Text feature [sensitive mek] present in test data point [True]
250 Text feature [nsh2 domain] present in test data point [True]
315 Text feature [nsh2] present in test data point [True]
329 Text feature [increased pi3k] present in test data point [True]
360 Text feature [activated pi3k] present in test data point [True]
362 Text feature [overexpression] present in test data point [True]
372 Text feature [attractive] present in test data point [True]
386 Text feature [pik3r2] present in test data point [True]
387 Text feature [molecular features] present in test data point [True]
407 Text feature [genetic aberrations] present in test data point [True]
415 Text feature [responsiveness] present in test data point [True]
436 Text feature [activation] present in test data point [True]
475 Text feature [cancers data] present in test data point [True]
487 Text feature [levels significantly] present in test data point [True]
Out of the top  500  features  19 are present in query point
```

4.3.2. Without Class balancing

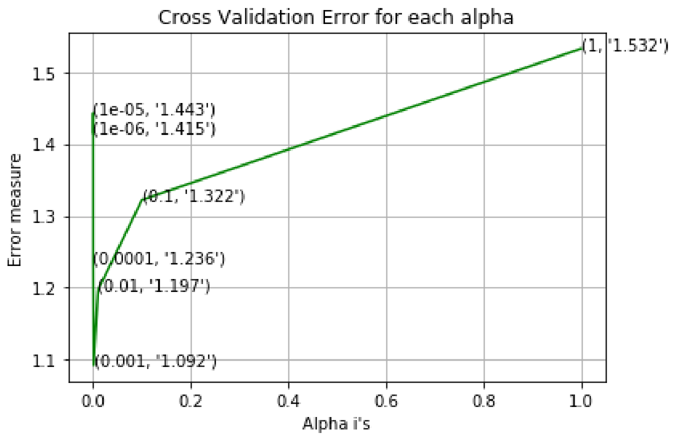
4.3.2.1. Hyper paramter tuning

In [98]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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

```
for alpha = 1e-06
Log Loss : 1.41529972834051
for alpha = 1e-05
Log Loss : 1.4430772403884005
for alpha = 0.0001
Log Loss : 1.2356484177535565
for alpha = 0.001
Log Loss : 1.0920236953958784
for alpha = 0.01
Log Loss : 1.1971917131604306
for alpha = 0.1
Log Loss : 1.322039559781752
for alpha = 1
Log Loss : 1.5322009015730178
```



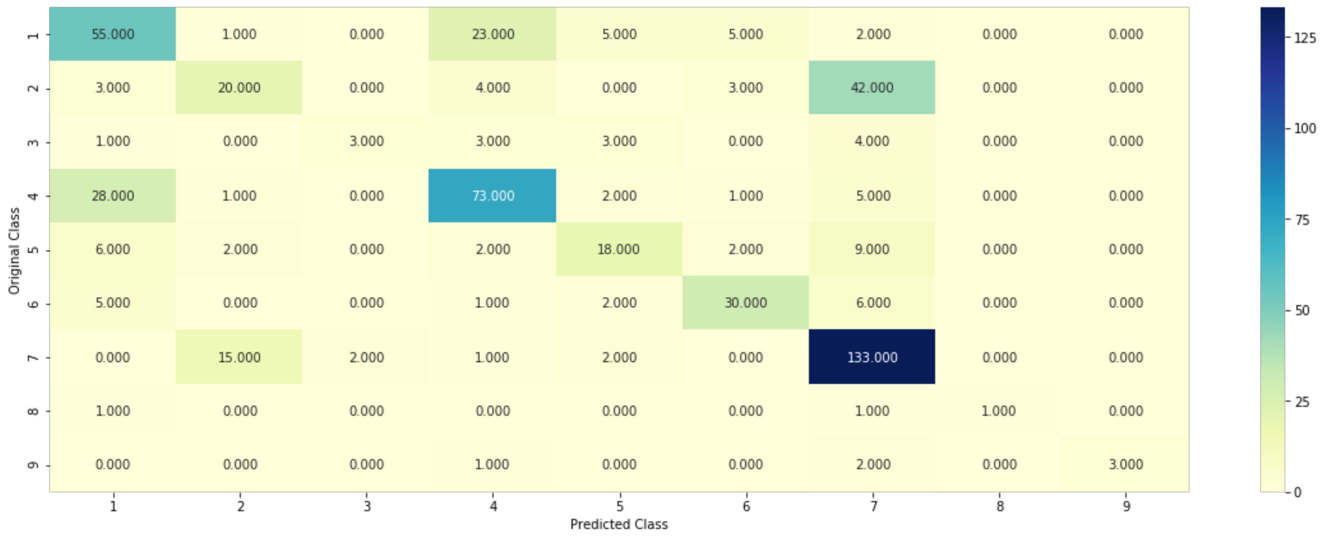
```
For values of best alpha =  0.001 The train log loss is: 0.5617525074982848
For values of best alpha =  0.001 The cross validation log loss is: 1.0920236953958784
For values of best alpha =  0.001 The test log loss is: 1.087075596416989
```

4.3.2.2. Testing model with best hyper parameters

In [99]:

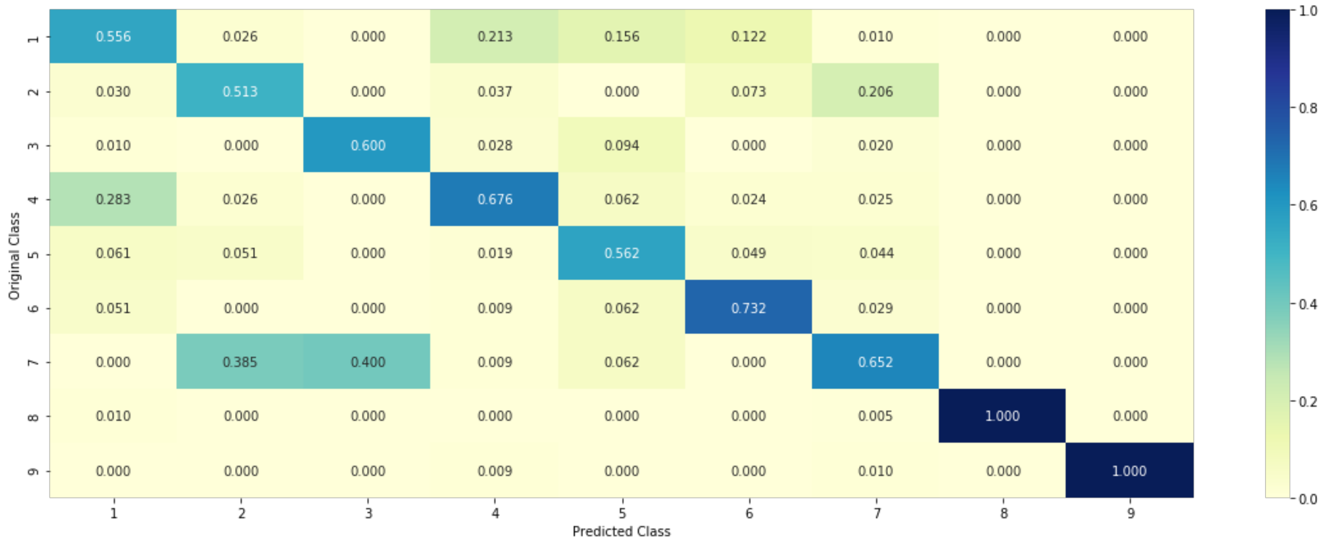
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.0920236953958784
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



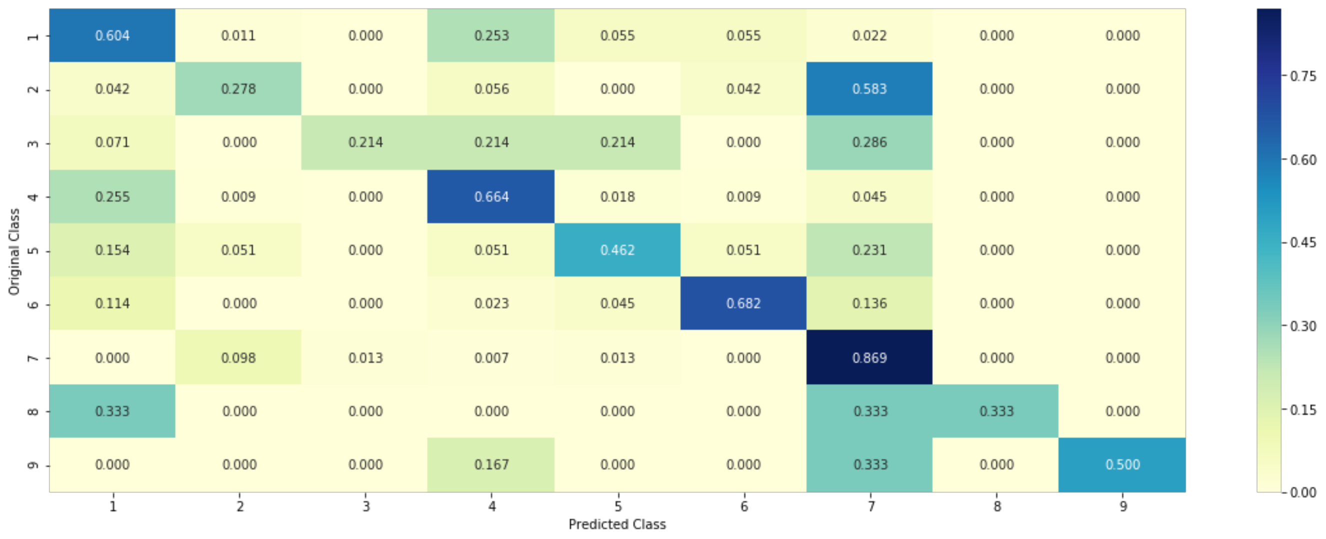
	1	2	3	4	5	6	7	8	9
1	55.000	1.000	0.000	23.000	5.000	5.000	2.000	0.000	0.000
2	3.000	20.000	0.000	4.000	0.000	3.000	42.000	0.000	0.000
3	1.000	0.000	3.000	3.000	3.000	0.000	4.000	0.000	0.000
4	28.000	1.000	0.000	73.000	2.000	1.000	5.000	0.000	0.000
5	6.000	2.000	0.000	2.000	18.000	2.000	9.000	0.000	0.000
6	5.000	0.000	0.000	1.000	2.000	30.000	6.000	0.000	0.000
7	0.000	15.000	2.000	1.000	2.000	0.000	133.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	3.000

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



	1	2	3	4	5	6	7	8	9
1	0.556	0.026	0.000	0.213	0.156	0.122	0.010	0.000	0.000
2	0.030	0.513	0.000	0.037	0.000	0.073	0.206	0.000	0.000
3	0.010	0.000	0.600	0.028	0.094	0.000	0.020	0.000	0.000
4	0.283	0.026	0.000	0.676	0.062	0.024	0.025	0.000	0.000
5	0.061	0.051	0.000	0.019	0.562	0.049	0.044	0.000	0.000
6	0.051	0.000	0.000	0.009	0.062	0.732	0.029	0.000	0.000
7	0.000	0.385	0.400	0.009	0.062	0.000	0.652	0.000	0.000
8	0.010	0.000	0.000	0.000	0.000	0.000	0.005	1.000	0.000
9	0.000	0.000	0.000	0.009	0.000	0.000	0.010	0.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.604	0.011	0.000	0.253	0.055	0.055	0.022	0.000	0.000
2	0.042	0.278	0.000	0.056	0.000	0.042	0.583	0.000	0.000
3	0.071	0.000	0.214	0.214	0.214	0.000	0.286	0.000	0.000
4	0.255	0.009	0.000	0.664	0.018	0.009	0.045	0.000	0.000
5	0.154	0.051	0.000	0.051	0.462	0.051	0.231	0.000	0.000
6	0.114	0.000	0.000	0.023	0.045	0.682	0.136	0.000	0.000
7	0.000	0.098	0.013	0.007	0.013	0.000	0.869	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.333	0.333	0.000
9	0.000	0.000	0.000	0.167	0.000	0.000	0.333	0.000	0.500

4.3.2.3. Feature Importance, Correctly Classified point

In [100]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[2.400e-03 3.000e-04 1.000e-04 3.700e-03 5.100e-02 9.421e-01 1.000e-04
  3.000e-04 0.000e+00]]
Actual Class : 6
--------------------------------------------------
136 Text feature [amplification] present in test data point [True]
169 Text feature [women] present in test data point [True]
232 Text feature [african] present in test data point [True]
326 Text feature [homozygotes] present in test data point [True]
363 Text feature [ethnic groups] present in test data point [True]
430 Text feature [ancestry] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [101]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[4.860e-02 4.650e-02 4.900e-03 6.380e-02 2.820e-02 8.000e-03 7.965e-01
  3.000e-03 6.000e-04]]
Actual Class : 6
--------------------------------------------------
111 Text feature [domain abd] present in test data point [True]
159 Text feature [binding domain abd] present in test data point [True]
212 Text feature [sensitive mek] present in test data point [True]
229 Text feature [tumor recurrence] present in test data point [True]
233 Text feature [activated] present in test data point [True]
237 Text feature [levels phosphorylated] present in test data point [True]
282 Text feature [nsh2 domain] present in test data point [True]
286 Text feature [overexpression] present in test data point [True]
318 Text feature [levels significantly] present in test data point [True]
337 Text feature [genetic aberrations] present in test data point [True]
373 Text feature [nsh2] present in test data point [True]
425 Text feature [activation] present in test data point [True]
438 Text feature [molecular features] present in test data point [True]
456 Text feature [determine effects] present in test data point [True]
457 Text feature [pik3r2] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

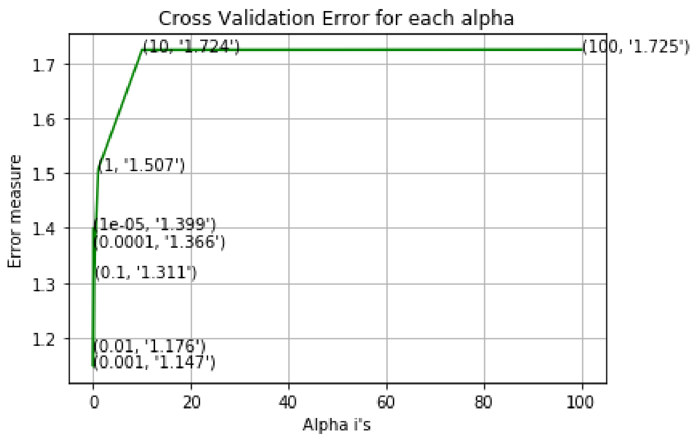
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [102]:

```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modul
es/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', 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', rando
m_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, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.3994920736974763
for C = 0.0001
Log Loss : 1.3664062895311295
for C = 0.001
Log Loss : 1.1474596321522532
for C = 0.01
Log Loss : 1.1763005322657383
for C = 0.1
Log Loss : 1.3111910656663983
for C = 1
Log Loss : 1.5069999789691324
for C = 10
Log Loss : 1.7243128829252075
for C = 100
Log Loss : 1.7247546365248712
```



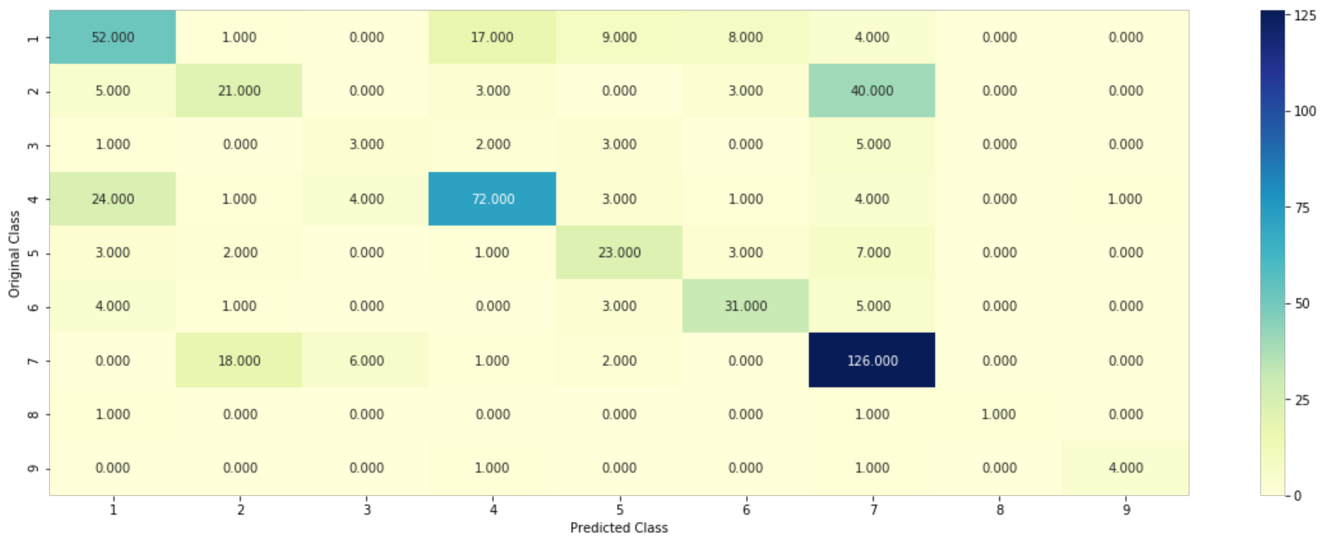
```
For values of best alpha =  0.001 The train log loss is: 0.6884210569219859
For values of best alpha =  0.001 The cross validation log loss is: 1.1474596321522532
For values of best alpha =  0.001 The test log loss is: 1.1353767299053383
```

4.4.2. Testing model with best hyper parameters

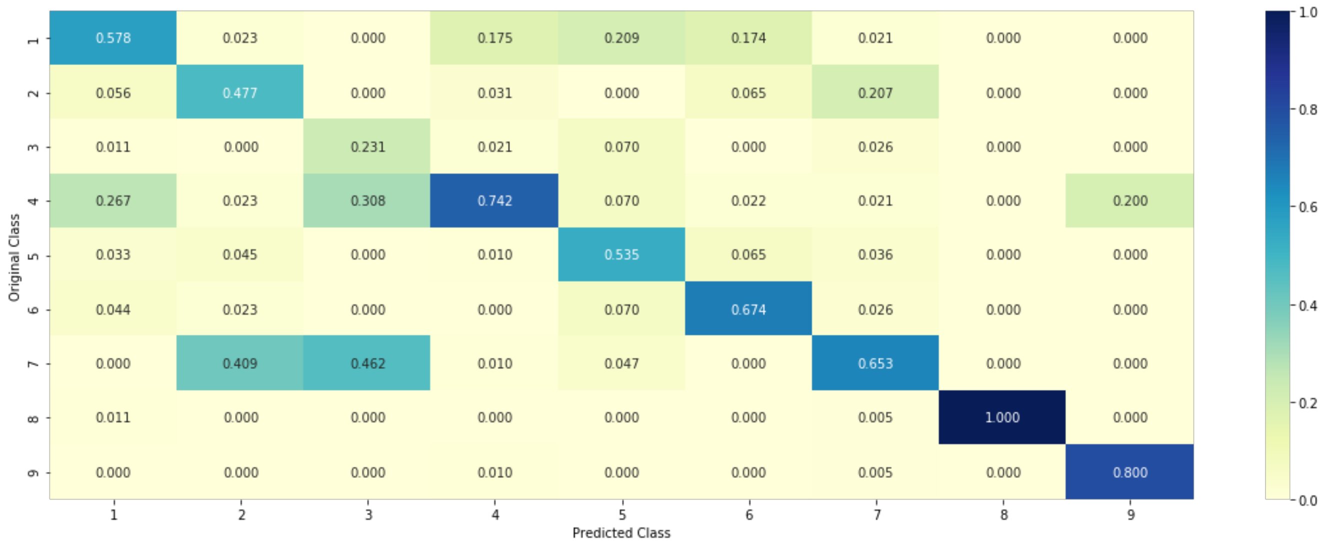
In [103]:

```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modul
es/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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


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

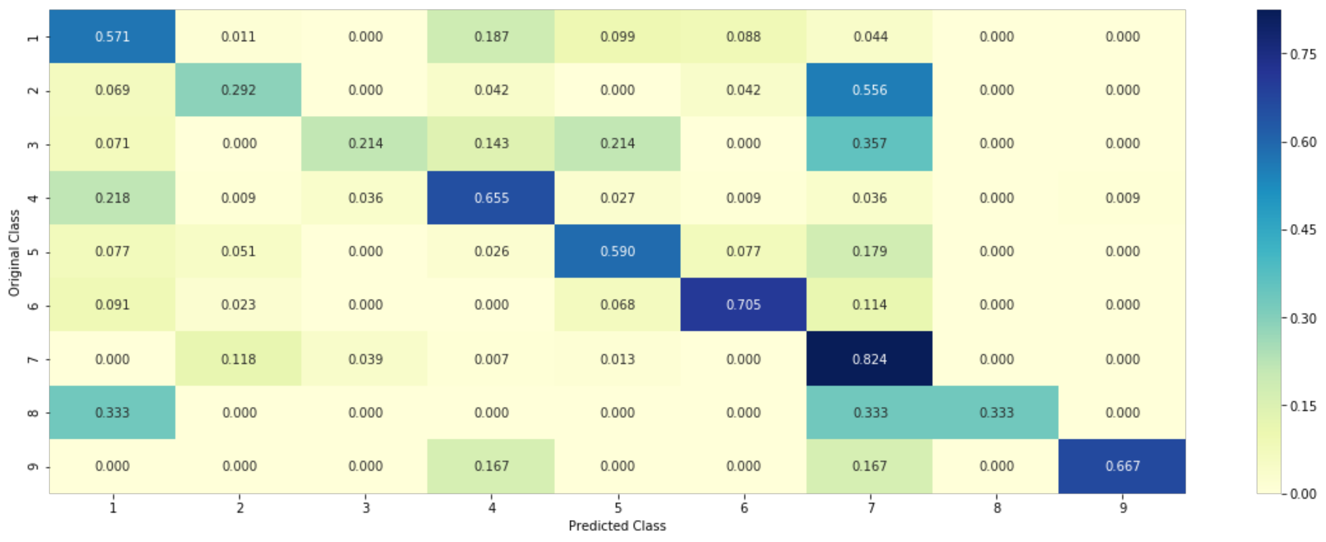
```
Log loss : 1.1474596321522532
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [104]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0031 0.0238 0.0041 0.0077 0.0408 0.8919 0.0242 0.0019 0.0025]]
Actual Class : 6
--------------------------------------------------
27 Text feature [amplification] present in test data point [True]
122 Text feature [women] present in test data point [True]
178 Text feature [ancestry] present in test data point [True]
209 Text feature [given family] present in test data point [True]
319 Text feature [family history ovarian cancer] present in test data point [True]
334 Text feature [susceptibility genes brca1] present in test data point [True]
338 Text feature [family history ovarian] present in test data point [True]
343 Text feature [ethnic groups] present in test data point [True]
423 Text feature [history ovarian cancer] present in test data point [True]
446 Text feature [history ovarian] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [105]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0727 0.0472 0.0145 0.1051 0.0471 0.0196 0.6881 0.0017 0.0039]]
Actual Class : 6
--------------------------------------------------
170 Text feature [domain abd] present in test data point [True]
171 Text feature [braf] present in test data point [True]
186 Text feature [tumor recurrence] present in test data point [True]
194 Text feature [nsh2 domain] present in test data point [True]
226 Text feature [binding domain abd] present in test data point [True]
236 Text feature [sensitive mek] present in test data point [True]
247 Text feature [nsh2] present in test data point [True]
269 Text feature [genetic aberrations] present in test data point [True]
285 Text feature [abd] present in test data point [True]
312 Text feature [mutations abd] present in test data point [True]
350 Text feature [determine effects] present in test data point [True]
364 Text feature [molecular features] present in test data point [True]
```

```
385 Text feature [levels significantly] present in test data point [True]
403 Text feature [adaptor binding] present in test data point [True]
412 Text feature [ish2] present in test data point [True]
452 Text feature [overexpression] present in test data point [True]
484 Text feature [helical domain] present in test data point [True]
485 Text feature [co mutation] present in test data point [True]
Out of the top  500  features  18 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [106]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, 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_dep
th[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

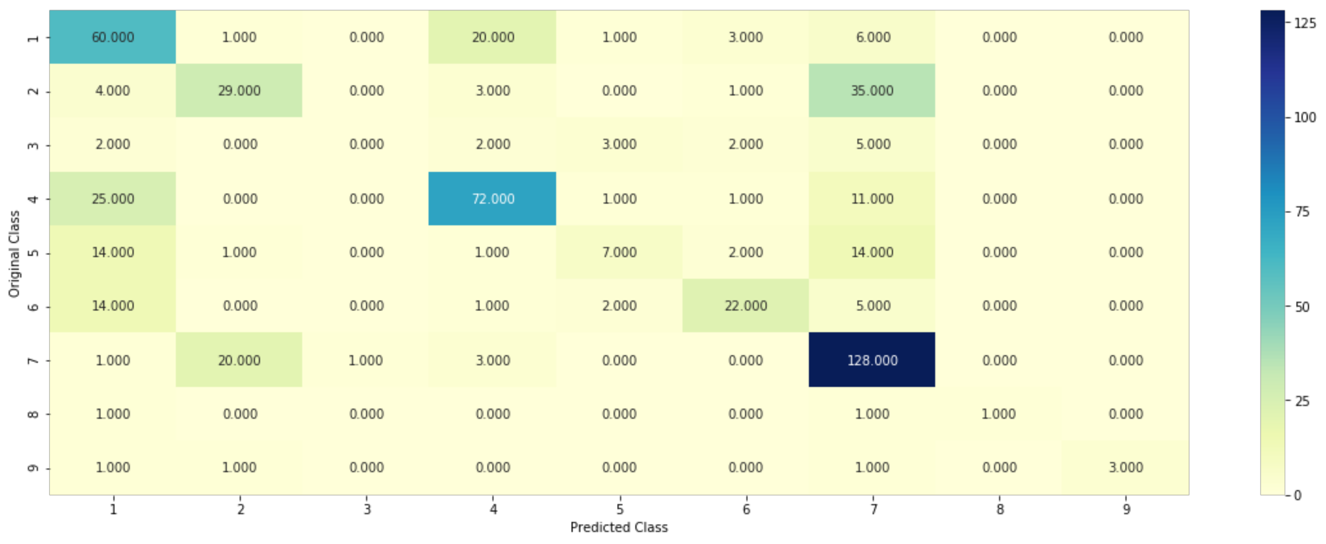
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2314149597598345
for n_estimators = 100 and max depth =  10
Log Loss : 1.1565976456184852
for n_estimators = 200 and max depth =  5
Log Loss : 1.2253320641506007
for n_estimators = 200 and max depth =  10
Log Loss : 1.1506179460401127
for n_estimators = 500 and max depth =  5
Log Loss : 1.2234311560669178
for n_estimators = 500 and max depth =  10
Log Loss : 1.1403698035073637
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2241519152421059
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1391594569447907
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2280777002215673
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1403116078092197
For values of best estimator =  1000 The train log loss is: 0.7002628108255472
For values of best estimator =  1000 The cross validation log loss is: 1.1391594569447907
For values of best estimator =  1000 The test log loss is: 1.1371289863414376
```

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

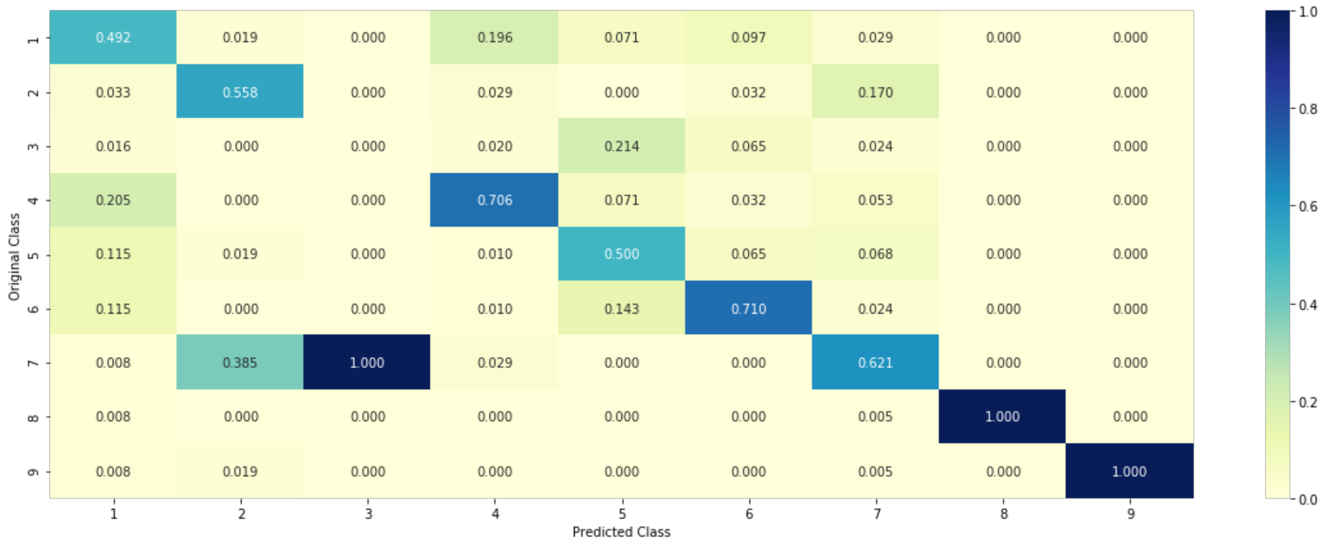
In [107]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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

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

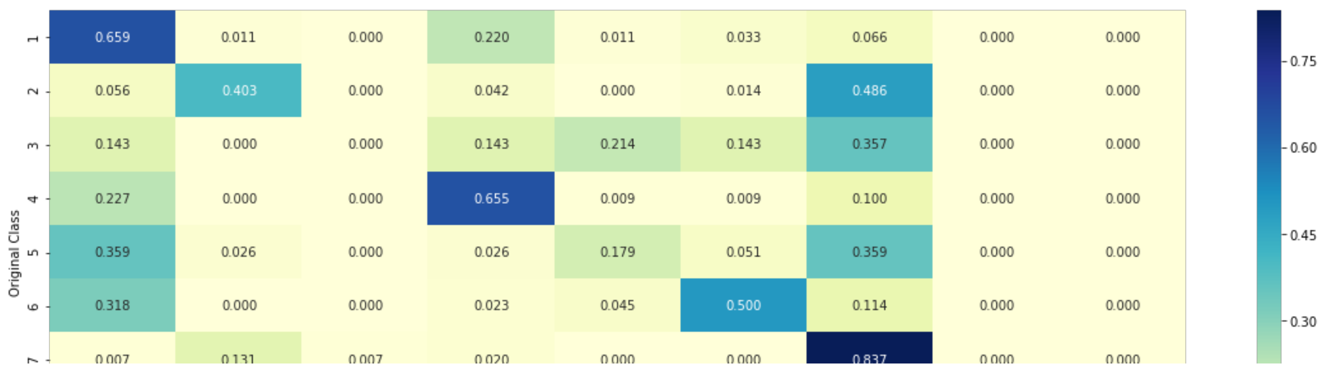
```
Log loss : 1.1391594569447907
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```

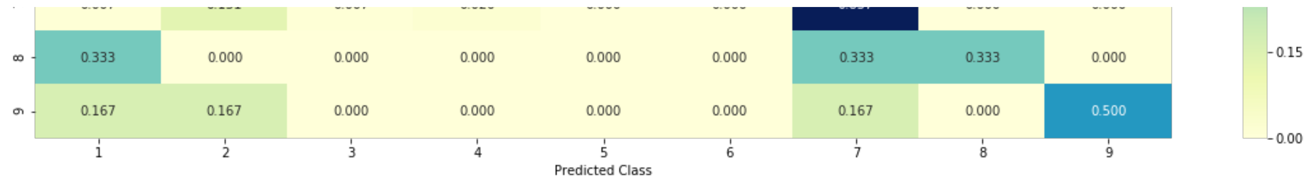


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [108]:

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

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0155 0.0094 0.0099 0.0097 0.1239 0.8183 0.0085 0.0019 0.0028]]
Actual Class : 6
--------------------------------------------------
5 Text feature [activation] present in test data point [True]
12 Text feature [function] present in test data point [True]
14 Text feature [missense] present in test data point [True]
21 Text feature [pathogenic] present in test data point [True]
23 Text feature [cells] present in test data point [True]
28 Text feature [loss] present in test data point [True]
42 Text feature [yeast] present in test data point [True]
45 Text feature [functional] present in test data point [True]
52 Text feature [patients] present in test data point [True]
57 Text feature [missense mutations] present in test data point [True]
61 Text feature [deleterious] present in test data point [True]
62 Text feature [therapeutic] present in test data point [True]
67 Text feature [neutral] present in test data point [True]
72 Text feature [variants] present in test data point [True]
73 Text feature [families] present in test data point [True]
78 Text feature [missense variants] present in test data point [True]
83 Text feature [alignments] present in test data point [True]
88 Text feature [repair] present in test data point [True]
93 Text feature [brca1] present in test data point [True]
94 Text feature [breast ovarian cancer] present in test data point [True]
97 Text feature [dna binding] present in test data point [True]
98 Text feature [dose] present in test data point [True]
99 Text feature [protein] present in test data point [True]
100 Text feature [potential] present in test data point [True]
102 Text feature [functions] present in test data point [True]
104 Text feature [family history] present in test data point [True]
113 Text feature [i2285v] present in test data point [True]
115 Text feature [proteins] present in test data point [True]
116 Text feature [bic] present in test data point [True]
117 Text feature [brca2] present in test data point [True]
121 Text feature [functional assays] present in test data point [True]
123 Text feature [predicted deleterious] present in test data point [True]
125 Text feature [clinical] present in test data point [True]
136 Text feature [brca1 mutation] present in test data point [True]
137 Text feature [cancer information] present in test data point [True]
140 Text feature [pathogenicity] present in test data point [True]
```

```
146 Text feature [pathogenicity] present in test data point [True]
147 Text feature [brca1 brca2] present in test data point [True]
157 Text feature [brct] present in test data point [True]
158 Text feature [response] present in test data point [True]
162 Text feature [family personal] present in test data point [True]
163 Text feature [mutations brca1] present in test data point [True]
168 Text feature [variants 33] present in test data point [True]
170 Text feature [brca1 missense] present in test data point [True]
174 Text feature [breast cancer information core] present in test data point [True]
175 Text feature [benefit] present in test data point [True]
184 Text feature [trans deleterious mutation] present in test data point [True]
185 Text feature [odds] present in test data point [True]
186 Text feature [brca1 mutations] present in test data point [True]
194 Text feature [brct domain] present in test data point [True]
205 Text feature [personal history] present in test data point [True]
210 Text feature [carriers known] present in test data point [True]
212 Text feature [sensitivity] present in test data point [True]
214 Text feature [probabilities] present in test data point [True]
217 Text feature [one two] present in test data point [True]
218 Text feature [gvgd] present in test data point [True]
219 Text feature [useful] present in test data point [True]
222 Text feature [variant brca1] present in test data point [True]
223 Text feature [individuals known] present in test data point [True]
224 Text feature [likelihood] present in test data point [True]
225 Text feature [known deleterious] present in test data point [True]
230 Text feature [amplification] present in test data point [True]
232 Text feature [exonic splice] present in test data point [True]
235 Text feature [variant] present in test data point [True]
236 Text feature [classify] present in test data point [True]
238 Text feature [independent] present in test data point [True]
239 Text feature [splice] present in test data point [True]
240 Text feature [personal family] present in test data point [True]
243 Text feature [patient] present in test data point [True]
246 Text feature [early onset] present in test data point [True]
247 Text feature [bic database] present in test data point [True]
249 Text feature [uncertain] present in test data point [True]
250 Text feature [family] present in test data point [True]
251 Text feature [expression] present in test data point [True]
254 Text feature [recently] present in test data point [True]
255 Text feature [domain brca1] present in test data point [True]
256 Text feature [ovarian cancer] present in test data point [True]
260 Text feature [majority variants] present in test data point [True]
262 Text feature [cosegregation] present in test data point [True]
270 Text feature [breast ovarian] present in test data point [True]
274 Text feature [cancer risk] present in test data point [True]
275 Text feature [clinical interpretation] present in test data point [True]
278 Text feature [neutrality] present in test data point [True]
279 Text feature [relevance brca1] present in test data point [True]
281 Text feature [predicted] present in test data point [True]
282 Text feature [personal family history] present in test data point [True]
284 Text feature [variants brca1] present in test data point [True]
285 Text feature [achieved] present in test data point [True]
296 Text feature [wild type] present in test data point [True]
297 Text feature [unclassified] present in test data point [True]
300 Text feature [null] present in test data point [True]
303 Text feature [model brca1] present in test data point [True]
306 Text feature [expected] present in test data point [True]
308 Text feature [binding domain amino] present in test data point [True]
310 Text feature [cancer information core] present in test data point [True]
311 Text feature [co] present in test data point [True]
313 Text feature [disrupt] present in test data point [True]
314 Text feature [final data] present in test data point [True]
330 Text feature [terminal] present in test data point [True]
332 Text feature [dna binding domain] present in test data point [True]
333 Text feature [using data] present in test data point [True]
337 Text feature [using approach] present in test data point [True]
340 Text feature [deleterious variants] present in test data point [True]
343 Text feature [also useful] present in test data point [True]
346 Text feature [myriad genetic] present in test data point [True]
349 Text feature [truncating] present in test data point [True]
351 Text feature [11] present in test data point [True]
352 Text feature [variants predicted] present in test data point [True]
354 Text feature [000 individuals] present in test data point [True]
357 Text feature [ethnic populations] present in test data point [True]
358 Text feature [absence] present in test data point [True]
360 Text feature [variant classified deleterious] present in test data point [True]
362 Text feature [results] present in test data point [True]
365 Text feature [binding] present in test data point [True]
```

```
365 Text feature [binding] present in test data point [True]
375 Text feature [affected] present in test data point [True]
377 Text feature [risk] present in test data point [True]
379 Text feature [structure] present in test data point [True]
380 Text feature [brca1 figure] present in test data point [True]
384 Text feature [sequence alignments] present in test data point [True]
385 Text feature [carriers] present in test data point [True]
386 Text feature [higher] present in test data point [True]
390 Text feature [brct domains] present in test data point [True]
393 Text feature [individuals known deleterious] present in test data point [True]
394 Text feature [information core] present in test data point [True]
397 Text feature [model] present in test data point [True]
402 Text feature [57] present in test data point [True]
403 Text feature [carrying brca1] present in test data point [True]
406 Text feature [carry deleterious mutation] present in test data point [True]
409 Text feature [wild type brca2] present in test data point [True]
412 Text feature [missense substitutions] present in test data point [True]
414 Text feature [p142h] present in test data point [True]
415 Text feature [presence] present in test data point [True]
416 Text feature [ivs13] present in test data point [True]
421 Text feature [structural] present in test data point [True]
423 Text feature [using] present in test data point [True]
426 Text feature [affect splicing] present in test data point [True]
430 Text feature [based functional] present in test data point [True]
431 Text feature [provide overall] present in test data point [True]
432 Text feature [12] present in test data point [True]
433 Text feature [transcriptional] present in test data point [True]
434 Text feature [brca mutation] present in test data point [True]
435 Text feature [core] present in test data point [True]
439 Text feature [vus observed] present in test data point [True]
443 Text feature [western] present in test data point [True]
444 Text feature [within] present in test data point [True]
446 Text feature [probability] present in test data point [True]
450 Text feature [history available] present in test data point [True]
455 Text feature [myriad] present in test data point [True]
460 Text feature [classified deleterious] present in test data point [True]
461 Text feature [confer] present in test data point [True]
463 Text feature [33] present in test data point [True]
466 Text feature [sequence variant] present in test data point [True]
469 Text feature [full length] present in test data point [True]
470 Text feature [breast cancer information] present in test data point [True]
471 Text feature [likely neutral] present in test data point [True]
472 Text feature [s1172l] present in test data point [True]
474 Text feature [combined] present in test data point [True]
476 Text feature [18 brca1] present in test data point [True]
478 Text feature [l1764p] present in test data point [True]
479 Text feature [minority ethnic] present in test data point [True]
480 Text feature [sequence] present in test data point [True]
481 Text feature [conservative substitution] present in test data point [True]
482 Text feature [brca1 missense mutations] present in test data point [True]
483 Text feature [useful classification] present in test data point [True]
489 Text feature [deleterious mutations] present in test data point [True]
491 Text feature [e3] present in test data point [True]
493 Text feature [increased] present in test data point [True]
495 Text feature [brca2 dna] present in test data point [True]
498 Text feature [length] present in test data point [True]
Out of the top  500  features  168 are present in query point
```

4.5.3.2. Incorrectly Classified point

In [109]:

```
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_i
ndex]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[
test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0609 0.1212 0.0186 0.0827 0.0489 0.0387 0.6178 0.0047 0.0065]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [tyrosine] present in test data point [True]
1 Text feature [kinase domain] present in test data point [True]
2 Text feature [activated] present in test data point [True]
3 Text feature [activating mutations] present in test data point [True]
4 Text feature [kinase] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [activating] present in test data point [True]
8 Text feature [inhibitors] present in test data point [True]
9 Text feature [kinase activity] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [phosphorylation] present in test data point [True]
15 Text feature [inhibitor] present in test data point [True]
16 Text feature [tumor suppressor] present in test data point [True]
17 Text feature [oncogenic] present in test data point [True]
18 Text feature [signaling] present in test data point [True]
22 Text feature [inhibition] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [activate] present in test data point [True]
26 Text feature [trials] present in test data point [True]
27 Text feature [treatment] present in test data point [True]
28 Text feature [loss] present in test data point [True]
29 Text feature [growth] present in test data point [True]
30 Text feature [downstream] present in test data point [True]
31 Text feature [receptor] present in test data point [True]
33 Text feature [receptor tyrosine] present in test data point [True]
35 Text feature [independent growth] present in test data point [True]
36 Text feature [ba] present in test data point [True]
38 Text feature [transforming] present in test data point [True]
39 Text feature [ba f3 cells] present in test data point [True]
41 Text feature [lines] present in test data point [True]
45 Text feature [functional] present in test data point [True]
46 Text feature [ba f3] present in test data point [True]
49 Text feature [therapy] present in test data point [True]
52 Text feature [patients] present in test data point [True]
53 Text feature [akt] present in test data point [True]
54 Text feature [treated] present in test data point [True]
55 Text feature [downstream signaling] present in test data point [True]
62 Text feature [therapeutic] present in test data point [True]
63 Text feature [growth factor] present in test data point [True]
65 Text feature [f3 cells] present in test data point [True]
68 Text feature [f3] present in test data point [True]
69 Text feature [erk1] present in test data point [True]
72 Text feature [variants] present in test data point [True]
79 Text feature [clinical trials] present in test data point [True]
80 Text feature [kinases] present in test data point [True]
81 Text feature [growth factor receptor] present in test data point [True]
84 Text feature [cell lines] present in test data point [True]
87 Text feature [stability] present in test data point [True]
88 Text feature [repair] present in test data point [True]
92 Text feature [cell] present in test data point [True]
99 Text feature [protein] present in test data point [True]
102 Text feature [functions] present in test data point [True]
103 Text feature [transformation] present in test data point [True]
108 Text feature [mitogen activated] present in test data point [True]
110 Text feature [mek] present in test data point [True]
111 Text feature [loss pten] present in test data point [True]
114 Text feature [inhibited] present in test data point [True]
115 Text feature [proteins] present in test data point [True]
119 Text feature [resistant] present in test data point [True]
122 Text feature [mitogen] present in test data point [True]
125 Text feature [clinical] present in test data point [True]
127 Text feature [protein stability] present in test data point [True]
128 Text feature [harboring] present in test data point [True]
131 Text feature [survival] present in test data point [True]
132 Text feature [inactivation] present in test data point [True]
134 Text feature [tyrosine kinases] present in test data point [True]
135 Text feature [kinase domains] present in test data point [True]
142 Text feature [pten mutations] present in test data point [True]
144 Text feature [pten protein] present in test data point [True]
146 Text feature [mutations pten] present in test data point [True]
152 Text feature [il] present in test data point [True]
156 Text feature [sensitive] present in test data point [True]
```

```
165 Text feature [protein kinase] present in test data point [True]
166 Text feature [pathway] present in test data point [True]
171 Text feature [therapies] present in test data point [True]
175 Text feature [benefit] present in test data point [True]
181 Text feature [metastatic] present in test data point [True]
191 Text feature [pten] present in test data point [True]
196 Text feature [gain function] present in test data point [True]
197 Text feature [enhanced] present in test data point [True]
202 Text feature [independence] present in test data point [True]
209 Text feature [factor receptor] present in test data point [True]
212 Text feature [sensitivity] present in test data point [True]
219 Text feature [useful] present in test data point [True]
220 Text feature [effective] present in test data point [True]
228 Text feature [binding pten] present in test data point [True]
238 Text feature [independent] present in test data point [True]
243 Text feature [patient] present in test data point [True]
251 Text feature [expression] present in test data point [True]
252 Text feature [phosphotyrosine] present in test data point [True]
259 Text feature [tumors] present in test data point [True]
261 Text feature [ras] present in test data point [True]
269 Text feature [phosphorylated] present in test data point [True]
273 Text feature [weeks] present in test data point [True]
276 Text feature [activated protein kinase] present in test data point [True]
281 Text feature [predicted] present in test data point [True]
289 Text feature [mitogen activated protein] present in test data point [True]
293 Text feature [harbored] present in test data point [True]
296 Text feature [wild type] present in test data point [True]
302 Text feature [receptor tyrosine kinases] present in test data point [True]
306 Text feature [expected] present in test data point [True]
311 Text feature [co] present in test data point [True]
312 Text feature [phosphatase] present in test data point [True]
313 Text feature [disrupt] present in test data point [True]
320 Text feature [activating mutation] present in test data point [True]
322 Text feature [cell line] present in test data point [True]
326 Text feature [molecular] present in test data point [True]
330 Text feature [terminal] present in test data point [True]
331 Text feature [somatic mutations] present in test data point [True]
350 Text feature [activated protein] present in test data point [True]
351 Text feature [11] present in test data point [True]
358 Text feature [absence] present in test data point [True]
359 Text feature [gain function mutations] present in test data point [True]
362 Text feature [results] present in test data point [True]
364 Text feature [progression] present in test data point [True]
365 Text feature [binding] present in test data point [True]
368 Text feature [retained] present in test data point [True]
371 Text feature [defects] present in test data point [True]
372 Text feature [factor] present in test data point [True]
377 Text feature [risk] present in test data point [True]
378 Text feature [complete loss] present in test data point [True]
381 Text feature [mapk] present in test data point [True]
383 Text feature [proliferation] present in test data point [True]
386 Text feature [higher] present in test data point [True]
387 Text feature [drugs] present in test data point [True]
397 Text feature [model] present in test data point [True]
404 Text feature [interleukin] present in test data point [True]
410 Text feature [wt] present in test data point [True]
415 Text feature [presence] present in test data point [True]
417 Text feature [membrane] present in test data point [True]
422 Text feature [co expression] present in test data point [True]
432 Text feature [12] present in test data point [True]
433 Text feature [transcriptional] present in test data point [True]
441 Text feature [experiments] present in test data point [True]
443 Text feature [western] present in test data point [True]
444 Text feature [within] present in test data point [True]
451 Text feature [1a] present in test data point [True]
458 Text feature [mutants] present in test data point [True]
459 Text feature [interaction] present in test data point [True]
463 Text feature [33] present in test data point [True]
474 Text feature [combined] present in test data point [True]
484 Text feature [signals] present in test data point [True]
485 Text feature [role] present in test data point [True]
486 Text feature [surface] present in test data point [True]
487 Text feature [mechanism] present in test data point [True]
488 Text feature [kras] present in test data point [True]
492 Text feature [ba f3 cell] present in test data point [True]
493 Text feature [increased] present in test data point [True]
494 Text feature [effect pten] present in test data point [True]
```

```
Out of the top  500  features  150 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [110]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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


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

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

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

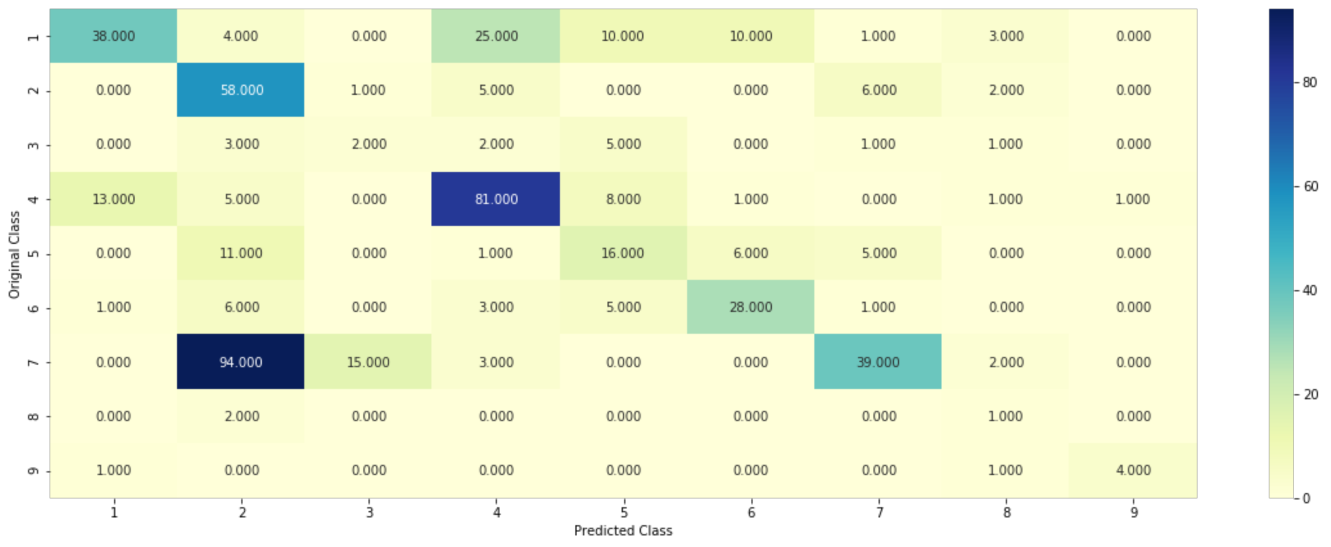
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.115807468495449
for n_estimators = 10 and max depth =  3
Log Loss : 1.880987021470951
for n_estimators = 10 and max depth =  5
Log Loss : 1.5208114591939845
for n_estimators = 10 and max depth =  10
Log Loss : 2.159416839044755
for n_estimators = 50 and max depth =  2
Log Loss : 1.7606914108165852
for n_estimators = 50 and max depth =  3
Log Loss : 1.4684144886744233
for n_estimators = 50 and max depth =  5
Log Loss : 1.4441650609842156
for n_estimators = 50 and max depth =  10
Log Loss : 1.7537827827013042
for n_estimators = 100 and max depth =  2
Log Loss : 1.5783248728005936
for n_estimators = 100 and max depth =  3
Log Loss : 1.453351834023957
for n_estimators = 100 and max depth =  5
Log Loss : 1.406642649745641
for n_estimators = 100 and max depth =  10
Log Loss : 1.8049496864596726
for n_estimators = 200 and max depth =  2
Log Loss : 1.6190726280020316
for n_estimators = 200 and max depth =  3
Log Loss : 1.4438681510920979
for n_estimators = 200 and max depth =  5
Log Loss : 1.4527470472663297
for n_estimators = 200 and max depth =  10
Log Loss : 1.7743624814385526
for n_estimators = 500 and max depth =  2
Log Loss : 1.6889838306117022
for n_estimators = 500 and max depth =  3
Log Loss : 1.5091203518929077
for n_estimators = 500 and max depth =  5
Log Loss : 1.4477524281617966
for n_estimators = 500 and max depth =  10
Log Loss : 1.7963611293720925
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6396991438230049
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5189711242467463
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4586401883886473
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7604238462483834
For values of best alpha =  100 The train log loss is: 0.05551133947636198
For values of best alpha =  100 The cross validation log loss is: 1.406642649745641
For values of best alpha =  100 The test log loss is: 1.3875321091912167
```

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

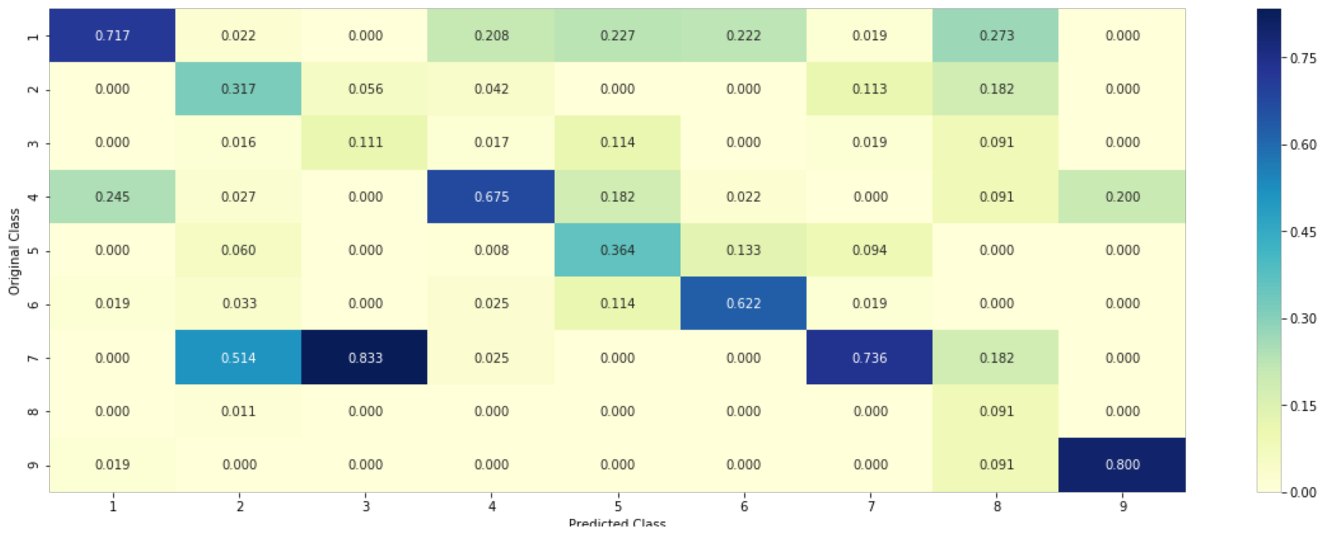
In [111]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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

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

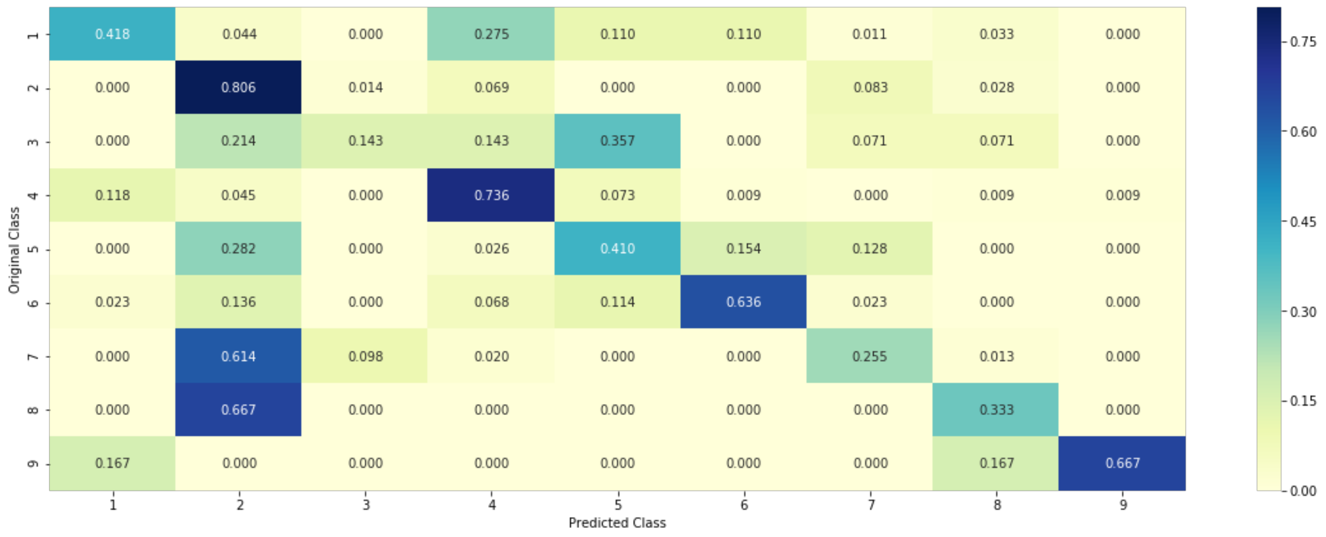
```
Log loss : 1.4066426497456408
Number of mis-classified points : 0.4981203007518797
-------------------- Confusion matrix --------------------
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [112]:

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


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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0056 0.0038 0.0065 0.0075 0.1295 0.8358 0.0021 0.0045 0.0046]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [113]:

```
test_point_index = 2
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point
_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0337 0.1545 0.2188 0.0868 0.036  0.0534 0.3519 0.0414 0.0234]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation 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 [114]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
del.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modul
es/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modul
es/generated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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


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

clf2 = SGDClassifier(alpha=0.001, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")
```

```
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.00001)
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_onehotCodi
ng))))
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_onehotC
oding))))
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_proba
s=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.p
redict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

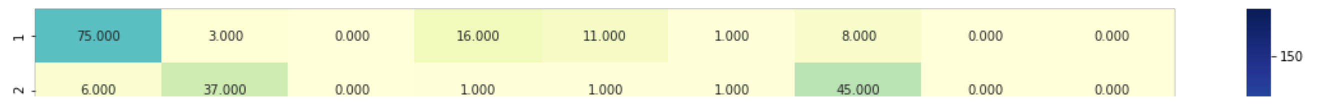
```
Logistic Regression :  Log Loss: 1.08
Support vector machines : Log Loss: 1.17
Naive Bayes : Log Loss: 1.44
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.176
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.016
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.467
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.179
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.350
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.623
```

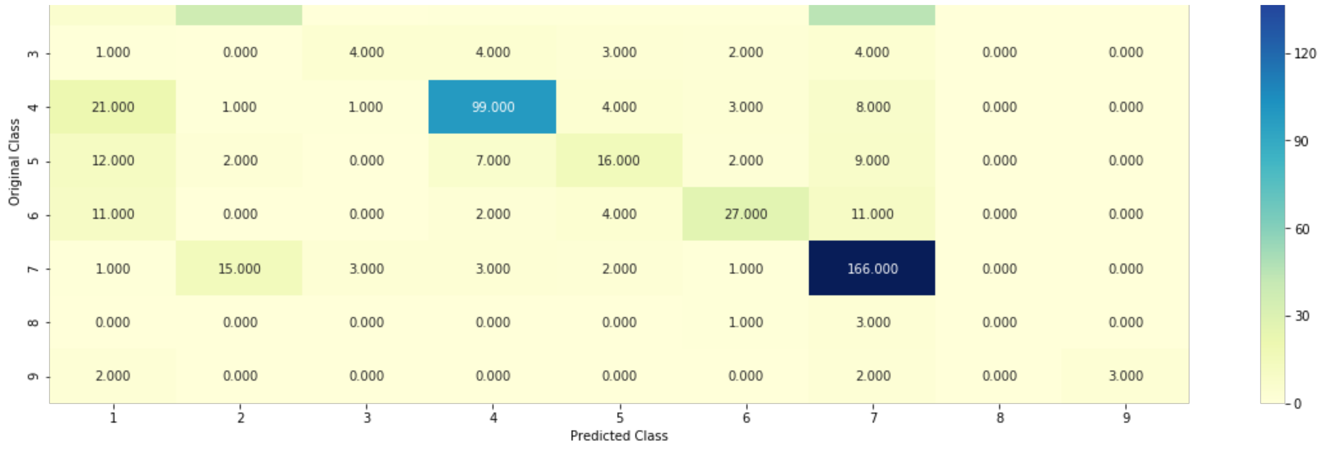
4.7.2 testing the model with the best hyper parameters

In [115]:

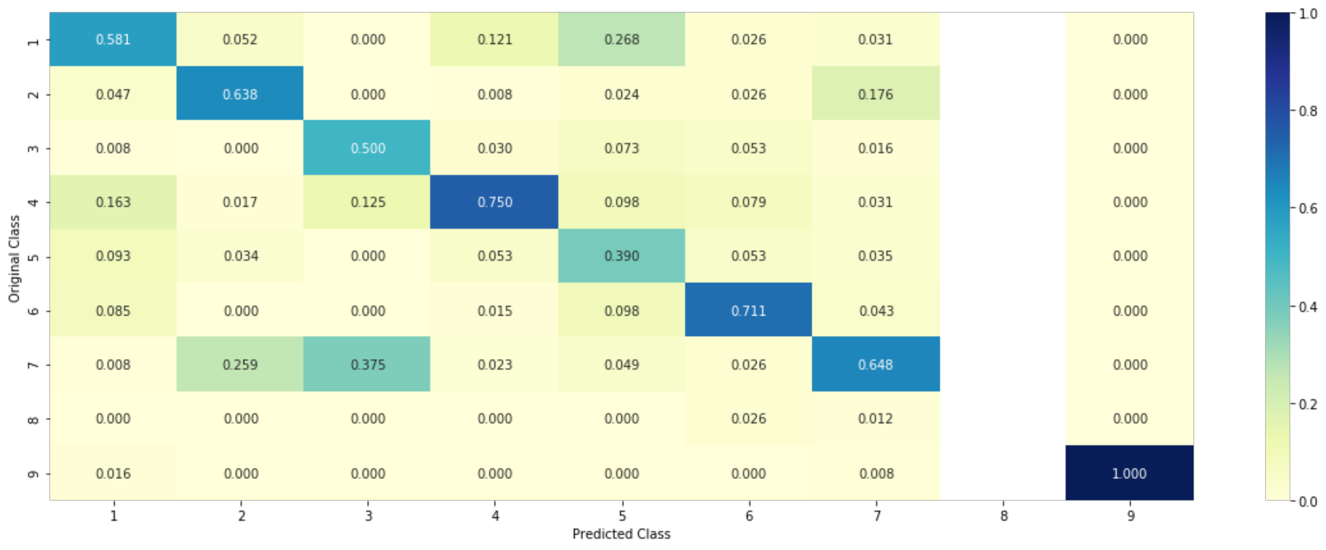
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=Tr
ue)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.5952703843542426
Log loss (CV) on the stacking classifier : 1.1788448673562748
Log loss (test) on the stacking classifier : 1.147020365275695
Number of missclassified point : 0.35789473684210527
-------------------- Confusion matrix --------------------
```

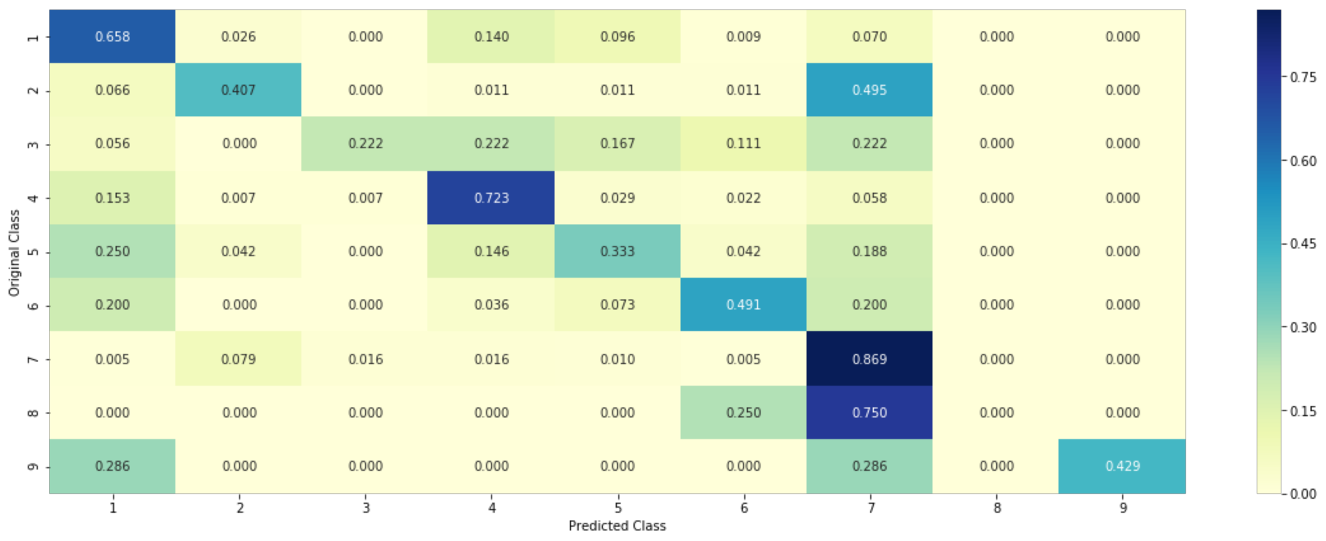




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

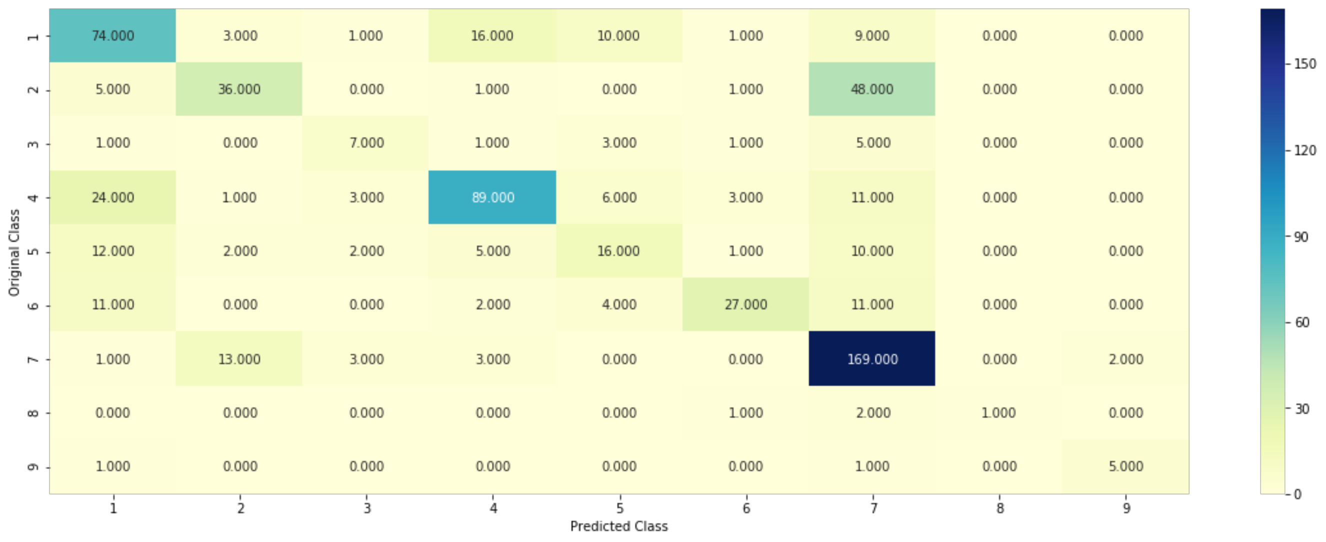


4.7.3 Maximum Voting classifier

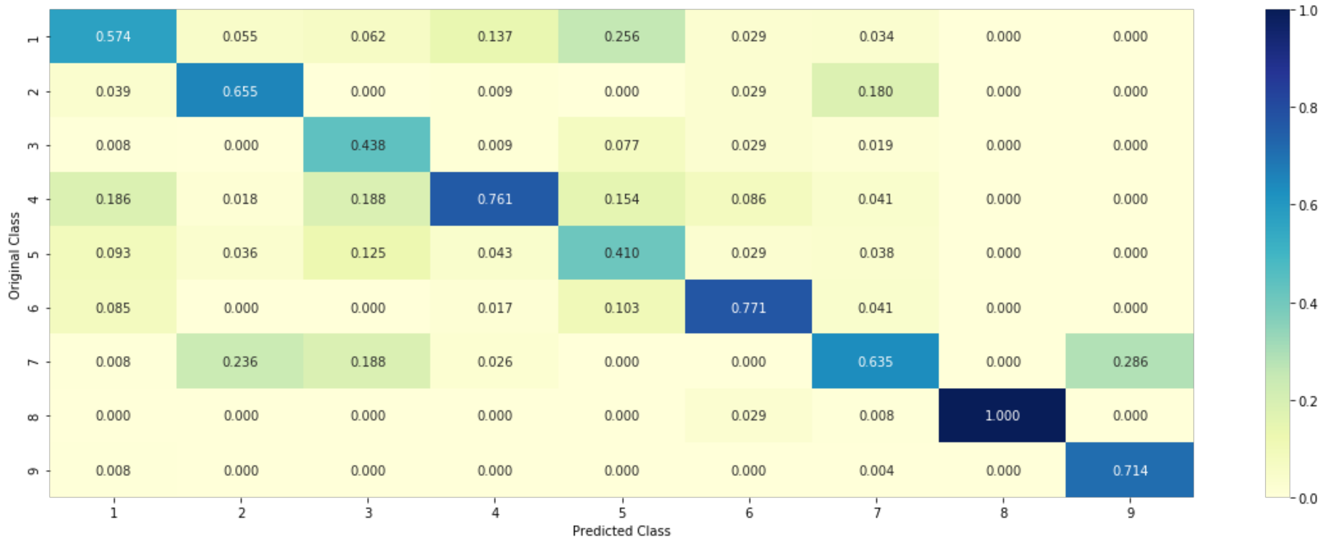
In [116]:

```
#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='sof
t')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehot
Coding)))
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_onehotCod
ing)))
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))
```

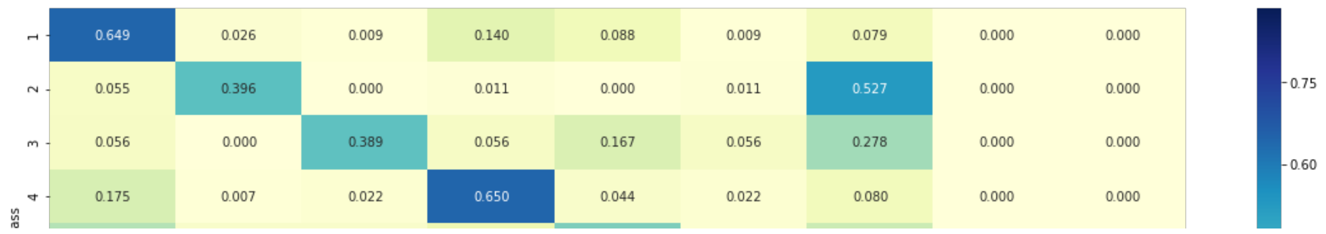
```
Log loss (train) on the VotingClassifier : 0.7259159808318318
Log loss (CV) on the VotingClassifier : 1.1196624070737842
Log loss (test) on the VotingClassifier : 1.1069324172547952
Number of missclassified point : 0.362406015037594
-------------------- Confusion matrix --------------------
```

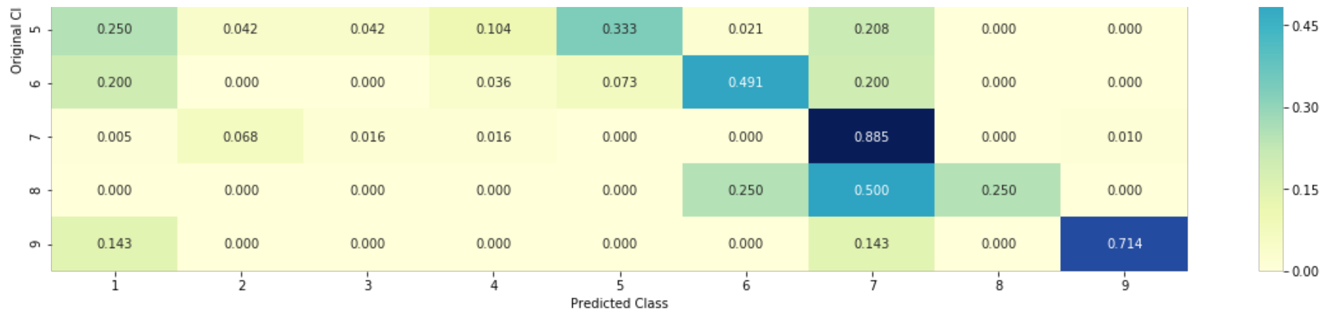


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





Conclusion

1. EDA
   - Read training data 'Gene and Variation'
   - Read training data 'Text'
   - Preprocessing 'Text' data
   - Handling missing 'Text' data and merge all features
2. Split data into 3 parts in ratio 64:20:16
3. See the distribution of each class in train, cv and test data
4. Train the Random Model (worst/dump) to find the upperbound logloss
5. Univariant analysis on Gene and Variant
   - What type of feature is?
   - How many categories are present
   - How they are distributed
   - How to featurize the feature: responsecoding with Laplace smoothing and onehotencoding (CountVectorizer)
   - How good is this feature in prediction y?
6. Univariant Analysis on Text data
   - Extract the number of words from each class
   - Featurized the text with onhotencoding(TdidfVectorizer) with unigra, bigram, trigram and 4-gram and taken max feature of 100000 and responsecoding with Laplace smoothing
   - How good is this feature in predicting y?
7. Stacked all features
8. Finally , train model and plot and observe confusion matrix, precision and recall
   - Naive Bayes (NB) and get feature importance
   - Logistic Regr (LR) with class balance and get feature importance
   - Logistic Regr (LR) without class balance and get feature importance
   - Linear SVM and get feature importance
   - Random Forest (RF) and get feature importance
   - StackModel (Logistic Regr, Linear SVM, Naive Bayes)+Logistic Regr(Final layer)
   - Maximum Voting Classifier

In [4]:

```python
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Feature", "Model", "Train logloss", "CV logloss", "Test logloss", "# of Misclassified
pts"]

print('OHE -> OneHotEncoding\nRC-> ResponseCoding with Laplace smoothing')

feature = ['OHE','RC','OHE','OHE','OHE','OHE','RC','OHE','OHE']
model_ = ['NB','KNN','LR (class balance)','LR (no class balance)','Linear SVM','RF','RF','StackModel+LR
','Maximum Voting Classifier']
train_loss = [1.06,0.61,0.56,0.56,0.69,0.7,0.05,0.59,0.72]
cv_loss = [1.43,1.11,1.09,1.09,1.15,1.14,1.41,1.18,1.12]
test_loss = [1.41,1.03,1.09,1.09,1.13,1.14,1.39,1.15,1.1]
mis_class = [0.462,0.389,0.374,0.368,0.374,0.394,0.498,0.358,0.362]

for i in range(0,9):
    x.add_row([feature[i],model_[i],train_loss[i],cv_loss[i],test_loss[i],mis_class[i]])

print(x)

print('\n\n1. From the observation above, LR(no class balance) are perform better than others.')
print('2. We cannot chose Maximum Voting Classifier because as per the business constraints, this model
is not intrepretable')
```

```
OHE -> OneHotEncoding
RC-> ResponseCoding with Laplace smoothing
+---------+---------------------------+---------------+------------+--------------+--------------------
----+
| Feature |           Model           | Train logloss | CV logloss | Test logloss | # of Misclassified
pts |
+---------+---------------------------+---------------+------------+--------------+--------------------
----+
|   OHE   |             NB            |      1.06     |    1.43    |     1.41     |         0.462
|
|    RC   |            KNN            |      0.61     |    1.11    |     1.03     |         0.389
|
|   OHE   |     LR (class balance)    |      0.56     |    1.09    |     1.09     |         0.374
|
|   OHE   |   LR (no class balance)   |      0.56     |    1.09    |     1.09     |         0.368
|
|   OHE   |         Linear SVM        |      0.69     |    1.15    |     1.13     |         0.374
|
|   OHE   |             RF            |      0.7      |    1.14    |     1.14     |         0.394
|
|    RC   |             RF            |      0.05     |    1.41    |     1.39     |         0.498
|
|   OHE   |       StackModel+LR       |      0.59     |    1.18    |     1.15     |         0.358
|
|   OHE   | Maximum Voting Classifier |      0.72     |    1.12    |     1.1      |         0.362
|
+---------+---------------------------+---------------+------------+--------------+--------------------
----+


1. From the observation above, LR(no class balance) are perform better than others.
2. We cannot chose Maximum Voting Classifier because as per the business constraints, this model is not
intrepretable
```

In [ ]:

```

```