# 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 [27]:

```
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 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
```

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

In [28]:

```
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[28]:

| | ID | Gene | Variation | Class |
|---|---|---|---|---|
| **0** | 0 | FAM58A | Truncating Mutations | 1 |

| | ID | Gene | Variation | Class |
|---|---|---|---|---|
| **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 [29]:

```
# 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[29]:

| | 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 [30]:

```
# 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 [31]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 212.402769 seconds
```

In [32]:

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

Out[32]:

| | 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 [33]:

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

Out[33]:

| | 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 [38]:

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

In [39]:

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

Out[39]:

| | ID | Gene | Variation | Class | TEXT |
|---|---|---|---|---|---|

| | 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 [40]:

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

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

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

In [41]:

```
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])
```

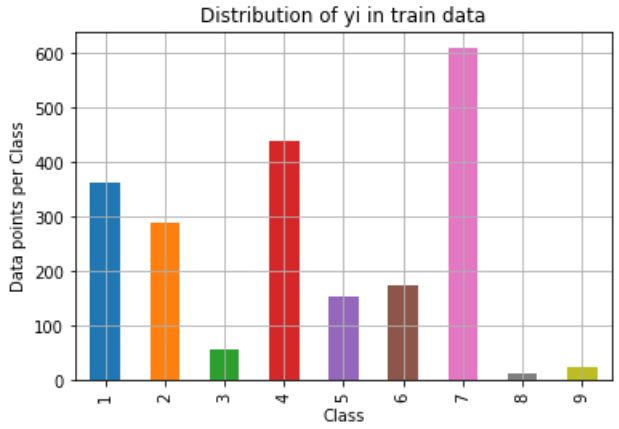
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

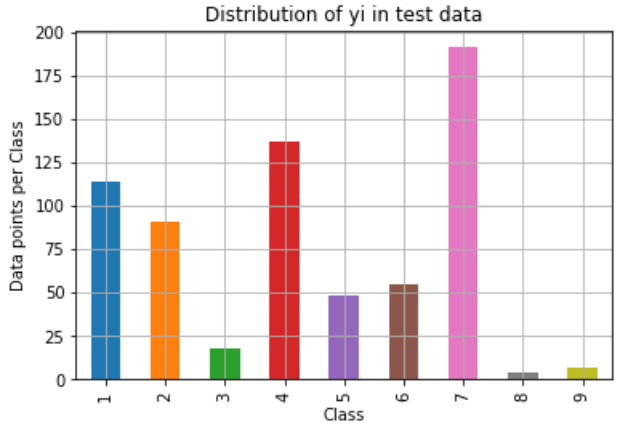
In [42]:

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

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

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

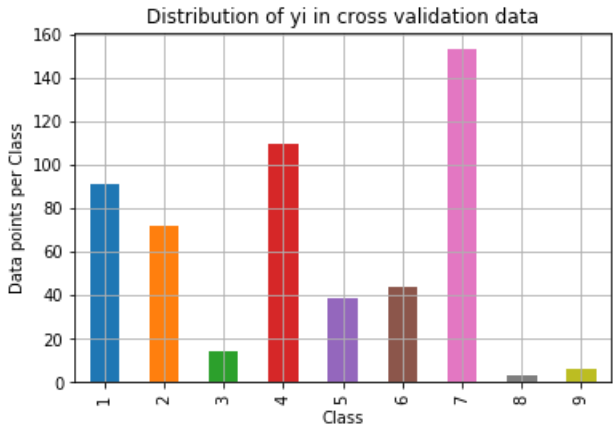



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 [43]:

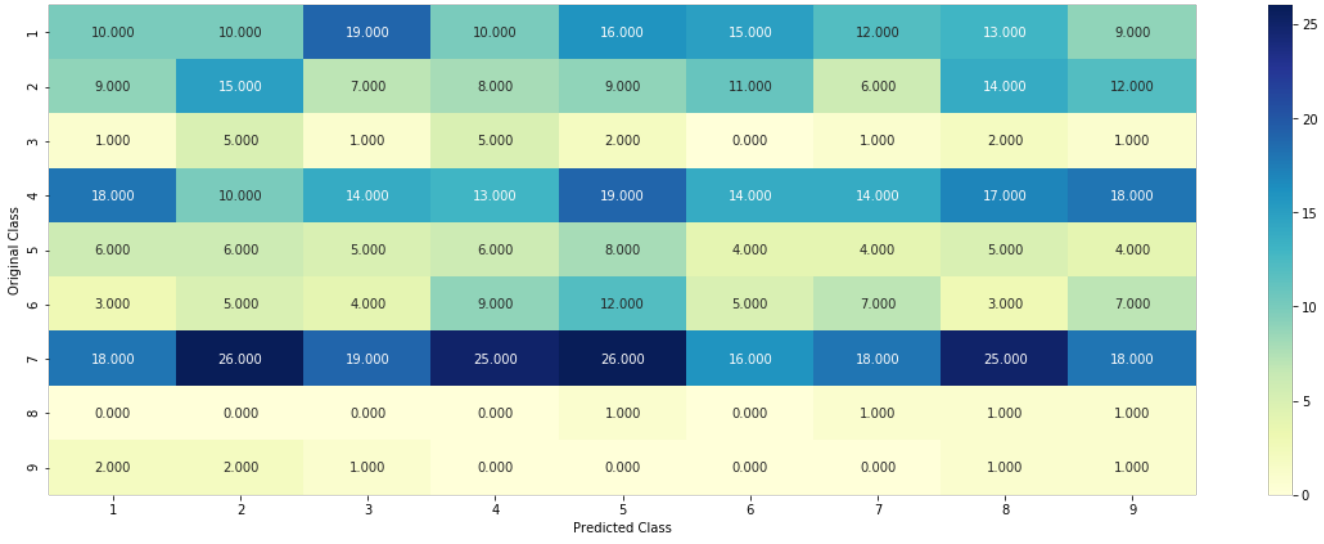
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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()
```

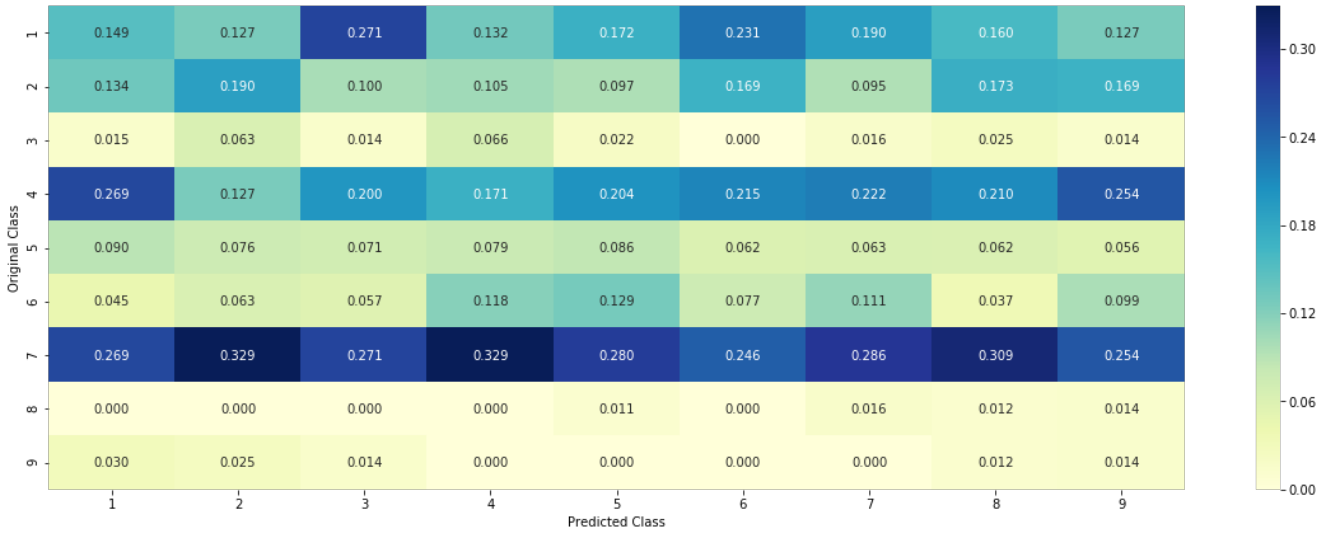
In [44]:

```
# 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)
```

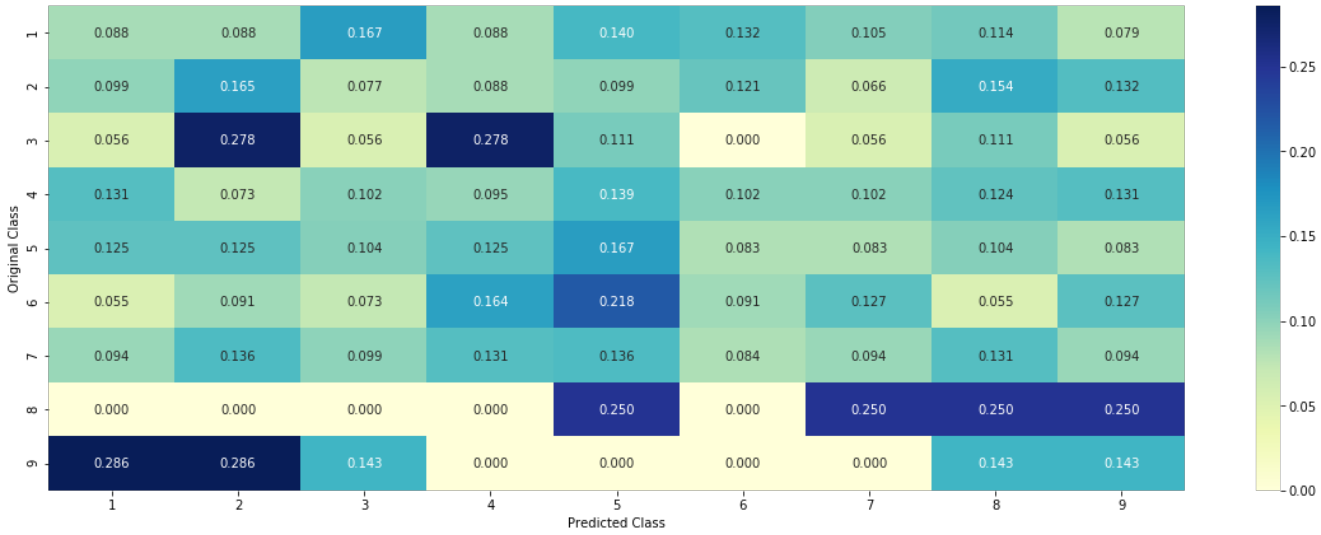
```
Log loss on Cross Validation Data using Random Model 2.50588112484003
Log loss on Test Data using Random Model 2.4952621487325057
-------------------- Confusion matrix --------------------
```




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




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




## 3.3 Univariate Analysis

In [45]:

```
# 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 / nu
mber 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
```

```
    #         TP53         100
    #         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 o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
```

```
0.0666666666666666, 0.0666666666666666],
    #       ...
    #     }
    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 da
ta
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

### 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 [48]:

```
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 : 228
BRCA1     163
TP53      106
PTEN       90
EGFR       85
BRCA2      75
BRAF       67
KIT        55
ALK        47
ERBB2      47
PDGFRA     40
Name: Gene, dtype: int64
```

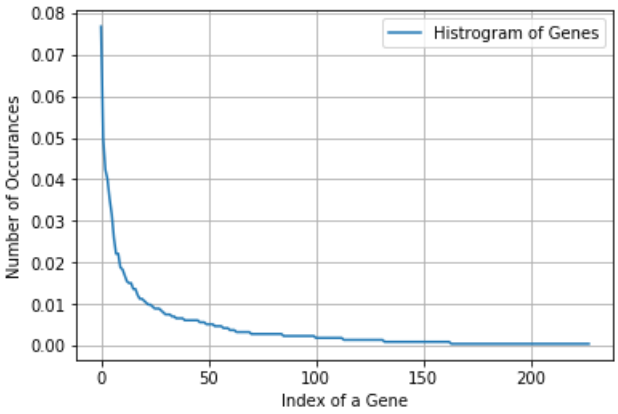
In [49]:

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

```
Ans: There are 228 different categories of genes in the train data, and they are distibuted as fol
lows
```

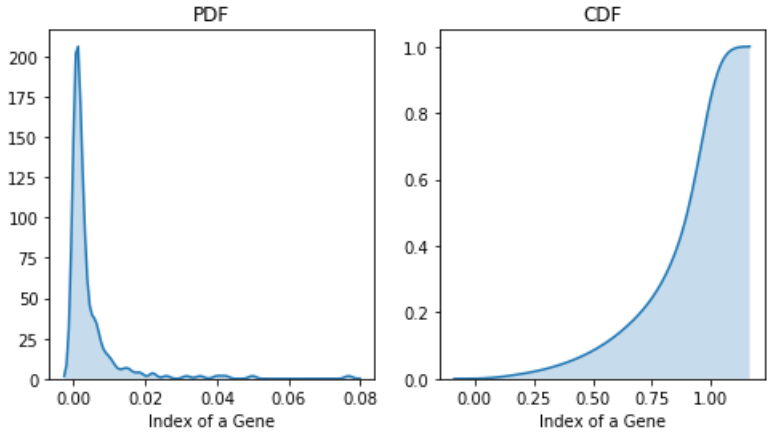
In [83]:

```
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 [86]:

```
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)
sns.kdeplot(h, shade=True, ax=ax1)
plt.xlabel('Index of a Gene')
plt.title("PDF")

ax2 = plt.subplot(122)
sns.kdeplot(c, shade=True, cumulative=True,ax=ax2)
plt.xlabel('Index of a Gene')
plt.title('CDF')

plt.show()
```

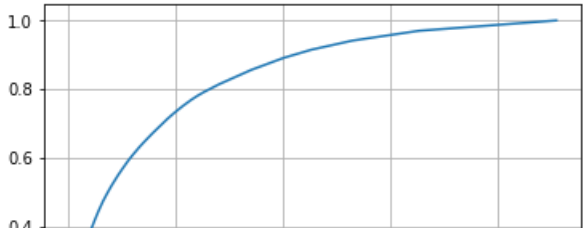


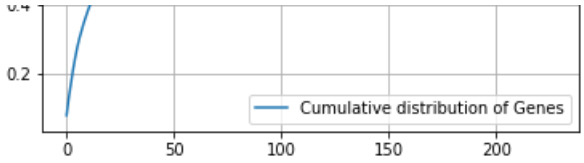

- **It is very skewed.. just like nunmber of occurances of genes.**

  - There are some genes (which are very popular) which are occured many times.

In [85]:

```
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 [60]:

```
#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 [61]:

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

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

In [62]:

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

In [63]:

```
train_df['Gene'].head()
```

Out[63]:

```
2114      GATA3
798       ERBB4
114        MSH6
1832     PPP2R1A
1191      PIK3CA
Name: Gene, dtype: object
```

In [64]:

```
gene_vectorizer.get_feature_names()
```

Out[64]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
```

```
'akt2',
'akt3',
'alk',
'apc',
'ar',
'araf',
'arid1a',
'arid1b',
'arid2',
'arid5b',
'atm',
'atrx',
'aurka',
'axin1',
'axl',
'b2m',
'bap1',
'bard1',
'bcl10',
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgfr1',
'fgfr2',
```

```
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gna11',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'il7r',
'inpp4b',
'jak1',
'jak2',
'kdm5c',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
'myod1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
```

```
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad51d',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xrcc2',
 'yap1']
```

In [65]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 228)
```

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

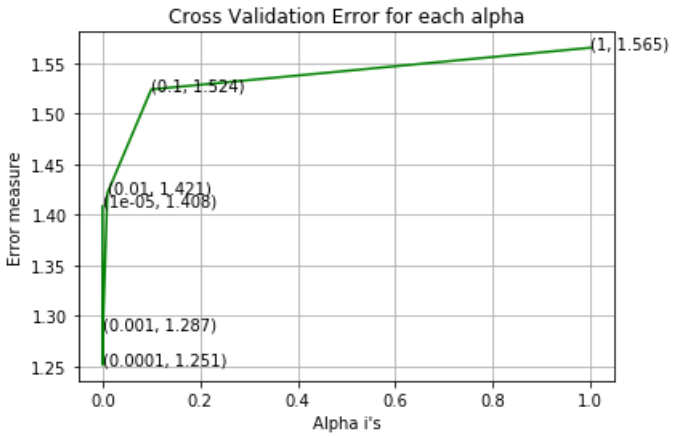
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

```
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_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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_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.clas
ses_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.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_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.408423417065706
For values of alpha =  0.0001 The log loss is: 1.251310142409505
For values of alpha =  0.001 The log loss is: 1.2871723355079334
For values of alpha =  0.01 The log loss is: 1.4212933535857133
```

```
For values of alpha =  0.1 The log loss is: 1.5239953396064498
For values of alpha =  1 The log loss is: 1.5652305898728975
```




```
For values of best alpha =  0.0001 The train log loss is: 1.020391101539154
For values of best alpha =  0.0001 The cross validation log loss is: 1.251310142409505
For values of best alpha =  0.0001 The test log loss is: 1.1660351411522343
```

### **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 [27]:

```
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.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  240  genes in train dataset?
Ans
1. In test data 655 out of 665 : 98.49624060150376
2. In cross validation data 516 out of  532 : 96.99248120300751
```

### 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 [66]:

```
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 : 1931
Truncating_Mutations     60
Deletion                 49
Amplification            40
Fusions                  26
G12V                      4
E17K                      3
Q61H                      3
```

```
A146V                   2
Q22K                    2
ETV6-NTRK3_Fusion       2
Name: Variation, dtype: int64
```

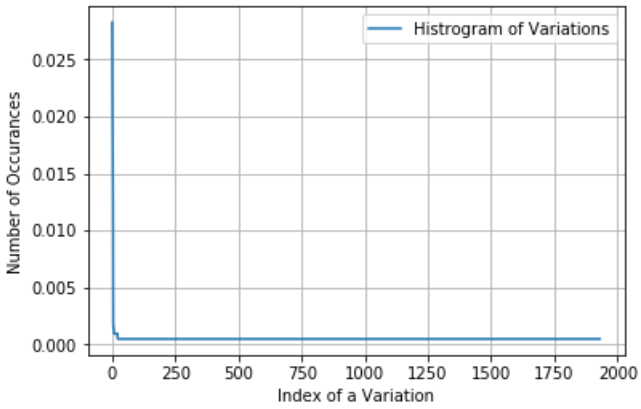
In [67]:

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

```
Ans: There are 1931 different categories of variations in the train data, and they are distibuted
as follows
```

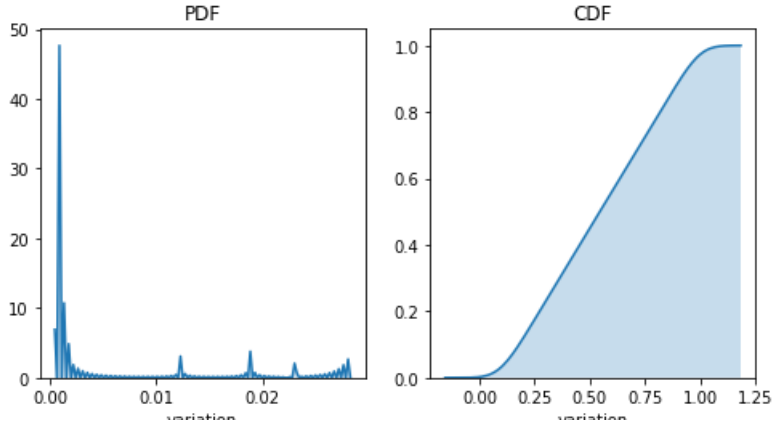
In [87]:

```
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 [90]:

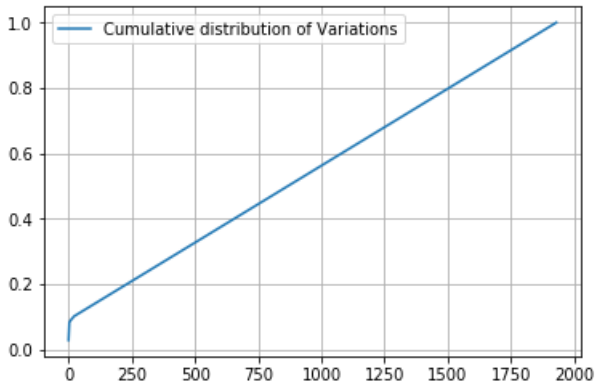
```
fig = plt.figure(figsize=plt.figaspect(.5))

ax1 = plt.subplot(121)
sns.kdeplot(h, shade=True, ax=ax1)
plt.xlabel('variation')
plt.title("PDF")

ax2 = plt.subplot(122)
sns.kdeplot(c, shade=True, cumulative=True,ax=ax2)
plt.xlabel('variation')
plt.title('CDF')

plt.show()
```



In [89]:

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

```
[0.02824859 0.05131827 0.07015066 ... 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 [32]:

```
# 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 [33]:

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

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

In [34]:

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

In [35]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
```

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1947)
```

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

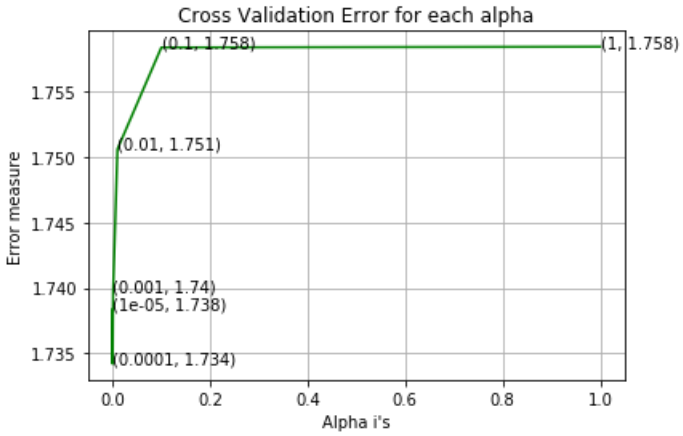
Let's build a model just like the earlier!

In [36]:

```
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_i
ter=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.clas
ses_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7383501540794506
```

```
For values of alpha =  0.0001 The log loss is: 1.7341959500000186
For values of alpha =  0.001 The log loss is: 1.7396875276070594
For values of alpha =  0.01 The log loss is: 1.7506082275667927
For values of alpha =  0.1 The log loss is: 1.758376955130027
For values of alpha =  1 The log loss is: 1.758432423539954
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7501707375360164
For values of best alpha =  0.0001 The cross validation log loss is: 1.7341959500000186
For values of best alpha =  0.0001 The test log loss is: 1.7232013461327338
```

**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 [37]:

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

```
Q12. How many data points are covered by total  1916  genes in test and cross validation data
sets?
Ans
1. In test data 62 out of 665 : 9.323308270676693
2. In cross validation data 50 out of  532 : 9.398496240601503
```

### 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 [38]:

```
# 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
```

```
    return dictionary
```

In [39]:

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

In [40]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) 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 : 1000
```

In [41]:

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

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


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

In [42]:

```
#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 [43]:

```
# https://stackoverflow.com/a/16202486
```

```
# we convert each row values such that they sum to 1
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).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(axis=1)).T
```

In [44]:

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

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

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

In [45]:

```
#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 [46]:

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

```
Counter({254.52592192747414: 1, 183.60105451279452: 1, 137.77140582953885: 1, 130.60668661451996:
1, 130.50353192229554: 1, 118.97773010390101: 1, 118.76397443464872: 1, 115.86126148386417: 1, 113
.16122849564975: 1, 108.86718414848515: 1, 107.11617247436749: 1, 93.1701403387532: 1,
91.71620021151391: 1, 89.81283661067735: 1, 83.84703089517348: 1, 83.15973127873539: 1,
81.5465985454937: 1, 80.20533194121133: 1, 79.476845982100029: 1, 78.41093349723481: 1,
77.26917313833424: 1, 76.78875816556051: 1, 72.01154468476723: 1, 71.913219160705: 1,
70.54491487742779: 1, 69.38206941229406: 1, 68.35059664694721: 1, 66.69488637120013: 1,
65.2400304195645: 1, 64.9744253737983: 1, 64.18976428206068: 1, 64.14425795357121: 1,
63.86491324847996: 1, 59.11415449861189: 1, 58.691298961649636: 1, 57.148806619128834: 1,
56.879401991270036: 1, 56.54575207985287: 1, 55.47810318083202: 1, 51.35643345104813: 1, 51.3193745
1902625: 1, 49.264687829004494: 1, 48.36763638831853: 1, 48.34896852651262: 1, 47.876035480682496:
1, 47.868326936894164: 1, 47.73302349187781: 1, 46.59489286227309: 1, 45.941185734919905: 1,
45.458896761886855: 1, 45.44766725778051: 1, 44.66601523798565: 1, 44.60176088586711: 1, 44.2927090
38307294: 1, 43.563556354766085: 1, 43.257046993769706: 1, 43.145794440115324: 1,
43.03102044465999: 1, 43.00564525733315: 1, 42.91459769952016: 1, 42.6871967243051: 1,
42.10860920970077: 1, 41.755555720090385: 1, 40.838967701087945: 1, 40.77393508427122: 1,
40.77044680644577: 1, 39.985999088462776: 1, 39.937349967267785: 1, 39.741851821568225: 1,
39.046958442721014: 1, 38.7910974690081: 1, 38.76956108799324: 1, 38.696827694343824: 1,
37.9864564525829: 1, 37.589356591313035: 1, 37.208310416274315: 1, 37.03646085989061: 1,
36.848775261984656: 1, 36.794236349755664: 1, 36.19049217968158: 1, 35.94700436042147: 1,
35.707347757945314: 1, 35.419993698686575: 1, 35.35368066765331: 1, 34.924778947129791,
34.531623852727535: 1, 34.51559743598166: 1, 33.95314434476312: 1, 33.93429005243216: 1, 33.6423475
9800014: 1, 33.362687170752565: 1, 33.17218034257704: 1, 32.86171682487145: 1, 32.846559499141364:
1, 32.67924104503026: 1, 32.59151861474852: 1, 32.56271308280809: 1, 32.470237706248035: 1,
32.42283482921731: 1, 32.212019832213926: 1, 32.164380308182494: 1, 32.13197581704733: 1,
32.061934834074975: 1, 31.994505103677124: 1, 31.982431587232767: 1, 31.90141774503617: 1,
31.816449434543824: 1, 31.778509673412724: 1, 31.619880824612505: 1, 31.428620402538485: 1,
31.424209016683093: 1, 31.268921704637723: 1, 31.202733030477711, 31.12591245393626: 1,
30.953839172705816: 1, 30.732635177850728: 1, 30.635775672766353: 1, 30.609738924390257: 1,
30.291288310098018: 1, 30.28090640219907: 1, 30.221202292256226: 1, 30.170220391560346: 1,
29.9141886227778: 1, 29.72438161013519: 1, 29.515192407510053: 1, 29.289917676207615: 1,
29.25468504406302: 1, 29.15649437760411: 1, 28.8868694818248: 1, 28.6069437636271181,
28.519157920122158: 1, 28.427290230455764: 1, 28.30405172502689: 1, 28.15723646508921: 1,
28.088247206792428: 1, 27.75556563298091: 1, 27.696026569761145: 1, 27.653789098457032: 1,
27.54246724549982: 1, 27.2375645726428: 1, 27.133577872557023: 1, 27.050339650191265: 1,
27.035014110514705: 1, 26.865261399133473: 1, 26.86383002586855: 1, 26.78648770103609: 1,
26.68441800150966: 1, 26.29400255232212: 1, 26.28850143999495: 1, 26.25434116500184: 1,
26.10771465957725: 1, 26.0623597346889: 1, 26.016597292007283: 1, 25.934160816190918: 1,
25.811792506347867: 1, 25.78826076852428: 1, 25.768779794289017: 1, 25.608856667688823: 1,
```

25.371402120253343: 1, 25.291678515889515: 1, 25.069312712801796: 1, 25.034373180648025: 1,
24.792788787781998: 1, 24.703874091302257: 1, 24.641607248059877: 1, 24.5988672430533: 1,
24.59211601709419: 1, 24.547555650566864: 1, 24.4764676806213: 1, 24.443610538995483: 1,
24.38620073793549: 1, 24.378347000543975: 1, 24.34932450712955: 1, 24.312519799545672: 1,
24.171448319053788: 1, 24.146661249397123: 1, 24.094796945551618: 1, 23.843075781810555: 1,
23.70861998022288: 1, 23.545678445032742: 1, 23.477450103731712: 1, 23.472779817596066: 1,
23.29627179315087: 1, 23.235271543927553: 1, 23.231649752081562: 1, 23.0859216729634: 1,
23.03130280600551: 1, 23.020059263500478: 1, 22.9119972820845: 1, 22.90030564864208: 1,
22.86438045923456: 1, 22.85752877236397: 1, 22.72242882975215: 1, 22.64611821136129: 1,
22.628555890986625: 1, 22.597644653281723: 1, 22.54579664898495: 1, 22.531584389114467: 1,
22.530386591731048: 1, 22.358030256483442: 1, 22.335355729816406: 1, 22.32336557173367: 1,
22.307887763034405: 1, 22.12775530627845: 1, 22.057976775485894: 1, 22.008923374286052: 1,
22.006224986002064: 1, 21.993708327542272: 1, 21.98327178870907: 1, 21.972877411027476: 1,
21.882237548651577: 1, 21.83695485725249: 1, 21.763047778189893: 1, 21.751987136803077: 1,
21.722031148091943: 1, 21.715181961632886: 1, 21.612092782923657: 1, 21.573593811841985: 1,
21.548494897753216: 1, 21.441743481758753: 1, 21.326184127961373: 1, 21.264271900696126: 1,
21.261893152332192: 1, 21.213711726841602: 1, 21.153862389324374: 1, 21.107141898471742: 1,
21.087708529598242: 1, 21.05377818077324: 1, 21.017565129111844: 1, 21.012261945050106: 1,
20.998477510915045: 1, 20.94355405975718: 1, 20.924654541152638: 1, 20.858248622744362: 1,
20.811219773731697: 1, 20.808694588047512: 1, 20.779041189437176: 1, 20.625978837666153: 1,
20.616428537005266: 1, 20.599009959089333: 1, 20.59551006060526: 1, 20.584652947614174: 1,
20.46352573591614: 1, 20.45622836956135: 1, 20.45591929284537: 1, 20.437286781294553: 1,
20.376988203601847: 1, 20.32690368380444: 1, 20.26249528792523: 1, 20.248085832399596: 1,
20.247417610198983: 1, 20.143568553190043: 1, 20.08725265391888: 1, 20.048866341814353: 1,
19.966445010365305: 1, 19.944870433609115: 1, 19.87778281389495: 1, 19.83284436308858: 1,
19.72546692168977: 1, 19.722412483646668: 1, 19.716721388583704: 1, 19.545396767531475: 1,
19.5421042404685: 1, 19.46017474117823: 1, 19.42161976557964: 1, 19.37844966820774: 1,
19.24272357126719: 1, 19.231931466926685: 1, 19.199298951848366: 1, 19.160174013066438: 1,
19.14937422386643: 1, 19.0890495223536: 1, 19.079439203458588: 1, 19.063145817831014: 1,
18.946752368153444: 1, 18.923396752064637: 1, 18.910879617432755: 1, 18.9034924435966: 1,
18.885420159829998: 1, 18.884370846745057: 1, 18.88081605044032: 1, 18.82157925201811: 1,
18.811716709117487: 1, 18.80140376592355: 1, 18.778979871011632: 1, 18.7762395540322: 1,
18.769235777204916: 1, 18.768553209752184: 1, 18.75328315988588: 1, 18.743165574109 84: 1,
18.711440537895278: 1, 18.655088316188053: 1, 18.649255929180683: 1, 18.632872048056754: 1,
18.585131875580167: 1, 18.578799054273016: 1, 18.578499421958504: 1, 18.56474532065759: 1,
18.509623034987495: 1, 18.43127407564247: 1, 18.285955569243022: 1, 18.25398241331258: 1,
18.242507638625973: 1, 18.22534996604229: 1, 18.205970519037315: 1, 18.195777124208867: 1,
18.180164722019207: 1, 18.174352231174502: 1, 18.123654522960596: 1, 18.114618748011 35: 1,
18.113623833075 57: 1, 18.089060625595692: 1, 18.077954991602347: 1, 18.020296231410335: 1,
18.018874653603817: 1, 18.0144920117635: 1, 17.970670265369673: 1, 17.915239988146617: 1,
17.90730949069613: 1, 17.861996416832778: 1, 17.734559567174227: 1, 17.71554780349916: 1,
17.67415276196686: 1, 17.650067626508548: 1, 17.491466259435125: 1, 17.49022101919867: 1,
17.472672920934173: 1, 17.448031794104665: 1, 17.42444000604434: 1, 17.417398542907808: 1,
17.416580358089536: 1, 17.377984320989608: 1, 17.367540336790345: 1, 17.328505217169926: 1,
17.299083059828796: 1, 17.28541462785061: 1, 17.27453961116781: 1, 17.25245429920402: 1, 17.2460598
0508473: 1, 17.20193803295196: 1, 17.171890953178384: 1, 17.168057019518727: 1,
17.141092962103873: 1, 17.08439721152977: 1, 17.053172967914882: 1, 17.04842835140129: 1,
17.044347209248077: 1, 17.040950585654436: 1, 17.02374751289883: 1, 16.93776531328489: 1,
16.8963815180542: 1, 16.892951936038678: 1, 16.876135754906997: 1, 16.848916350456086: 1,
16.780315644042105: 1, 16.776203111142543: 1, 16.747382452114614: 1, 16.71720880904232: 1,
16.690264803124162: 1, 16.640514090190166: 1, 16.639497849378508: 1, 16.622346931600134: 1,
16.62003601586526: 1, 16.607543670266253: 1, 16.60540858317221: 1, 16.585824056146397: 1,
16.572720911363536: 1, 16.57116639745167: 1, 16.570968558377754: 1, 16.51242678928594: 1,
16.462382632048683: 1, 16.45111478943778: 1, 16.45101493607511: 1, 16.37817669576286: 1, 16.3534392
2074471: 1, 16.327054541191373: 1, 16.29442745042774: 1, 16.26841262352852: 1, 16.232420682784838:
1, 16.223441206134595: 1, 16.18445790721299: 1, 16.12479466567028: 1, 16.09748980582311: 1,
16.039227825707368: 1, 15.996736655531825: 1, 15.995003720938044: 1, 15.993435800169152: 1,
15.974687994109441: 1, 15.96627198056753: 1, 15.951388371788997: 1, 15.940833259455136: 1,
15.939897788885983: 1, 15.936945120802333: 1, 15.907206463659955: 1, 15.886290434026783: 1,
15.806279176984074: 1, 15.766163121126427: 1, 15.755458332842025: 1, 15.694958086494417: 1,
15.637854668566277: 1, 15.568188159677174: 1, 15.484793757732202: 1, 15.435330822065183: 1,
15.400613053259226: 1, 15.34184115383534: 1, 15.319569996466395: 1, 15.298567602033277: 1,
15.27966172575856: 1, 15.258490263238: 1, 15.244298443530237: 1, 15.221029375206369: 1,
15.198522329269855: 1, 15.19727430915682: 1, 15.158060480463798: 1, 15.138026598176113: 1,
15.116825637147832: 1, 15.102194135226158: 1, 15.09286199136193: 1, 15.085826283460296: 1,
15.04960269932347: 1, 15.049107375582393: 1, 15.047268524464688: 1, 15.04039710592238: 1,
14.985779875796041: 1, 14.980754042665541: 1, 14.964197247289341: 1, 14.852732174861885: 1,
14.846192655084838: 1, 14.834033053731545: 1, 14.792343210099627: 1, 14.791371748214578: 1,
14.778658870601573: 1, 14.774161463060988: 1, 14.746916226528 73: 1, 14.734617477579151: 1,
14.717821517372004: 1, 14.697374384260272: 1, 14.69737139095894: 1, 14.682385321753 46: 1,
14.668225337702825: 1, 14.653945841814625: 1, 14.63369181655688: 1, 14.572323350147522: 1,
14.566853330318246: 1, 14.550736282484603: 1, 14.530885076647937: 1, 14.509522714018754: 1,
14.504501767189 51: 1, 14.490147775015473: 1, 14.47514124767722: 1, 14.474589376419498: 1,
14.461903021839463: 1, 14.432562029427006: 1, 14.410906617262324: 1, 14.405518846585995: 1,
14.386862743295 45: 1, 14.384763072690935: 1, 14.367201862024535: 1, 14.313643361074238: 1,
14.256836056069195: 1, 14.18766641972555: 1, 14.182992088516645: 1, 14.175017700351297: 1,
14.129274344450103: 1, 14.123828400915194: 1, 14.113600052882422: 1, 14.112735017681842: 1,

14.081708184380732: 1, 14.080212741679457: 1, 13.998552301947104: 1, 13.986742225814583: 1,
13.939233239881204: 1, 13.897949611377587: 1, 13.82096265832168: 1, 13.817036329804003: 1,
13.810874554285599: 1, 13.802804944124961: 1, 13.77762806026862: 1, 13.757943909237467: 1,
13.738516601365758: 1, 13.728131467277361: 1, 13.70266276376155: 1, 13.682226498705601: 1,
13.66848227440636: 1, 13.666672128291097: 1, 13.658490323849374: 1, 13.642197988517605: 1,
13.635690009777777: 1, 13.59004821612936: 1, 13.583322374397019: 1, 13.577076356163447: 1,
13.55936214299302: 1, 13.552198371973388: 1, 13.532458756834087: 1, 13.51230752455928: 1,
13.47857370812565: 1, 13.474018585202904: 1, 13.404855874741777: 1, 13.404049126811682: 1,
13.378195192676465: 1, 13.331022937558329: 1, 13.329466563647518: 1, 13.327457370765519: 1,
13.304577026156318: 1, 13.288123724825137: 1, 13.285972540949185: 1, 13.224956817314881: 1,
13.224792135822219: 1, 13.223660905401065: 1, 13.222025830763467: 1, 13.186950521844185: 1,
13.161620348080497: 1, 13.135906611699907: 1, 13.129667847693442: 1, 13.097878748916319: 1,
13.042931939639109: 1, 13.023425512015006: 1, 13.019932689608115: 1, 13.000605189890553: 1,
12.96936402230458: 1, 12.962647272488836: 1, 12.922828399565315: 1, 12.9164606618024358: 1,
12.898045928909797: 1, 12.89374690150829: 1, 12.845604327149433: 1, 12.837219343219O8: 1,
12.829232466364713: 1, 12.82139504224592: 1, 12.815258972533037: 1, 12.806601179301046: 1,
12.784316643534229: 1, 12.766185232910082: 1, 12.752616208093674: 1, 12.701290332388483: 1,
12.685634776234448: 1, 12.651099330037125: 1, 12.650619935710761: 1, 12.616006899819949: 1,
12.580423389326985: 1, 12.562526143153713: 1, 12.546949341275981: 1, 12.520258800033096: 1,
12.50951236631139: 1, 12.496966552134149: 1, 12.491731888640555: 1, 12.458500176127922: 1,
12.433840693658338: 1, 12.432679216587404: 1, 12.428896240016192: 1, 12.423096260519285: 1,
12.379145481999423: 1, 12.364444051482437: 1, 12.363688787276615: 1, 12.30171956568379: 1,
12.300320641557867: 1, 12.23492363568366: 1, 12.232530060040876: 1, 12.226509421548101: 1,
12.225602867736331: 1, 12.175298117579485: 1, 12.168659323839258: 1, 12.167907810132354: 1,
12.166001840499995: 1, 12.14589324265871: 1, 12.123184305782443: 1, 12.119519084339133: 1,
12.103101949592286: 1, 12.093464226065496: 1, 12.073575128291074: 1, 12.056690061050611: 1,
12.04356905180932: 1, 12.03850477435337: 1, 12.016752669863992: 1, 12.004696529641587: 1,
11.95840000476332: 1, 11.91366043681087: 1, 11.894720864663741: 1, 11.89450306064825: 1,
11.885029703426076: 1, 11.873992206797942: 1, 11.854241884101118: 1, 11.852134182362104: 1,
11.82979454176772: 1, 11.804895194374488: 1, 11.79217631815355: 1, 11.7881993862877491: 1,
11.781913207624005: 1, 11.778062925055174: 1, 11.767459265794422: 1, 11.76483048172266: 1,
11.758001738814652: 1, 11.757046887444183: 1, 11.72654126748584: 1, 11.7140866615367861: 1,
11.673988633681804: 1, 11.641053785718388: 1, 11.63946934070686: 1, 11.611885221998287: 1,
11.583940401878158: 1, 11.57848015400457: 1, 11.548772145241994: 1, 11.547856121148065: 1,
11.521093192577071: 1, 11.517547467914172: 1, 11.458187884154322: 1, 11.45296630942062: 1,
11.443450725039986: 1, 11.443297759663926: 1, 11.438616482898803: 1, 11.414749879819921: 1,
11.371780131546782: 1, 11.359090116893743: 1, 11.3527376939223: 1, 11.346943134753069: 1,
11.341071892695581: 1, 11.327227296976824: 1, 11.316764474173025: 1, 11.30743709081445: 1,
11.257798159972637: 1, 11.242113741715656: 1, 11.217762926742: 1, 11.2081638742776661: 1,
11.205675046857312: 1, 11.194020789737673: 1, 11.183510744557001: 1, 11.178471426430434: 1,
11.176325591347517: 1, 11.149588247594615: 1, 11.14318931089953: 1, 11.135175345217657: 1,
11.127755011524517: 1, 11.119393973108988: 1, 11.097125833618579: 1, 11.090947665187768: 1,
11.08891582317535: 1, 11.087721006164005: 1, 11.07644734296257: 1, 11.0632634999180421: 1,
11.061933442465344: 1, 11.039644838658635: 1, 11.036045151660643: 1, 11.032575903315676: 1,
11.022703165178601: 1, 11.003253535485166: 1, 10.972369609437623: 1, 10.961669204680035: 1,
10.952699460194891: 1, 10.921283611647718: 1, 10.92016133154617: 1, 10.9056275382586721: 1,
10.900485012875045: 1, 10.89890518527273: 1, 10.895603802363686: 1, 10.8934686988793581: 1,
10.88336889374278: 1, 10.883142160637117: 1, 10.87772577474781: 1, 10.8691576461467051: 1,
10.868358023166959: 1, 10.863024194862838: 1, 10.855309053144875: 1, 10.8419875100703861: 1,
10.814615397789026: 1, 10.806153534469415: 1, 10.790201373006646: 1, 10.7894618755799141: 1,
10.783082507730402: 1, 10.736007828501734: 1, 10.70283544728341: 1, 10.7027055260947571: 1,
10.699474248722286: 1, 10.676967466356693: 1, 10.673347657316366: 1, 10.6669518967040831: 1,
10.649795906261273: 1, 10.649265512347435: 1, 10.646339453489571: 1, 10.6415542827223271: 1,
10.637820128049793: 1, 10.637577772634128: 1, 10.636480955432612: 1, 10.6192577118939991: 1,
10.567562799036056: 1, 10.560383369324239: 1, 10.557069846900667: 1, 10.5543124629129551: 1,
10.540409653446309: 1, 10.535837208305558: 1, 10.522896113265253: 1, 10.5211129111945431: 1,
10.52107897985159: 1, 10.496635192718584: 1, 10.483527482593924: 1, 10.467085943878041: 1,
10.459846648165623: 1, 10.45497552354573: 1, 10.439220472157208: 1, 10.4335919493726711: 1,
10.428388688737892: 1, 10.409363846776923: 1, 10.402927617561215: 1, 10.3976286771503231: 1,
10.338746622583725: 1, 10.284475523350686: 1, 10.278757340812525: 1, 10.2593252657587261: 1,
10.258947016682978: 1, 10.241943680159086: 1, 10.231908685957288: 1, 10.2315019732863451: 1,
10.215964840410608: 1, 10.21314879393688: 1, 10.210087193153553: 1, 10.192502731879571: 1,
10.186249227053912: 1, 10.178735403257443: 1, 10.163952420294017: 1, 10.1594961738233211: 1,
10.157237011830723: 1, 10.155814673959155: 1, 10.153810703069704: 1, 10.1271145284348291: 1,
10.104473683735682: 1, 10.024142487627891: 1, 10.019326289005866: 1, 10.0122936548695861: 1,
10.00249829576651: 1, 9.980743269175655: 1, 9.979053655859246: 1, 9.9755227700890251: 1,
9.97375764449367: 1, 9.971211007301816: 1, 9.923926741743704: 1, 9.896124950019665: 1,
9.8905796150531: 1, 9.868044221607489: 1, 9.863483686652653: 1, 9.854275720731492: 1, 9.84264670569
0726: 1, 9.83369509489395: 1, 9.830908641767138: 1, 9.794667513235302: 1, 9.788654392804528: 1,
9.785353540763673: 1, 9.783290089667288: 1, 9.782354246083138: 1, 9.740135005606994: 1,
9.732081807200858: 1, 9.715528553216231: 1, 9.715507729317521: 1, 9.708972291709981: 1,
9.692426399174314: 1, 9.690246879223157: 1, 9.684913068771806: 1, 9.6780263327716871: 1,
9.667525013564878: 1, 9.659289439288981: 1, 9.64443229111481: 1, 9.632536177752431: 1,
9.629745293559235: 1, 9.621584899910914: 1, 9.621369205953478: 1, 9.592683307749941: 1,
9.587603107768903: 1, 9.57786539982038: 1, 9.532377144076685: 1, 9.528212984024657: 1,
9.517225519135614: 1, 9.498198944133264: 1, 9.49682354801519: 1, 9.493720125070721: 1,
9.492049723581763: 1, 9.473674391623502: 1, 9.464785770649133: 1, 9.4520918603630351:

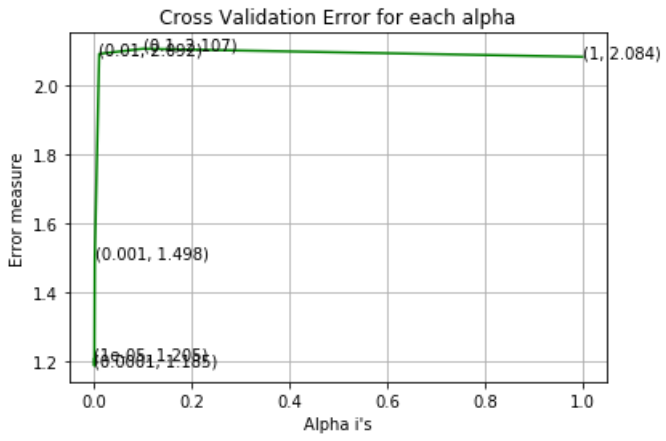
```
9.444942329381714: 1, 9.416095595667091: 1, 9.4061261995113323: 1, 9.4040887150035881: 1,
9.388433591741911: 1, 9.369221580302552: 1, 9.366275098583035: 1, 9.357955948716743: 1,
9.357150588268698: 1, 9.348966023257892: 1, 9.346057467787611: 1, 9.341061306018425: 1,
9.33527387340837: 1, 9.328708634415136: 1, 9.328212518684287: 1, 9.324429439249128: 1,
9.321066912392086: 1, 9.318196147160446: 1, 9.316870224423111: 1, 9.296526747597433: 1,
9.285687410737717: 1, 9.284867212168168: 1, 9.252872545932922: 1, 9.247536284158027: 1,
9.229374238024027: 1, 9.228447734555703: 1, 9.225072359338542: 1, 9.219579674161297: 1,
9.215808735336601: 1, 9.214241662902939: 1, 9.205260496219427: 1, 9.195794252684198: 1,
9.194993691590614: 1, 9.188054023849448: 1, 9.184280609558877: 1, 9.180204872473416: 1,
9.159920990138938: 1, 9.156651477862678: 1, 9.150446721095095: 1, 9.146250543004944: 1,
9.140299732561434: 1, 9.139805723950234: 1, 9.110801269030594: 1, 9.085794694196935: 1,
9.07901356542031: 1, 9.070248755836973: 1, 9.06243813928735: 1, 9.04528109371469: 1,
9.037949416574351: 1, 9.028784361667457: 1, 8.99456794832822: 1, 8.988511657878698: 1,
8.986332517287245: 1, 8.978610533076512: 1, 8.970533320580568: 1, 8.9696817121207019: 1,
8.953027006274407: 1, 8.913772025459286: 1, 8.895140538229045: 1, 8.889781987409256: 1,
8.889323052055516: 1, 8.83307278440843: 1, 8.830225213837066: 1, 8.81613030781641: 1,
8.79924521453622: 1, 8.793551244449962: 1, 8.739524936742923: 1, 8.718952429088723: 1,
8.704275560314983: 1, 8.689954253111743: 1, 8.688260114763851: 1, 8.679853262709464: 1,
8.667883493965038: 1, 8.648044334457248: 1, 8.64757416517268: 1, 8.642669841924688: 1,
8.639576659245336: 1, 8.634954634150008: 1, 8.595067939902314: 1, 8.594410737833135: 1,
8.583735850364617: 1, 8.577514689927812: 1, 8.555835003543235: 1, 8.554291932424531: 1,
8.549258478587754: 1, 8.544079927676584: 1, 8.543308199850088: 1, 8.543256412306517: 1,
8.532383147022369: 1, 8.52761599590634: 1, 8.524232769021573: 1, 8.523863430419503: 1,
8.513157966209542: 1, 8.500515976582513: 1, 8.493730695818169: 1, 8.488666407018577: 1,
8.478792771082936: 1, 8.468659512881432: 1, 8.467772212691298: 1, 8.463514162030195: 1,
8.457675131941533: 1, 8.449627354618489: 1, 8.44928463355024: 1, 8.422555524678117: 1,
8.406854299921779: 1, 8.403182181547407: 1, 8.39978678607536: 1, 8.398246189709692: 1,
8.386486284637867: 1, 8.364422994243338: 1, 8.362973216188811: 1, 8.353822183179338: 1,
8.347458859733507: 1, 8.335923724546868: 1, 8.334246013927995: 1, 8.332464236290877: 1,
8.329161675321675: 1, 8.323917119694164: 1, 8.295537396179576: 1, 8.279078433454561: 1,
8.276182187920936: 1, 8.215387875562474: 1, 8.207396360700093: 1, 8.205444086388665: 1,
8.201668616048973: 1, 8.201547075731108: 1, 8.199971505971881: 1, 8.189065080342067: 1,
8.188151855757079: 1, 8.177063783752434: 1, 8.1758610643886: 1, 8.171241625508571: 1,
8.165799991373198: 1, 8.162622582526334: 1, 8.13567232568708: 1, 8.129096949767595: 1,
8.128068047051745: 1, 8.112777225388147: 1, 8.110156903253444: 1, 8.081666307834492: 1,
8.051487735435995: 1, 8.033864157906502: 1, 8.019160230471375: 1, 8.004772657881702: 1,
8.004713076561533: 1, 7.986166307238766: 1, 7.983334454661324: 1, 7.972612924595187: 1,
7.9562012105779045: 1, 7.949202375055044: 1, 7.943104458483849: 1, 7.926891841603556: 1, 7.92354354
72999685: 1, 7.911380113369022: 1, 7.905418740639337: 1, 7.886245019888085: 1, 7.883547937874482:
1, 7.873720725517586: 1, 7.864525550229159: 1, 7.832074575752864: 1, 7.823537737948577: 1,
7.8160538445491845: 1, 7.7945493398678: 1, 7.791833195516489: 1, 7.781569573973018: 1,
7.757220764467218: 1, 7.749371934969986: 1, 7.729338665787515: 1, 7.720088518326062: 1,
7.6768284536909155: 1, 7.668347019160342: 1, 7.66482900050324: 1, 7.634350009071232: 1,
7.625106288442586: 1, 7.6153784037702685: 1, 7.58852032467025: 1, 7.581701376284079: 1,
7.57458625126042: 1, 7.566085647664379: 1, 7.565756464310964: 1, 7.559813719857333: 1,
7.540121865887277: 1, 7.527140371115726: 1, 7.502994561252464: 1, 7.4993417112125025: 1,
7.493875158264159: 1, 7.447615954053527: 1, 7.433385255470019: 1, 7.43294641018732: 1,
7.431970728041107: 1, 7.428097411784907: 1, 7.410003582506409: 1, 7.398135580018345: 1,
7.382150244696346: 1, 7.369221120994188: 1, 7.36722486472831: 1, 7.347378455839896: 1,
7.33579657642076: 1, 7.268473307318397: 1, 7.226306494831496: 1, 7.219738155001889: 1,
7.1992216596298455: 1, 7.180111288754653: 1, 7.150885453585379: 1, 7.138539390968147: 1, 7.07859971
51039: 1, 7.025930321537426: 1, 7.014784807649769: 1, 7.005180853017911: 1, 6.9851968259127695: 1,
6.983703582846159: 1, 6.976335634964589: 1, 6.963069371379077: 1, 6.899926179994526: 1,
6.860284372707164: 1, 6.826766968573364: 1, 6.823335689062202: 1, 6.806508110978466: 1,
6.75500428503125: 1, 6.708478193017719: 1, 6.566414734332034: 1, 6.52616226731837: 1,
6.5200358850868065: 1, 6.4933548399036845: 1})
```

In [47]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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_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.clas
ses_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2052896959960406
For values of alpha =  0.0001 The log loss is: 1.1853420006719324
For values of alpha =  0.001 The log loss is: 1.4978288707765155
For values of alpha =  0.01 The log loss is: 2.092265441623324
For values of alpha =  0.1 The log loss is: 2.1070611605774943
For values of alpha =  1 The log loss is: 2.0835241782053497
```




```
For values of best alpha =  0.0001 The train log loss is: 0.8181995262257217
For values of best alpha =  0.0001 The cross validation log loss is: 1.1853420006719324
For values of best alpha =  0.0001 The test log loss is: 1.0901851807271508
```

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

**Ans.** Yes, it seems like!

In [52]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [53]:

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

```
95.3 % of word of test data appeared in train data
94.1 % of word of Cross Validation appeared in train data
```

# 4. Machine Learning Models

In [65]:

```
#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 [66]:

```
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 [67]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

    fea1_len = len(gene_vec.get_feature_names())
```

```
    fea2_len = len(var_count_vec.get_feature_names())

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

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

## Stacking the three types of features

In [68]:

```
# merging gene, variance and text features

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

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

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

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

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


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

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

In [69]:

In [69]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3186)
(number of data points * number of features) in test data =  (665, 3186)
(number of data points * number of features) in cross validation data = (532, 3186)
```

In [70]:

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

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

# 4.1. Base Line Model

## 4.1.1. Naive Bayes

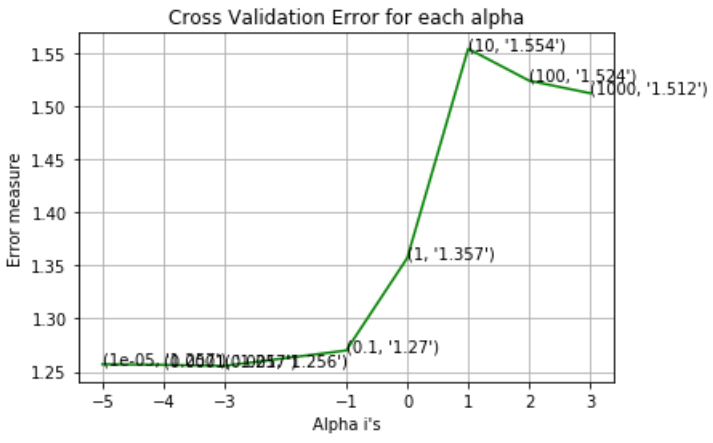
#### 4.1.1.1. Hyper parameter tuning

In [71]:

```
# 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-
algorithm-1/
# -----------------------


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
```

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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.2570566021429748
for alpha = 0.0001
Log Loss : 1.2565049892268831
for alpha = 0.001
Log Loss : 1.2555569799336603
for alpha = 0.1
Log Loss : 1.2700400410941022
for alpha = 1
Log Loss : 1.3570595738279727
for alpha = 10
Log Loss : 1.5541915395660462
for alpha = 100
Log Loss : 1.524107006747969
for alpha = 1000
Log Loss : 1.5123093125686857
```




```
For values of best alpha =  0.001 The train log loss is: 0.5057185779258165
For values of best alpha =  0.001 The cross validation log loss is: 1.2555569799336603
```

For values of best alpha =  0.001 The cross validation log loss is: 1.2599509799336603
For values of best alpha =  0.001 The test log loss is: 1.1535396155821565

#### 4.1.1.2. Testing the model with best hyper paramters

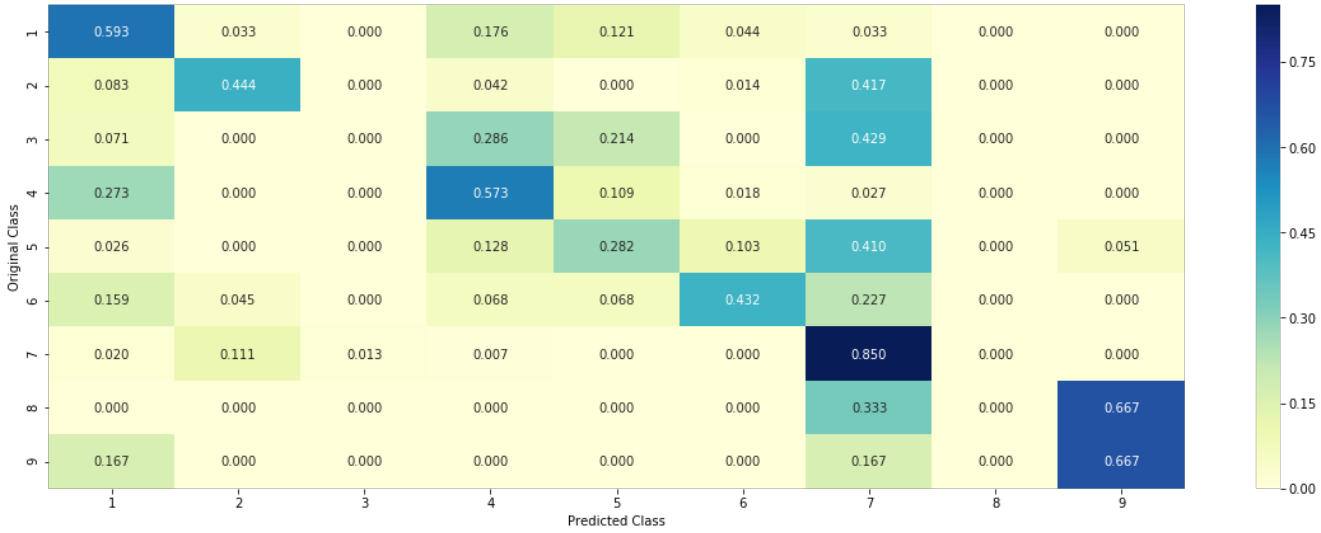
In [72]:

```
# 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-
algorithm-1/
# -----------------------


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

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
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 : 1.2555569799336603
Number of missclassified point : 0.4116541353383459
-------------------- Confusion matrix --------------------
```


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


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




#### 4.1.1.3. Feature Importance, Correctly classified point

In [73]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0557 0.0396 0.011  0.0601 0.0323 0.7183 0.0775 0.0029 0.0027]]
Actual Class : 6
--------------------------------------------------
5 Text feature [odds] present in test data point [True]
6 Text feature [brca] present in test data point [True]
7 Text feature [deleterious] present in test data point [True]
8 Text feature [57] present in test data point [True]
9 Text feature [classified] present in test data point [True]
10 Text feature [basis] present in test data point [True]
11 Text feature [history] present in test data point [True]
```

```
12 Text feature [combined] present in test data point [True]
14 Text feature [brca1] present in test data point [True]
15 Text feature [43] present in test data point [True]
16 Text feature [family] present in test data point [True]
18 Text feature [evidence] present in test data point [True]
19 Text feature [predicted] present in test data point [True]
20 Text feature [26] present in test data point [True]
22 Text feature [brca2] present in test data point [True]
23 Text feature [sequence] present in test data point [True]
24 Text feature [models] present in test data point [True]
25 Text feature [model] present in test data point [True]
27 Text feature [expected] present in test data point [True]
29 Text feature [000] present in test data point [True]
30 Text feature [23] present in test data point [True]
31 Text feature [substitutions] present in test data point [True]
34 Text feature [testing] present in test data point [True]
36 Text feature [41] present in test data point [True]
37 Text feature [42] present in test data point [True]
38 Text feature [known] present in test data point [True]
39 Text feature [ring] present in test data point [True]
40 Text feature [35] present in test data point [True]
41 Text feature [use] present in test data point [True]
42 Text feature [likely] present in test data point [True]
44 Text feature [given] present in test data point [True]
45 Text feature [variant] present in test data point [True]
46 Text feature [used] present in test data point [True]
47 Text feature [variants] present in test data point [True]
49 Text feature [data] present in test data point [True]
52 Text feature [70] present in test data point [True]
53 Text feature [significance] present in test data point [True]
54 Text feature [significant] present in test data point [True]
56 Text feature [50] present in test data point [True]
57 Text feature [majority] present in test data point [True]
58 Text feature [would] present in test data point [True]
60 Text feature [neutral] present in test data point [True]
61 Text feature [conserved] present in test data point [True]
62 Text feature [54] present in test data point [True]
63 Text feature [75] present in test data point [True]
65 Text feature [studied] present in test data point [True]
66 Text feature [individuals] present in test data point [True]
67 Text feature [although] present in test data point [True]
69 Text feature [least] present in test data point [True]
70 Text feature [100] present in test data point [True]
71 Text feature [overall] present in test data point [True]
72 Text feature [missense] present in test data point [True]
73 Text feature [56] present in test data point [True]
74 Text feature [ovarian] present in test data point [True]
75 Text feature [25] present in test data point [True]
78 Text feature [analysis] present in test data point [True]
81 Text feature [prior] present in test data point [True]
83 Text feature [population] present in test data point [True]
84 Text feature [29] present in test data point [True]
85 Text feature [approach] present in test data point [True]
86 Text feature [thus] present in test data point [True]
87 Text feature [34] present in test data point [True]
88 Text feature [12] present in test data point [True]
90 Text feature [28] present in test data point [True]
91 Text feature [methods] present in test data point [True]
92 Text feature [developed] present in test data point [True]
94 Text feature [number] present in test data point [True]
95 Text feature [available] present in test data point [True]
96 Text feature [three] present in test data point [True]
97 Text feature [based] present in test data point [True]
98 Text feature [30] present in test data point [True]
99 Text feature [information] present in test data point [True]
Out of the top  100  features  72 are present in query point
```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

In [82]:

```
test_point_index = 50
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1351 0.0687 0.0295 0.1715 0.149  0.3024 0.1342 0.005  0.0047]]
Actual Class : 3
--------------------------------------------------
5 Text feature [odds] present in test data point [True]
7 Text feature [deleterious] present in test data point [True]
9 Text feature [classified] present in test data point [True]
10 Text feature [basis] present in test data point [True]
11 Text feature [history] present in test data point [True]
12 Text feature [combined] present in test data point [True]
14 Text feature [brca1] present in test data point [True]
16 Text feature [family] present in test data point [True]
19 Text feature [predicted] present in test data point [True]
20 Text feature [26] present in test data point [True]
21 Text feature [31] present in test data point [True]
22 Text feature [brca2] present in test data point [True]
23 Text feature [sequence] present in test data point [True]
25 Text feature [model] present in test data point [True]
27 Text feature [expected] present in test data point [True]
28 Text feature [ligase] present in test data point [True]
29 Text feature [000] present in test data point [True]
30 Text feature [23] present in test data point [True]
31 Text feature [substitutions] present in test data point [True]
33 Text feature [47] present in test data point [True]
34 Text feature [testing] present in test data point [True]
35 Text feature [e2] present in test data point [True]
38 Text feature [known] present in test data point [True]
39 Text feature [ring] present in test data point [True]
40 Text feature [35] present in test data point [True]
41 Text feature [use] present in test data point [True]
42 Text feature [likely] present in test data point [True]
44 Text feature [given] present in test data point [True]
45 Text feature [variant] present in test data point [True]
46 Text feature [used] present in test data point [True]
47 Text feature [variants] present in test data point [True]
48 Text feature [ubiquitin] present in test data point [True]
49 Text feature [data] present in test data point [True]
51 Text feature [substitution] present in test data point [True]
52 Text feature [70] present in test data point [True]
53 Text feature [significance] present in test data point [True]
54 Text feature [significant] present in test data point [True]
56 Text feature [50] present in test data point [True]
57 Text feature [majority] present in test data point [True]
58 Text feature [would] present in test data point [True]
59 Text feature [interaction] present in test data point [True]
60 Text feature [neutral] present in test data point [True]
61 Text feature [conserved] present in test data point [True]
63 Text feature [75] present in test data point [True]
64 Text feature [32] present in test data point [True]
66 Text feature [individuals] present in test data point [True]
67 Text feature [although] present in test data point [True]
68 Text feature [60] present in test data point [True]
69 Text feature [least] present in test data point [True]
70 Text feature [100] present in test data point [True]
71 Text feature [overall] present in test data point [True]
72 Text feature [missense] present in test data point [True]
74 Text feature [ovarian] present in test data point [True]
75 Text feature [25] present in test data point [True]
78 Text feature [analysis] present in test data point [True]
79 Text feature [proportion] present in test data point [True]
80 Text feature [set] present in test data point [True]
82 Text feature [database] present in test data point [True]
83 Text feature [population] present in test data point [True]
84 Text feature [29] present in test data point [True]
85 Text feature [approach] present in test data point [True]
86 Text feature [thus] present in test data point [True]
88 Text feature [12] present in test data point [True]
```

```
89 Text feature [24] present in test data point [True]
90 Text feature [28] present in test data point [True]
91 Text feature [methods] present in test data point [True]
92 Text feature [developed] present in test data point [True]
93 Text feature [values] present in test data point [True]
94 Text feature [number] present in test data point [True]
95 Text feature [available] present in test data point [True]
96 Text feature [three] present in test data point [True]
97 Text feature [based] present in test data point [True]
99 Text feature [information] present in test data point [True]
Out of the top  100  features  73 are present in query point
```

## 4.2. K Nearest Neighbour Classification

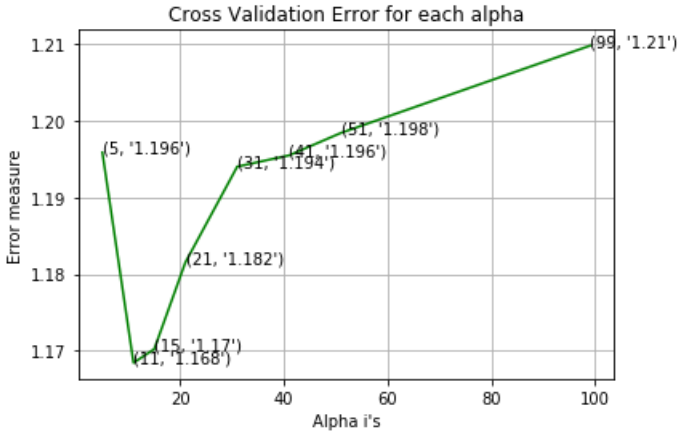
### 4.2.1. Hyper parameter tuning

In [83]:

```
# 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-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------


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


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

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.1958913504323438
for alpha = 11
Log Loss : 1.1683906197459837
for alpha = 15
Log Loss : 1.1701931221700888
for alpha = 21
Log Loss : 1.1815158643105412
for alpha = 31
Log Loss : 1.1940427919370884
for alpha = 41
Log Loss : 1.1955105634823004
for alpha = 51
Log Loss : 1.1984442025584388
for alpha = 99
Log Loss : 1.2098402204820415
```




```
For values of best alpha =  11 The train log loss is: 0.6354295691196185
For values of best alpha =  11 The cross validation log loss is: 1.1683906197459837
For values of best alpha =  11 The test log loss is: 1.0227261687301359
```

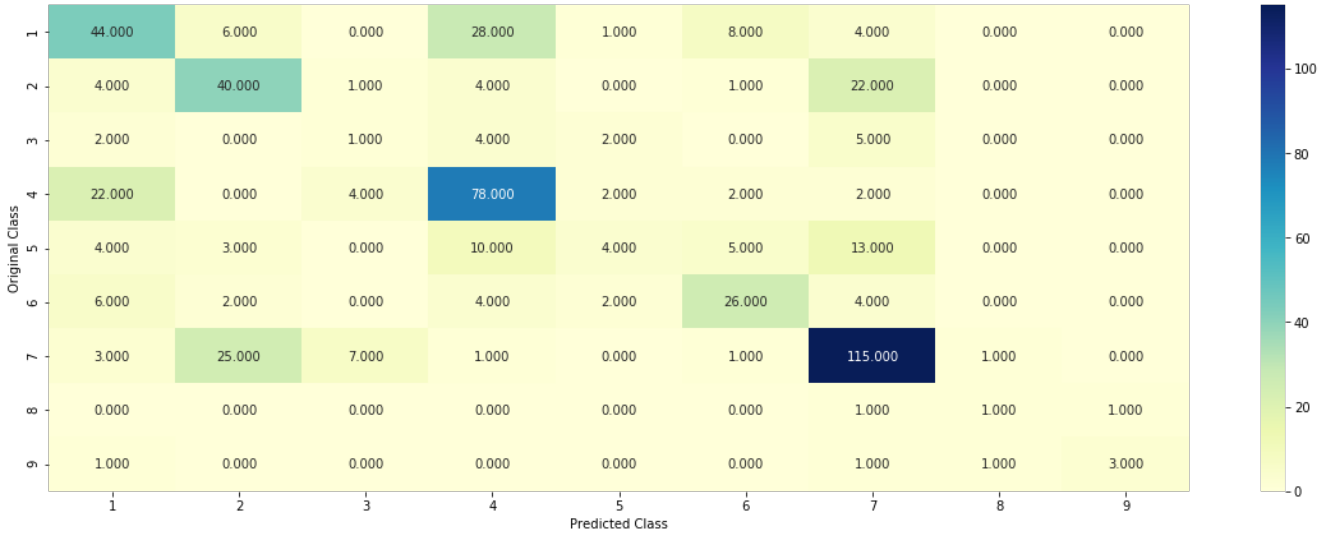
### 4.2.2. Testing the model with best hyper paramters

In [84]:

```
# 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-ne
```

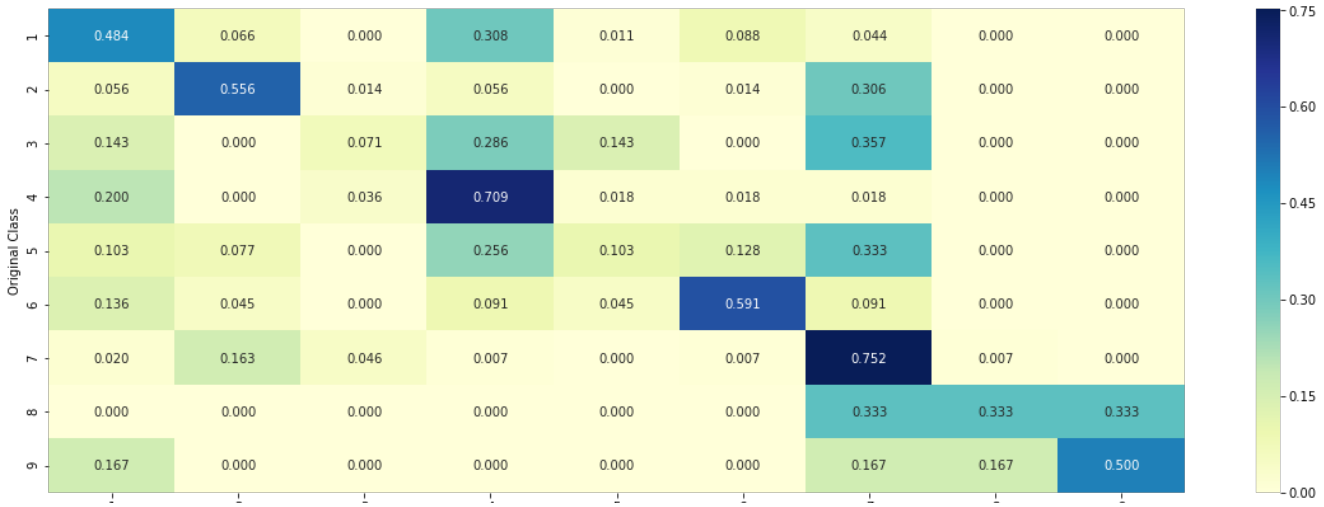
```
ighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.1683906197459837
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```




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


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




### 4.2.3.Sample Query point -1

In [85]:

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

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

```
Predicted Class : 6
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 [86]:

```
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 be
longs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [4 4 4
1 1 1 1 1 4 1 4]
Fequency of nearest points : Counter({1: 6, 4: 5})
```

## 4.3. Logistic Regression

### 4.3.1. With Class balancing

#### 4.3.1.1. Hyper paramter tuning

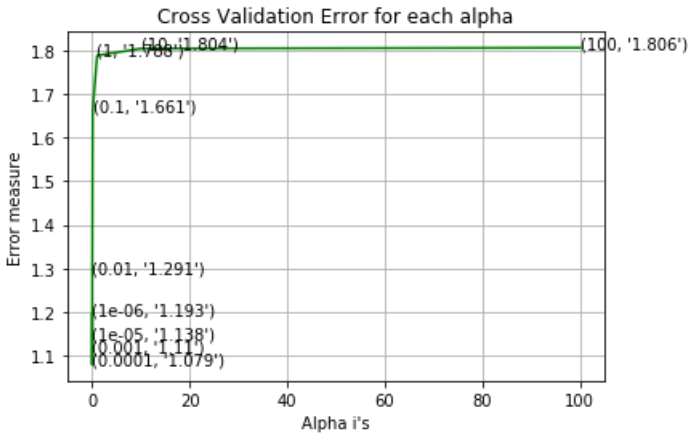
In [88]:

```
# 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_i
ter=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: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#--------------------------------


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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', 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', ran
dom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1932711807464214
for alpha = 1e-05
Log Loss : 1.1378221696519017
for alpha = 0.0001
```

```
Log Loss : 1.0785297549666297
for alpha = 0.001
Log Loss : 1.1098293125963181
for alpha = 0.01
Log Loss : 1.2907709634821625
for alpha = 0.1
Log Loss : 1.6607474979805439
for alpha = 1
Log Loss : 1.7876096621707669
for alpha = 10
Log Loss : 1.8037445123957971
for alpha = 100
Log Loss : 1.8055583179933101
```



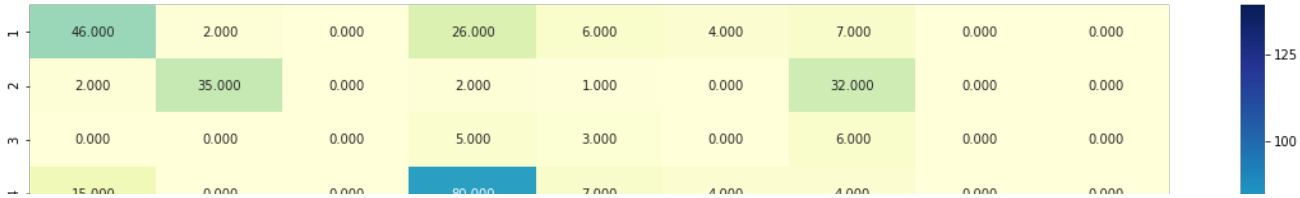

```
For values of best alpha =  0.0001 The train log loss is: 0.4330281638609431
For values of best alpha =  0.0001 The cross validation log loss is: 1.0785297549666297
For values of best alpha =  0.0001 The test log loss is: 0.9810022409425427
```

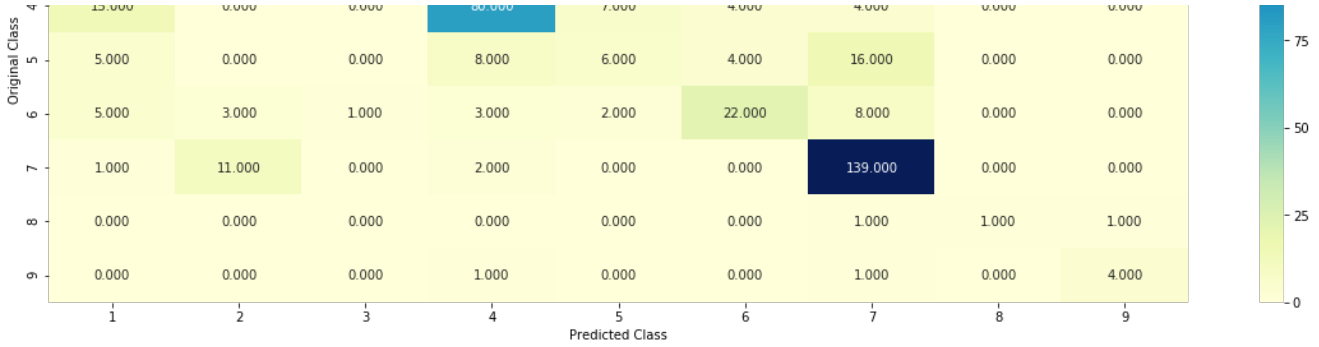
#### 4.3.1.2. Testing the model with best hyper paramters

In [89]:

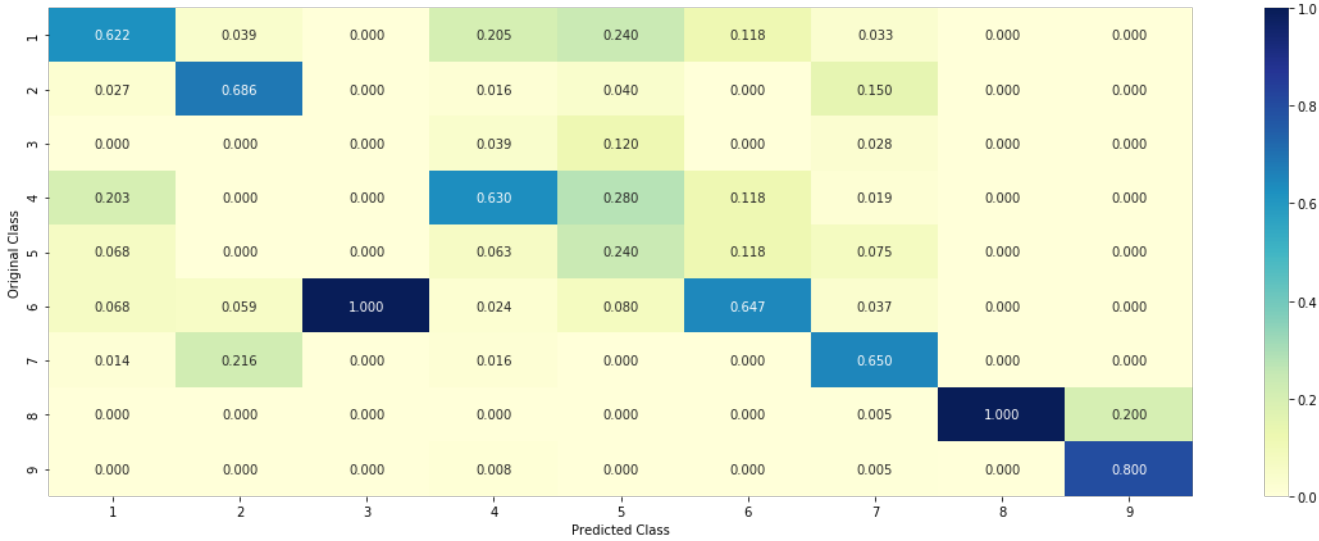
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.0785297549666297
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```

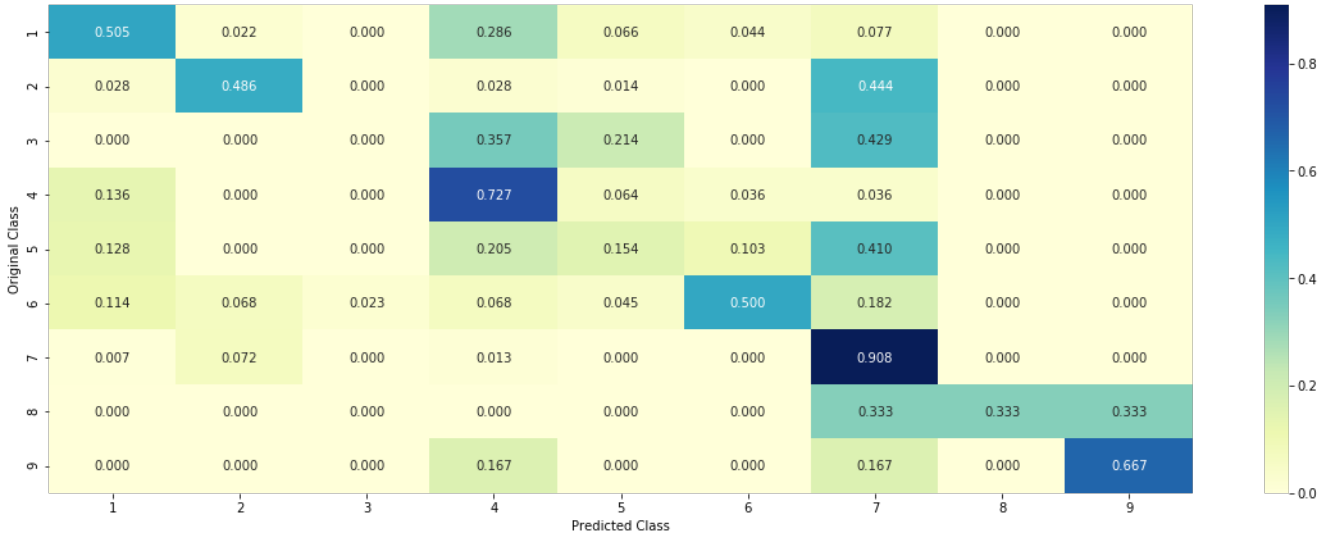





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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




#### 4.3.1.3. Feature Importance

In [90]:

```
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 [91]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0374 0.0158 0.0059 0.0135 0.0826 0.8388 0.0022 0.0019 0.0019]]
Actual Class : 6
--------------------------------------------------
109 Text feature [showing] present in test data point [True]
113 Text feature [substitutions] present in test data point [True]
118 Text feature [ring] present in test data point [True]
123 Text feature [significant] present in test data point [True]
133 Text feature [brca] present in test data point [True]
134 Text feature [42] present in test data point [True]
136 Text feature [basis] present in test data point [True]
138 Text feature [models] present in test data point [True]
150 Text feature [deleterious] present in test data point [True]
157 Text feature [studied] present in test data point [True]
161 Text feature [individuals] present in test data point [True]
163 Text feature [43] present in test data point [True]
165 Text feature [site] present in test data point [True]
168 Text feature [expected] present in test data point [True]
169 Text feature [57] present in test data point [True]
178 Text feature [decrease] present in test data point [True]
179 Text feature [classified] present in test data point [True]
181 Text feature [enzyme] present in test data point [True]
182 Text feature [observation] present in test data point [True]
188 Text feature [000] present in test data point [True]
189 Text feature [odds] present in test data point [True]
192 Text feature [predicted] present in test data point [True]
193 Text feature [reduction] present in test data point [True]
196 Text feature [overall] present in test data point [True]
208 Text feature [difference] present in test data point [True]
211 Text feature [history] present in test data point [True]
214 Text feature [state] present in test data point [True]
215 Text feature [identified] present in test data point [True]
221 Text feature [confer] present in test data point [True]
227 Text feature [model] present in test data point [True]
238 Text feature [evidence] present in test data point [True]
246 Text feature [five] present in test data point [True]
250 Text feature [important] present in test data point [True]
252 Text feature [loss] present in test data point [True]
262 Text feature [receptor] present in test data point [True]
263 Text feature [frequently] present in test data point [True]
265 Text feature [breast] present in test data point [True]
```

```
267 Text feature [missense] present in test data point [True]
273 Text feature [brca1] present in test data point [True]
274 Text feature [lower] present in test data point [True]
275 Text feature [family] present in test data point [True]
277 Text feature [risk] present in test data point [True]
283 Text feature [status] present in test data point [True]
284 Text feature [35] present in test data point [True]
285 Text feature [ovarian] present in test data point [True]
287 Text feature [group] present in test data point [True]
292 Text feature [pr] present in test data point [True]
294 Text feature [prior] present in test data point [True]
297 Text feature [30] present in test data point [True]
301 Text feature [combination] present in test data point [True]
304 Text feature [determined] present in test data point [True]
305 Text feature [clinically] present in test data point [True]
308 Text feature [including] present in test data point [True]
310 Text feature [time] present in test data point [True]
311 Text feature [factors] present in test data point [True]
316 Text feature [binding] present in test data point [True]
318 Text feature [population] present in test data point [True]
319 Text feature [75] present in test data point [True]
324 Text feature [given] present in test data point [True]
328 Text feature [affect] present in test data point [True]
337 Text feature [thus] present in test data point [True]
346 Text feature [members] present in test data point [True]
349 Text feature [20] present in test data point [True]
350 Text feature [average] present in test data point [True]
353 Text feature [four] present in test data point [True]
354 Text feature [conserved] present in test data point [True]
355 Text feature [acid] present in test data point [True]
361 Text feature [showed] present in test data point [True]
365 Text feature [56] present in test data point [True]
366 Text feature [approximately] present in test data point [True]
369 Text feature [100] present in test data point [True]
371 Text feature [none] present in test data point [True]
381 Text feature [70] present in test data point [True]
382 Text feature [studies] present in test data point [True]
386 Text feature [applied] present in test data point [True]
390 Text feature [factor] present in test data point [True]
391 Text feature [significance] present in test data point [True]
394 Text feature [considered] present in test data point [True]
397 Text feature [reaction] present in test data point [True]
398 Text feature [altered] present in test data point [True]
404 Text feature [following] present in test data point [True]
405 Text feature [34] present in test data point [True]
408 Text feature [17] present in test data point [True]
414 Text feature [tumors] present in test data point [True]
416 Text feature [reports] present in test data point [True]
417 Text feature [type] present in test data point [True]
419 Text feature [approach] present in test data point [True]
422 Text feature [isolated] present in test data point [True]
433 Text feature [frequency] present in test data point [True]
434 Text feature [majority] present in test data point [True]
438 Text feature [rates] present in test data point [True]
439 Text feature [methods] present in test data point [True]
440 Text feature [calculated] present in test data point [True]
441 Text feature [statistical] present in test data point [True]
443 Text feature [acids] present in test data point [True]
444 Text feature [stage] present in test data point [True]
445 Text feature [10] present in test data point [True]
446 Text feature [23] present in test data point [True]
449 Text feature [characteristics] present in test data point [True]
450 Text feature [change] present in test data point [True]
452 Text feature [go] present in test data point [True]
453 Text feature [potential] present in test data point [True]
457 Text feature [least] present in test data point [True]
458 Text feature [yet] present in test data point [True]
459 Text feature [classification] present in test data point [True]
468 Text feature [two] present in test data point [True]
469 Text feature [multiple] present in test data point [True]
470 Text feature [sequence] present in test data point [True]
471 Text feature [well] present in test data point [True]
474 Text feature [dna] present in test data point [True]
482 Text feature [according] present in test data point [True]
483 Text feature [developed] present in test data point [True]
484 Text feature [54] present in test data point [True]
487 Text feature [staining] present in test data point [True]
```

```
489 Text feature [using] present in test data point [True]
495 Text feature [greater] present in test data point [True]
496 Text feature [information] present in test data point [True]
Out of the top  500  features  117 are present in query point
```

#### *4.3.1.3.2. Incorrectly Classified point*

In [95]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0485 0.0076 0.0631 0.1426 0.0935 0.6352 0.0027 0.0031 0.0035]]
Actual Class : 3
--------------------------------------------------
57 Text feature [blue] present in test data point [True]
61 Text feature [e2] present in test data point [True]
80 Text feature [values] present in test data point [True]
81 Text feature [ligase] present in test data point [True]
83 Text feature [ubiquitin] present in test data point [True]
100 Text feature [binds] present in test data point [True]
112 Text feature [substitution] present in test data point [True]
113 Text feature [substitutions] present in test data point [True]
114 Text feature [interaction] present in test data point [True]
118 Text feature [ring] present in test data point [True]
120 Text feature [concentration] present in test data point [True]
121 Text feature [s2] present in test data point [True]
123 Text feature [significant] present in test data point [True]
130 Text feature [substrate] present in test data point [True]
135 Text feature [resistance] present in test data point [True]
136 Text feature [basis] present in test data point [True]
150 Text feature [deleterious] present in test data point [True]
161 Text feature [individuals] present in test data point [True]
162 Text feature [direct] present in test data point [True]
164 Text feature [development] present in test data point [True]
165 Text feature [site] present in test data point [True]
167 Text feature [mutagenesis] present in test data point [True]
168 Text feature [expected] present in test data point [True]
170 Text feature [free] present in test data point [True]
176 Text feature [60] present in test data point [True]
177 Text feature [suppression] present in test data point [True]
178 Text feature [decrease] present in test data point [True]
179 Text feature [classified] present in test data point [True]
181 Text feature [enzyme] present in test data point [True]
188 Text feature [000] present in test data point [True]
189 Text feature [odds] present in test data point [True]
190 Text feature [induce] present in test data point [True]
191 Text feature [resistant] present in test data point [True]
192 Text feature [predicted] present in test data point [True]
193 Text feature [reduction] present in test data point [True]
196 Text feature [overall] present in test data point [True]
200 Text feature [selection] present in test data point [True]
207 Text feature [sensitive] present in test data point [True]
208 Text feature [difference] present in test data point [True]
211 Text feature [history] present in test data point [True]
214 Text feature [state] present in test data point [True]
215 Text feature [identified] present in test data point [True]
219 Text feature [degradation] present in test data point [True]
221 Text feature [confer] present in test data point [True]
227 Text feature [model] present in test data point [True]
235 Text feature [interactions] present in test data point [True]
236 Text feature [helix] present in test data point [True]
242 Text feature [2006] present in test data point [True]
```

```
245 Text feature [single] present in test data point [True]
246 Text feature [five] present in test data point [True]
247 Text feature [mutated] present in test data point [True]
248 Text feature [luciferase] present in test data point [True]
250 Text feature [important] present in test data point [True]
252 Text feature [loss] present in test data point [True]
253 Text feature [95] present in test data point [True]
254 Text feature [copy] present in test data point [True]
256 Text feature [concentrations] present in test data point [True]
258 Text feature [terminus] present in test data point [True]
259 Text feature [structural] present in test data point [True]
261 Text feature [cause] present in test data point [True]
262 Text feature [receptor] present in test data point [True]
263 Text feature [frequently] present in test data point [True]
265 Text feature [breast] present in test data point [True]
266 Text feature [inhibit] present in test data point [True]
267 Text feature [missense] present in test data point [True]
273 Text feature [brca1] present in test data point [True]
274 Text feature [lower] present in test data point [True]
275 Text feature [family] present in test data point [True]
277 Text feature [risk] present in test data point [True]
279 Text feature [47] present in test data point [True]
281 Text feature [40] present in test data point [True]
282 Text feature [active] present in test data point [True]
283 Text feature [status] present in test data point [True]
284 Text feature [35] present in test data point [True]
285 Text feature [ovarian] present in test data point [True]
287 Text feature [group] present in test data point [True]
293 Text feature [red] present in test data point [True]
301 Text feature [combination] present in test data point [True]
303 Text feature [software] present in test data point [True]
304 Text feature [determined] present in test data point [True]
305 Text feature [clinically] present in test data point [True]
308 Text feature [including] present in test data point [True]
309 Text feature [selected] present in test data point [True]
310 Text feature [time] present in test data point [True]
311 Text feature [factors] present in test data point [True]
312 Text feature [strand] present in test data point [True]
314 Text feature [indicates] present in test data point [True]
315 Text feature [times] present in test data point [True]
316 Text feature [binding] present in test data point [True]
318 Text feature [population] present in test data point [True]
319 Text feature [75] present in test data point [True]
324 Text feature [given] present in test data point [True]
328 Text feature [affect] present in test data point [True]
329 Text feature [bind] present in test data point [True]
330 Text feature [double] present in test data point [True]
334 Text feature [recognition] present in test data point [True]
337 Text feature [thus] present in test data point [True]
338 Text feature [region] present in test data point [True]
340 Text feature [residues] present in test data point [True]
343 Text feature [formation] present in test data point [True]
344 Text feature [versus] present in test data point [True]
349 Text feature [20] present in test data point [True]
350 Text feature [average] present in test data point [True]
351 Text feature [44] present in test data point [True]
353 Text feature [four] present in test data point [True]
354 Text feature [conserved] present in test data point [True]
355 Text feature [acid] present in test data point [True]
356 Text feature [set] present in test data point [True]
357 Text feature [interface] present in test data point [True]
358 Text feature [associated] present in test data point [True]
361 Text feature [showed] present in test data point [True]
366 Text feature [approximately] present in test data point [True]
369 Text feature [100] present in test data point [True]
370 Text feature [stability] present in test data point [True]
371 Text feature [none] present in test data point [True]
374 Text feature [hydrophobic] present in test data point [True]
379 Text feature [required] present in test data point [True]
381 Text feature [70] present in test data point [True]
382 Text feature [studies] present in test data point [True]
383 Text feature [disease] present in test data point [True]
386 Text feature [applied] present in test data point [True]
388 Text feature [materials] present in test data point [True]
390 Text feature [factor] present in test data point [True]
391 Text feature [significance] present in test data point [True]
393 Text feature [structures] present in test data point [True]
```

```
394 Text feature [considered] present in test data point [True]
397 Text feature [reaction] present in test data point [True]
398 Text feature [altered] present in test data point [True]
399 Text feature [2004] present in test data point [True]
401 Text feature [sequencing] present in test data point [True]
403 Text feature [1a] present in test data point [True]
404 Text feature [following] present in test data point [True]
407 Text feature [2007] present in test data point [True]
408 Text feature [17] present in test data point [True]
409 Text feature [mechanism] present in test data point [True]
413 Text feature [mediated] present in test data point [True]
414 Text feature [tumors] present in test data point [True]
415 Text feature [structure] present in test data point [True]
417 Text feature [type] present in test data point [True]
418 Text feature [targeted] present in test data point [True]
419 Text feature [approach] present in test data point [True]
420 Text feature [relatively] present in test data point [True]
423 Text feature [seven] present in test data point [True]
424 Text feature [proportion] present in test data point [True]
426 Text feature [core] present in test data point [True]
428 Text feature [decreased] present in test data point [True]
430 Text feature [pathways] present in test data point [True]
431 Text feature [affinity] present in test data point [True]
432 Text feature [directed] present in test data point [True]
434 Text feature [majority] present in test data point [True]
435 Text feature [express] present in test data point [True]
436 Text feature [inactive] present in test data point [True]
437 Text feature [days] present in test data point [True]
439 Text feature [methods] present in test data point [True]
441 Text feature [statistical] present in test data point [True]
442 Text feature [subjected] present in test data point [True]
443 Text feature [acids] present in test data point [True]
445 Text feature [10] present in test data point [True]
446 Text feature [23] present in test data point [True]
447 Text feature [relevant] present in test data point [True]
448 Text feature [2000] present in test data point [True]
449 Text feature [characteristics] present in test data point [True]
450 Text feature [change] present in test data point [True]
452 Text feature [go] present in test data point [True]
453 Text feature [potential] present in test data point [True]
454 Text feature [05] present in test data point [True]
455 Text feature [interestingly] present in test data point [True]
456 Text feature [leading] present in test data point [True]
457 Text feature [least] present in test data point [True]
459 Text feature [classification] present in test data point [True]
460 Text feature [form] present in test data point [True]
461 Text feature [side] present in test data point [True]
462 Text feature [mm] present in test data point [True]
466 Text feature [impact] present in test data point [True]
467 Text feature [low] present in test data point [True]
468 Text feature [two] present in test data point [True]
469 Text feature [multiple] present in test data point [True]
470 Text feature [sequence] present in test data point [True]
471 Text feature [well] present in test data point [True]
473 Text feature [correlated] present in test data point [True]
474 Text feature [dna] present in test data point [True]
475 Text feature [types] present in test data point [True]
477 Text feature [invitrogen] present in test data point [True]
478 Text feature [database] present in test data point [True]
481 Text feature [embryonic] present in test data point [True]
482 Text feature [according] present in test data point [True]
483 Text feature [developed] present in test data point [True]
485 Text feature [response] present in test data point [True]
488 Text feature [screening] present in test data point [True]
489 Text feature [using] present in test data point [True]
490 Text feature [stable] present in test data point [True]
491 Text feature [remains] present in test data point [True]
494 Text feature [blotting] present in test data point [True]
495 Text feature [greater] present in test data point [True]
496 Text feature [information] present in test data point [True]
Out of the top  500  features  195 are present in query point
```

### 4.3.2. Without Class balancing

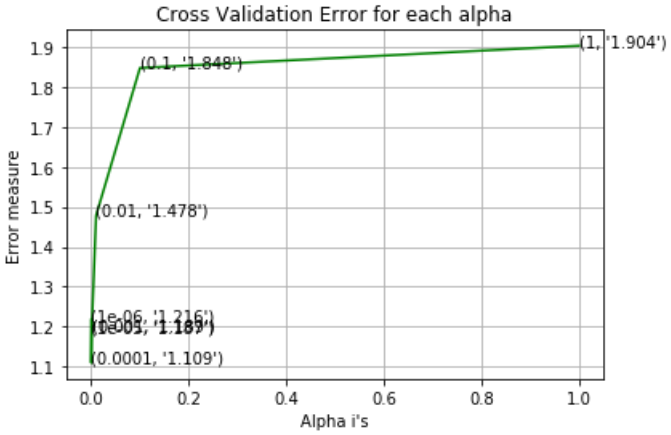
#### 4.3.2.1. Hyper paramter tuning

In [96]:

```
# 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_i
ter=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: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#-------------------------------



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

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

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.2163021891535317
for alpha = 1e-05
Log Loss : 1.1874100888380696
for alpha = 0.0001
Log Loss : 1.1087421687971688
for alpha = 0.001
Log Loss : 1.188840185384272
for alpha = 0.01
Log Loss : 1.4784341806959775
for alpha = 0.1
Log Loss : 1.8481338182592686
for alpha = 1
Log Loss : 1.9038001179795572
```



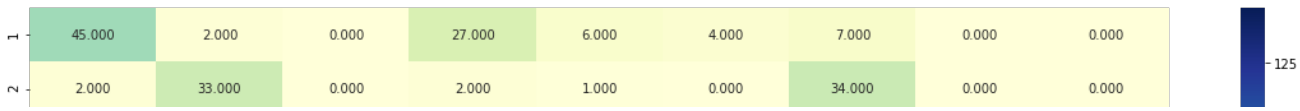

```
For values of best alpha =  0.0001 The train log loss is: 0.4260502987353983
For values of best alpha =  0.0001 The cross validation log loss is: 1.1087421687971688
For values of best alpha =  0.0001 The test log loss is: 0.9979324685162506
```

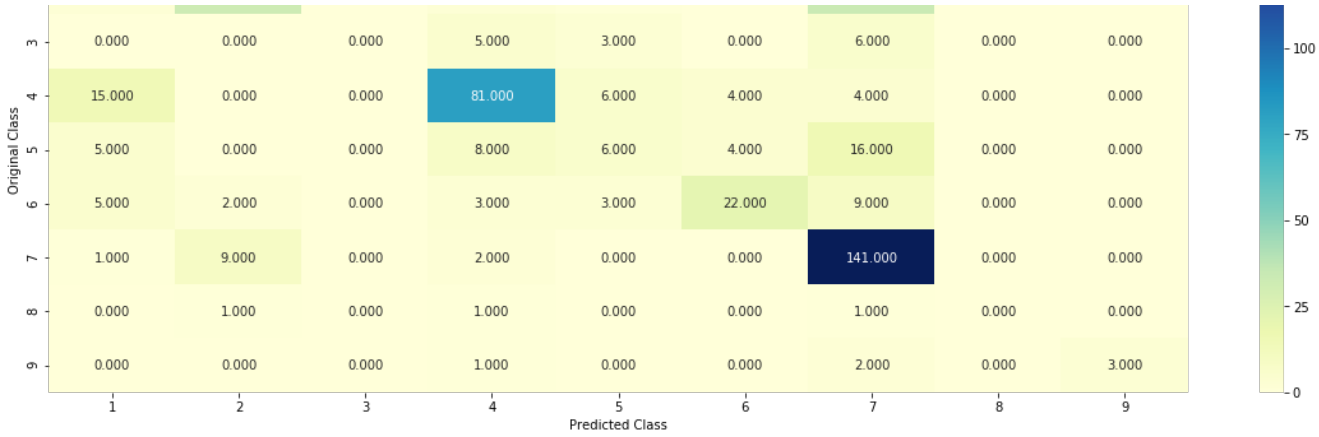
#### 4.3.2.2. Testing model with best hyper parameters

In [97]:

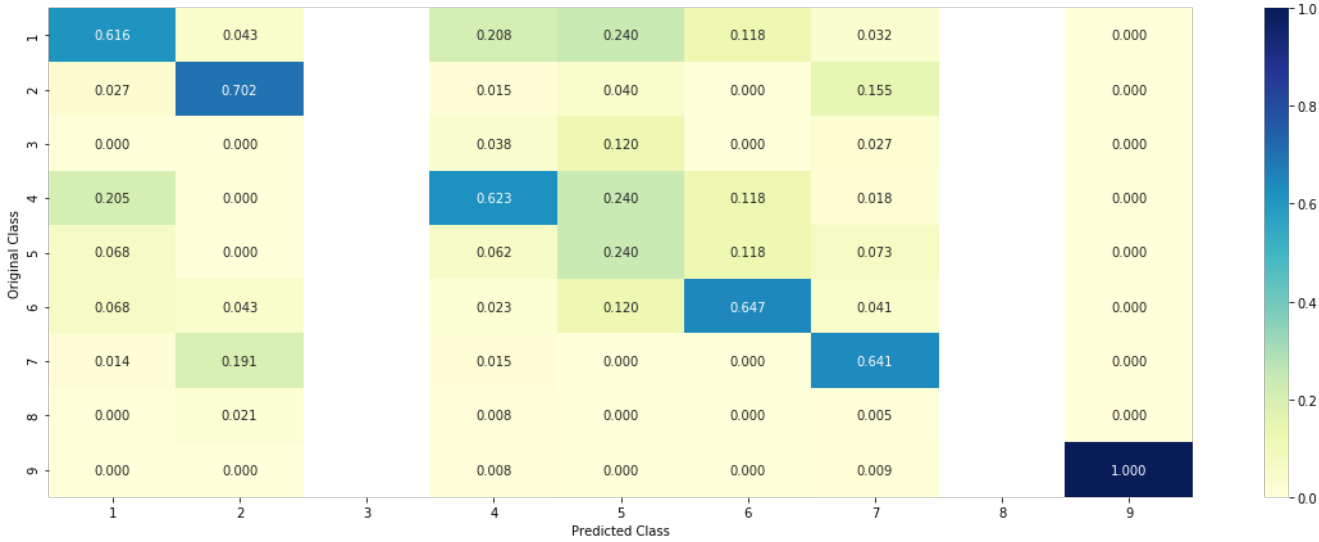
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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:
#------------------------------

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 : 1.1087421687971688
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```

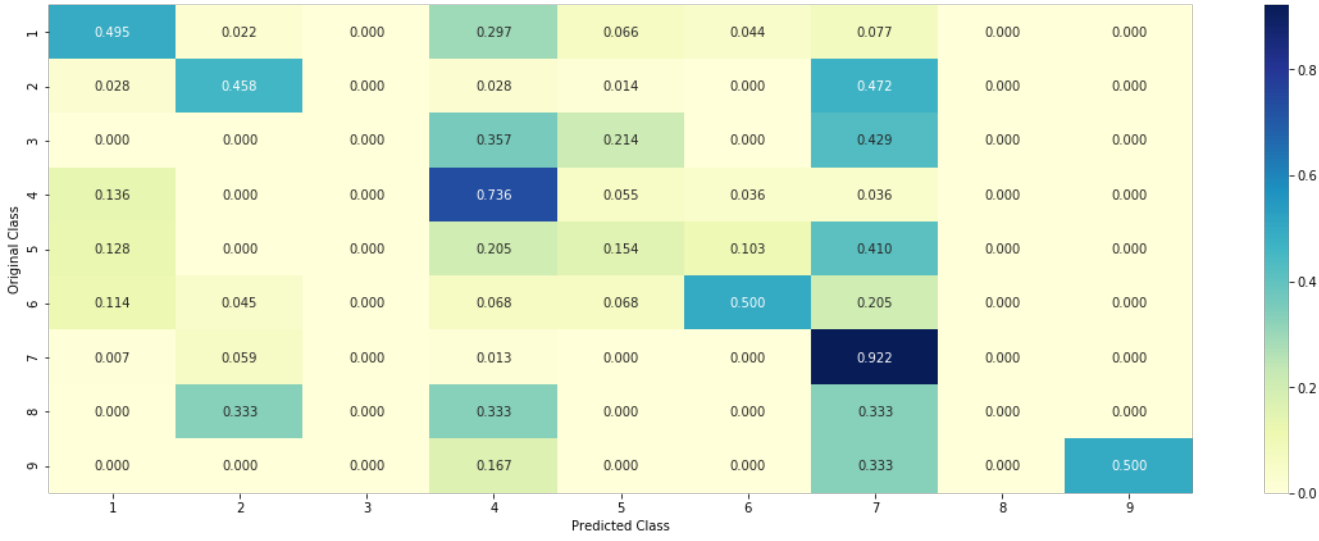





---------- Precision matrix (Columm Sum=1) ----------




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




#### 4.3.2.3. Feature Importance, Correctly Classified point

In [98]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
```

```
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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: [[3.860e-02 1.700e-02 3.400e-03 1.570e-02 7.790e-02 8.418e-01 2.600
e-03
  2.500e-03 5.000e-04]]
Actual Class : 6
--------------------------------------------------
111 Text feature [showing] present in test data point [True]
116 Text feature [substitutions] present in test data point [True]
117 Text feature [ring] present in test data point [True]
125 Text feature [brca] present in test data point [True]
127 Text feature [significant] present in test data point [True]
135 Text feature [42] present in test data point [True]
136 Text feature [basis] present in test data point [True]
137 Text feature [models] present in test data point [True]
150 Text feature [deleterious] present in test data point [True]
158 Text feature [studied] present in test data point [True]
162 Text feature [57] present in test data point [True]
165 Text feature [43] present in test data point [True]
167 Text feature [site] present in test data point [True]
168 Text feature [individuals] present in test data point [True]
169 Text feature [expected] present in test data point [True]
173 Text feature [decrease] present in test data point [True]
179 Text feature [enzyme] present in test data point [True]
180 Text feature [classified] present in test data point [True]
186 Text feature [odds] present in test data point [True]
188 Text feature [000] present in test data point [True]
190 Text feature [predicted] present in test data point [True]
193 Text feature [observation] present in test data point [True]
198 Text feature [overall] present in test data point [True]
199 Text feature [reduction] present in test data point [True]
200 Text feature [history] present in test data point [True]
211 Text feature [difference] present in test data point [True]
212 Text feature [confer] present in test data point [True]
213 Text feature [state] present in test data point [True]
220 Text feature [identified] present in test data point [True]
227 Text feature [model] present in test data point [True]
234 Text feature [evidence] present in test data point [True]
245 Text feature [five] present in test data point [True]
254 Text feature [loss] present in test data point [True]
260 Text feature [receptor] present in test data point [True]
264 Text feature [important] present in test data point [True]
265 Text feature [brca1] present in test data point [True]
267 Text feature [family] present in test data point [True]
268 Text feature [lower] present in test data point [True]
274 Text feature [missense] present in test data point [True]
278 Text feature [combination] present in test data point [True]
281 Text feature [35] present in test data point [True]
282 Text feature [frequently] present in test data point [True]
283 Text feature [ovarian] present in test data point [True]
285 Text feature [risk] present in test data point [True]
290 Text feature [clinically] present in test data point [True]
291 Text feature [breast] present in test data point [True]
294 Text feature [prior] present in test data point [True]
295 Text feature [group] present in test data point [True]
297 Text feature [75] present in test data point [True]
298 Text feature [status] present in test data point [True]
299 Text feature [pr] present in test data point [True]
304 Text feature [factors] present in test data point [True]
305 Text feature [time] present in test data point [True]
313 Text feature [30] present in test data point [True]
315 Text feature [determined] present in test data point [True]
317 Text feature [binding] present in test data point [True]
319 Text feature [including] present in test data point [True]
324 Text feature [affect] present in test data point [True]
325 Text feature [given] present in test data point [True]
328 Text feature [population] present in test data point [True]
343 Text feature [members] present in test data point [True]
```

```
344 Text feature [56] present in test data point [True]
350 Text feature [thus] present in test data point [True]
353 Text feature [20] present in test data point [True]
360 Text feature [70] present in test data point [True]
364 Text feature [none] present in test data point [True]
365 Text feature [four] present in test data point [True]
368 Text feature [average] present in test data point [True]
370 Text feature [conserved] present in test data point [True]
373 Text feature [acid] present in test data point [True]
375 Text feature [100] present in test data point [True]
377 Text feature [approximately] present in test data point [True]
385 Text feature [applied] present in test data point [True]
386 Text feature [studies] present in test data point [True]
387 Text feature [showed] present in test data point [True]
391 Text feature [characteristics] present in test data point [True]
393 Text feature [significance] present in test data point [True]
394 Text feature [factor] present in test data point [True]
397 Text feature [34] present in test data point [True]
398 Text feature [following] present in test data point [True]
400 Text feature [reaction] present in test data point [True]
403 Text feature [altered] present in test data point [True]
405 Text feature [considered] present in test data point [True]
412 Text feature [calculated] present in test data point [True]
413 Text feature [reports] present in test data point [True]
414 Text feature [17] present in test data point [True]
421 Text feature [statistical] present in test data point [True]
422 Text feature [type] present in test data point [True]
425 Text feature [rates] present in test data point [True]
427 Text feature [approach] present in test data point [True]
431 Text feature [acids] present in test data point [True]
432 Text feature [tumors] present in test data point [True]
433 Text feature [majority] present in test data point [True]
440 Text feature [change] present in test data point [True]
441 Text feature [isolated] present in test data point [True]
444 Text feature [23] present in test data point [True]
445 Text feature [stage] present in test data point [True]
447 Text feature [frequency] present in test data point [True]
448 Text feature [10] present in test data point [True]
455 Text feature [methods] present in test data point [True]
457 Text feature [potential] present in test data point [True]
458 Text feature [go] present in test data point [True]
459 Text feature [greater] present in test data point [True]
462 Text feature [two] present in test data point [True]
466 Text feature [classification] present in test data point [True]
467 Text feature [information] present in test data point [True]
469 Text feature [multiple] present in test data point [True]
470 Text feature [yet] present in test data point [True]
471 Text feature [54] present in test data point [True]
476 Text feature [least] present in test data point [True]
478 Text feature [sequence] present in test data point [True]
479 Text feature [dna] present in test data point [True]
480 Text feature [well] present in test data point [True]
488 Text feature [staining] present in test data point [True]
490 Text feature [according] present in test data point [True]
495 Text feature [amino] present in test data point [True]
496 Text feature [using] present in test data point [True]
497 Text feature [number] present in test data point [True]
498 Text feature [terminal] present in test data point [True]
499 Text feature [genes] present in test data point [True]
Out of the top  500  features  120 are present in query point
```

#### 4.3.2.4. Feature Importance, Inorrectly Classified point

In [101]:

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

```
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: [[4.870e-02 8.600e-03 4.020e-02 1.577e-01 1.017e-01 6.353e-01 3.900
e-03
  3.600e-03 4.000e-04]]
Actual Class : 3
--------------------------------------------------
56 Text feature [blue] present in test data point [True]
57 Text feature [e2] present in test data point [True]
74 Text feature [ligase] present in test data point [True]
76 Text feature [ubiquitin] present in test data point [True]
78 Text feature [values] present in test data point [True]
100 Text feature [binds] present in test data point [True]
110 Text feature [interaction] present in test data point [True]
112 Text feature [substitution] present in test data point [True]
116 Text feature [substitutions] present in test data point [True]
117 Text feature [ring] present in test data point [True]
119 Text feature [concentration] present in test data point [True]
121 Text feature [s2] present in test data point [True]
127 Text feature [significant] present in test data point [True]
130 Text feature [substrate] present in test data point [True]
134 Text feature [resistance] present in test data point [True]
136 Text feature [basis] present in test data point [True]
150 Text feature [deleterious] present in test data point [True]
163 Text feature [suppression] present in test data point [True]
167 Text feature [site] present in test data point [True]
168 Text feature [individuals] present in test data point [True]
169 Text feature [expected] present in test data point [True]
171 Text feature [direct] present in test data point [True]
172 Text feature [development] present in test data point [True]
173 Text feature [decrease] present in test data point [True]
174 Text feature [free] present in test data point [True]
175 Text feature [60] present in test data point [True]
179 Text feature [enzyme] present in test data point [True]
180 Text feature [classified] present in test data point [True]
183 Text feature [mutagenesis] present in test data point [True]
186 Text feature [odds] present in test data point [True]
188 Text feature [000] present in test data point [True]
190 Text feature [predicted] present in test data point [True]
191 Text feature [resistant] present in test data point [True]
194 Text feature [induce] present in test data point [True]
197 Text feature [selection] present in test data point [True]
198 Text feature [overall] present in test data point [True]
199 Text feature [reduction] present in test data point [True]
200 Text feature [history] present in test data point [True]
210 Text feature [sensitive] present in test data point [True]
211 Text feature [difference] present in test data point [True]
212 Text feature [confer] present in test data point [True]
213 Text feature [state] present in test data point [True]
220 Text feature [identified] present in test data point [True]
227 Text feature [model] present in test data point [True]
233 Text feature [interactions] present in test data point [True]
239 Text feature [inhibit] present in test data point [True]
241 Text feature [degradation] present in test data point [True]
245 Text feature [five] present in test data point [True]
246 Text feature [95] present in test data point [True]
247 Text feature [single] present in test data point [True]
250 Text feature [helix] present in test data point [True]
251 Text feature [luciferase] present in test data point [True]
254 Text feature [loss] present in test data point [True]
255 Text feature [copy] present in test data point [True]
256 Text feature [concentrations] present in test data point [True]
259 Text feature [2006] present in test data point [True]
260 Text feature [receptor] present in test data point [True]
264 Text feature [important] present in test data point [True]
265 Text feature [brca1] present in test data point [True]
266 Text feature [cause] present in test data point [True]
267 Text feature [family] present in test data point [True]
268 Text feature [lower] present in test data point [True]
271 Text feature [terminus] present in test data point [True]
272 Text feature [structural] present in test data point [True]
274 Text feature [missense] present in test data point [True]
276 Text feature [mutated] present in test data point [True]
278 Text feature [combination] present in test data point [True]
```

```
280 Text feature [47] present in test data point [True]
281 Text feature [35] present in test data point [True]
282 Text feature [frequently] present in test data point [True]
283 Text feature [ovarian] present in test data point [True]
285 Text feature [risk] present in test data point [True]
290 Text feature [clinically] present in test data point [True]
291 Text feature [breast] present in test data point [True]
295 Text feature [group] present in test data point [True]
296 Text feature [40] present in test data point [True]
297 Text feature [75] present in test data point [True]
298 Text feature [status] present in test data point [True]
300 Text feature [active] present in test data point [True]
301 Text feature [red] present in test data point [True]
304 Text feature [factors] present in test data point [True]
305 Text feature [time] present in test data point [True]
308 Text feature [selected] present in test data point [True]
314 Text feature [software] present in test data point [True]
315 Text feature [determined] present in test data point [True]
316 Text feature [times] present in test data point [True]
317 Text feature [binding] present in test data point [True]
319 Text feature [including] present in test data point [True]
322 Text feature [versus] present in test data point [True]
323 Text feature [strand] present in test data point [True]
324 Text feature [affect] present in test data point [True]
325 Text feature [given] present in test data point [True]
328 Text feature [population] present in test data point [True]
333 Text feature [recognition] present in test data point [True]
335 Text feature [bind] present in test data point [True]
336 Text feature [double] present in test data point [True]
337 Text feature [residues] present in test data point [True]
338 Text feature [indicates] present in test data point [True]
340 Text feature [region] present in test data point [True]
341 Text feature [44] present in test data point [True]
350 Text feature [thus] present in test data point [True]
353 Text feature [20] present in test data point [True]
355 Text feature [formation] present in test data point [True]
356 Text feature [set] present in test data point [True]
359 Text feature [interface] present in test data point [True]
360 Text feature [70] present in test data point [True]
364 Text feature [none] present in test data point [True]
365 Text feature [four] present in test data point [True]
368 Text feature [average] present in test data point [True]
370 Text feature [conserved] present in test data point [True]
372 Text feature [required] present in test data point [True]
373 Text feature [acid] present in test data point [True]
375 Text feature [100] present in test data point [True]
377 Text feature [approximately] present in test data point [True]
378 Text feature [hydrophobic] present in test data point [True]
381 Text feature [stability] present in test data point [True]
382 Text feature [associated] present in test data point [True]
385 Text feature [applied] present in test data point [True]
386 Text feature [studies] present in test data point [True]
387 Text feature [showed] present in test data point [True]
390 Text feature [structures] present in test data point [True]
391 Text feature [characteristics] present in test data point [True]
393 Text feature [significance] present in test data point [True]
394 Text feature [factor] present in test data point [True]
395 Text feature [2004] present in test data point [True]
398 Text feature [following] present in test data point [True]
400 Text feature [reaction] present in test data point [True]
401 Text feature [decreased] present in test data point [True]
403 Text feature [altered] present in test data point [True]
405 Text feature [considered] present in test data point [True]
406 Text feature [days] present in test data point [True]
407 Text feature [disease] present in test data point [True]
409 Text feature [structure] present in test data point [True]
410 Text feature [materials] present in test data point [True]
414 Text feature [17] present in test data point [True]
415 Text feature [targeted] present in test data point [True]
416 Text feature [1a] present in test data point [True]
417 Text feature [seven] present in test data point [True]
418 Text feature [mediated] present in test data point [True]
419 Text feature [express] present in test data point [True]
421 Text feature [statistical] present in test data point [True]
422 Text feature [type] present in test data point [True]
423 Text feature [core] present in test data point [True]
426 Text feature [05] present in test data point [True]
```

```
427 Text feature [approach] present in test data point [True]
428 Text feature [relatively] present in test data point [True]
429 Text feature [mechanism] present in test data point [True]
430 Text feature [pathways] present in test data point [True]
431 Text feature [acids] present in test data point [True]
432 Text feature [tumors] present in test data point [True]
433 Text feature [majority] present in test data point [True]
434 Text feature [affinity] present in test data point [True]
435 Text feature [subjected] present in test data point [True]
436 Text feature [sequencing] present in test data point [True]
437 Text feature [proportion] present in test data point [True]
438 Text feature [interestingly] present in test data point [True]
440 Text feature [change] present in test data point [True]
442 Text feature [inactive] present in test data point [True]
443 Text feature [2007] present in test data point [True]
444 Text feature [23] present in test data point [True]
446 Text feature [directed] present in test data point [True]
448 Text feature [10] present in test data point [True]
450 Text feature [form] present in test data point [True]
451 Text feature [mm] present in test data point [True]
452 Text feature [correlated] present in test data point [True]
453 Text feature [2000] present in test data point [True]
454 Text feature [relevant] present in test data point [True]
455 Text feature [methods] present in test data point [True]
457 Text feature [potential] present in test data point [True]
458 Text feature [go] present in test data point [True]
459 Text feature [greater] present in test data point [True]
462 Text feature [two] present in test data point [True]
466 Text feature [classification] present in test data point [True]
467 Text feature [information] present in test data point [True]
468 Text feature [database] present in test data point [True]
469 Text feature [multiple] present in test data point [True]
472 Text feature [impact] present in test data point [True]
474 Text feature [leading] present in test data point [True]
475 Text feature [side] present in test data point [True]
476 Text feature [least] present in test data point [True]
477 Text feature [invitrogen] present in test data point [True]
478 Text feature [sequence] present in test data point [True]
479 Text feature [dna] present in test data point [True]
480 Text feature [well] present in test data point [True]
482 Text feature [embryonic] present in test data point [True]
483 Text feature [types] present in test data point [True]
484 Text feature [remains] present in test data point [True]
487 Text feature [volume] present in test data point [True]
490 Text feature [according] present in test data point [True]
491 Text feature [response] present in test data point [True]
492 Text feature [observations] present in test data point [True]
494 Text feature [screening] present in test data point [True]
495 Text feature [amino] present in test data point [True]
496 Text feature [using] present in test data point [True]
497 Text feature [number] present in test data point [True]
498 Text feature [terminal] present in test data point [True]
499 Text feature [genes] present in test data point [True]
Out of the top  500  features  197 are present in query point
```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper paramter tuning

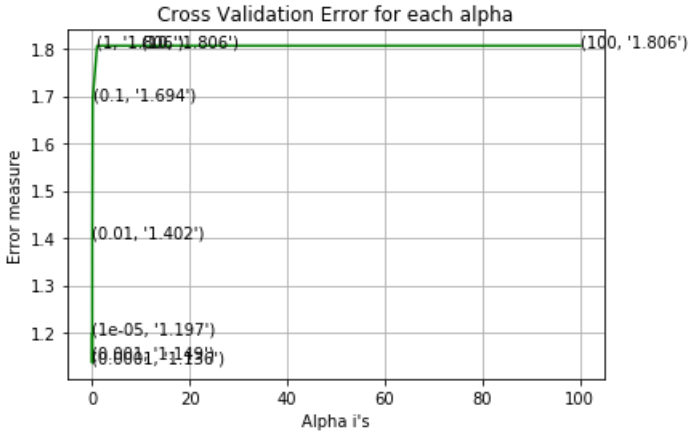
In [102]:

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

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
```

```
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.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', r
andom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for C = 1e-05
Log Loss : 1.1974297396222382
for C = 0.0001
Log Loss : 1.1363434876121787
for C = 0.001
Log Loss : 1.1485221302059576
for C = 0.01
Log Loss : 1.402249776882556
for C = 0.1
Log Loss : 1.6936976470723217
```

```
for C = 1
Log Loss : 1.8059193192017937
for C = 10
Log Loss : 1.8059370936694696
for C = 100
Log Loss : 1.8059371406269502
```



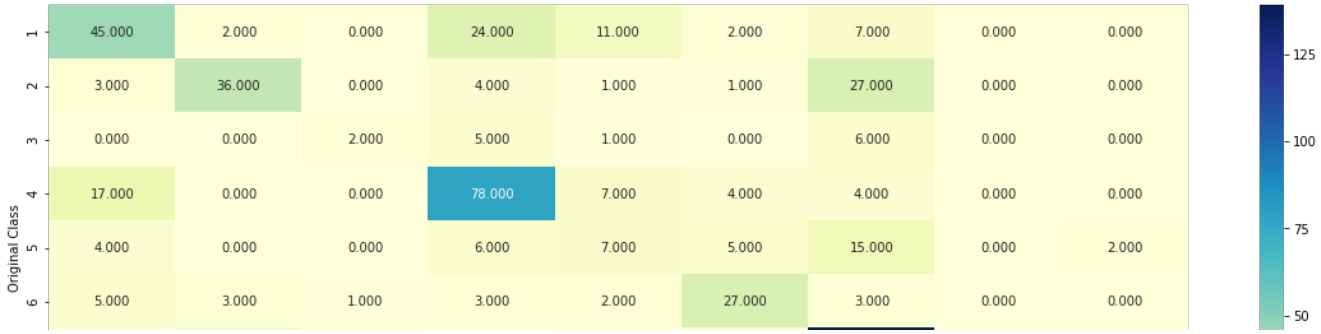

```
For values of best alpha =  0.0001 The train log loss is: 0.4514238241445011
For values of best alpha =  0.0001 The cross validation log loss is: 1.1363434876121787
For values of best alpha =  0.0001 The test log loss is: 1.0399302501554777
```

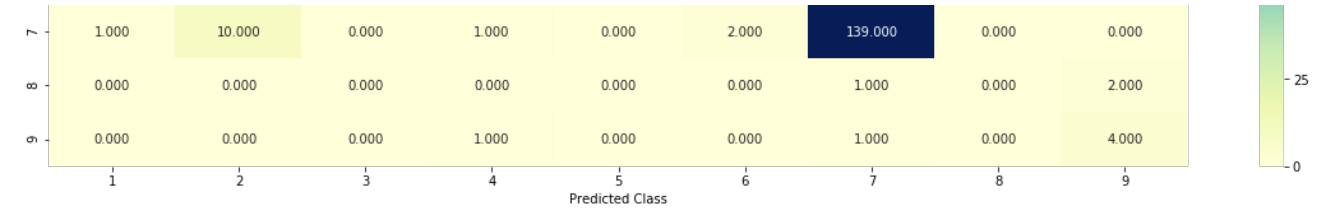
### 4.4.2. Testing model with best hyper parameters

In [103]:

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

# ----------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

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


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

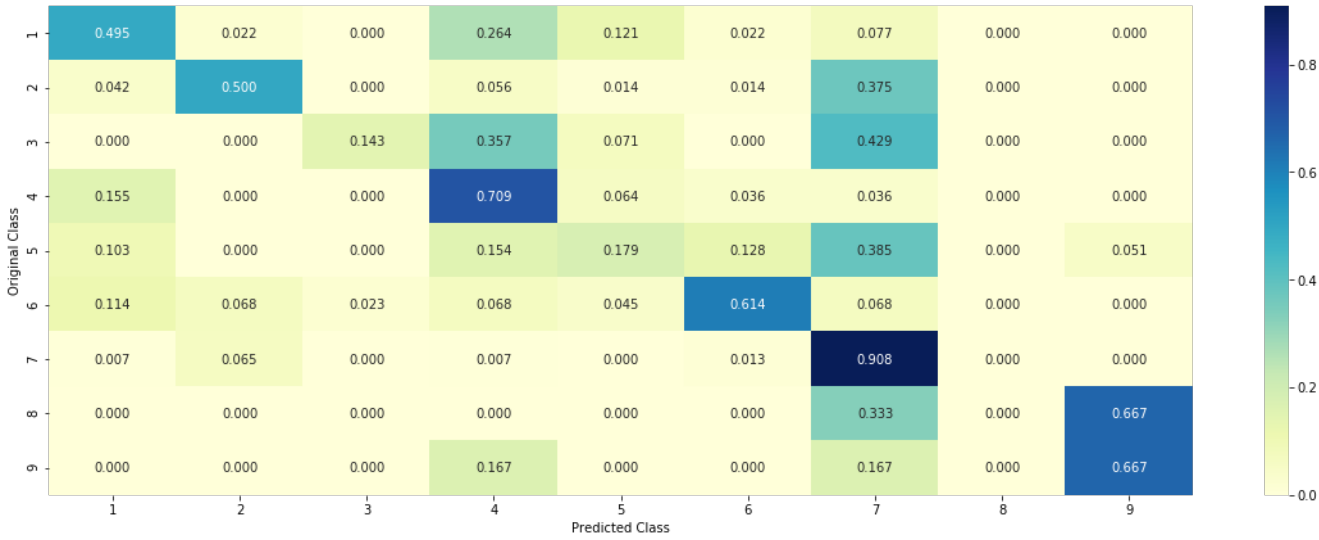
```
Log loss : 1.1363434876121787
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```






```
-------------------- 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_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
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: [[0.0359 0.0444 0.0208 0.057  0.0396 0.7774 0.0214 0.0019 0.0015]]
Actual Class : 6
--------------------------------------------------
108 Text feature [models] present in test data point [True]
113 Text feature [substitutions] present in test data point [True]
115 Text feature [brca] present in test data point [True]
118 Text feature [classified] present in test data point [True]
119 Text feature [site] present in test data point [True]
120 Text feature [ring] present in test data point [True]
123 Text feature [deleterious] present in test data point [True]
124 Text feature [observation] present in test data point [True]
128 Text feature [studied] present in test data point [True]
129 Text feature [showing] present in test data point [True]
130 Text feature [basis] present in test data point [True]
134 Text feature [significant] present in test data point [True]
135 Text feature [43] present in test data point [True]
202 Text feature [confer] present in test data point [True]
207 Text feature [57] present in test data point [True]
208 Text feature [42] present in test data point [True]
210 Text feature [predicted] present in test data point [True]
211 Text feature [000] present in test data point [True]
212 Text feature [enzyme] present in test data point [True]
215 Text feature [identified] present in test data point [True]
216 Text feature [showed] present in test data point [True]
220 Text feature [reports] present in test data point [True]
221 Text feature [individuals] present in test data point [True]
223 Text feature [expected] present in test data point [True]
227 Text feature [odds] present in test data point [True]
228 Text feature [overall] present in test data point [True]
236 Text feature [ovarian] present in test data point [True]
239 Text feature [loss] present in test data point [True]
243 Text feature [applied] present in test data point [True]
244 Text feature [reduction] present in test data point [True]
246 Text feature [receptor] present in test data point [True]
247 Text feature [five] present in test data point [True]
248 Text feature [missense] present in test data point [True]
251 Text feature [35] present in test data point [True]
252 Text feature [difference] present in test data point [True]
258 Text feature [type] present in test data point [True]
260 Text feature [pr] present in test data point [True]
261 Text feature [none] present in test data point [True]
262 Text feature [tumors] present in test data point [True]
265 Text feature [frequently] present in test data point [True]
266 Text feature [binding] present in test data point [True]
268 Text feature [increased] present in test data point [True]
274 Text feature [including] present in test data point [True]
277 Text feature [characteristics] present in test data point [True]
279 Text feature [acid] present in test data point [True]
282 Text feature [altered] present in test data point [True]
283 Text feature [75] present in test data point [True]
284 Text feature [breast] present in test data point [True]
288 Text feature [model] present in test data point [True]
289 Text feature [according] present in test data point [True]
291 Text feature [decrease] present in test data point [True]
292 Text feature [lower] present in test data point [True]
294 Text feature [factor] present in test data point [True]
295 Text feature [status] present in test data point [True]
298 Text feature [thus] present in test data point [True]
299 Text feature [four] present in test data point [True]
301 Text feature [affect] present in test data point [True]
304 Text feature [rates] present in test data point [True]
305 Text feature [20] present in test data point [True]
306 Text feature [staining] present in test data point [True]
307 Text feature [brca1] present in test data point [True]
308 Text feature [history] present in test data point [True]
310 Text feature [yet] present in test data point [True]
313 Text feature [isolated] present in test data point [True]
315 Text feature [family] present in test data point [True]
316 Text feature [members] present in test data point [True]
325 Text feature [greater] present in test data point [True]
```

```
328 Text feature [significance] present in test data point [True]
331 Text feature [statistical] present in test data point [True]
333 Text feature [acids] present in test data point [True]
334 Text feature [studies] present in test data point [True]
335 Text feature [classification] present in test data point [True]
338 Text feature [majority] present in test data point [True]
339 Text feature [two] present in test data point [True]
341 Text feature [given] present in test data point [True]
342 Text feature [determined] present in test data point [True]
343 Text feature [calculated] present in test data point [True]
345 Text feature [clinically] present in test data point [True]
346 Text feature [wild] present in test data point [True]
347 Text feature [17] present in test data point [True]
353 Text feature [combination] present in test data point [True]
360 Text feature [time] present in test data point [True]
362 Text feature [numbers] present in test data point [True]
365 Text feature [prior] present in test data point [True]
367 Text feature [used] present in test data point [True]
368 Text feature [evidence] present in test data point [True]
370 Text feature [information] present in test data point [True]
371 Text feature [following] present in test data point [True]
374 Text feature [30] present in test data point [True]
375 Text feature [reaction] present in test data point [True]
376 Text feature [methods] present in test data point [True]
377 Text feature [factors] present in test data point [True]
379 Text feature [risk] present in test data point [True]
381 Text feature [frequency] present in test data point [True]
385 Text feature [published] present in test data point [True]
386 Text feature [approximately] present in test data point [True]
388 Text feature [buffer] present in test data point [True]
389 Text feature [group] present in test data point [True]
391 Text feature [average] present in test data point [True]
392 Text feature [well] present in test data point [True]
394 Text feature [considered] present in test data point [True]
395 Text feature [introduction] present in test data point [True]
396 Text feature [developed] present in test data point [True]
401 Text feature [cases] present in test data point [True]
403 Text feature [finally] present in test data point [True]
406 Text feature [example] present in test data point [True]
409 Text feature [six] present in test data point [True]
410 Text feature [population] present in test data point [True]
411 Text feature [neutral] present in test data point [True]
413 Text feature [important] present in test data point [True]
414 Text feature [using] present in test data point [True]
416 Text feature [change] present in test data point [True]
417 Text feature [amino] present in test data point [True]
423 Text feature [10] present in test data point [True]
427 Text feature [least] present in test data point [True]
436 Text feature [university] present in test data point [True]
438 Text feature [100] present in test data point [True]
441 Text feature [frequent] present in test data point [True]
443 Text feature [34] present in test data point [True]
446 Text feature [stage] present in test data point [True]
452 Text feature [number] present in test data point [True]
458 Text feature [variants] present in test data point [True]
462 Text feature [identification] present in test data point [True]
463 Text feature [different] present in test data point [True]
468 Text feature [basal] present in test data point [True]
470 Text feature [11] present in test data point [True]
473 Text feature [sequence] present in test data point [True]
474 Text feature [41] present in test data point [True]
478 Text feature [approach] present in test data point [True]
479 Text feature [state] present in test data point [True]
480 Text feature [01] present in test data point [True]
482 Text feature [18] present in test data point [True]
483 Text feature [conserved] present in test data point [True]
484 Text feature [controls] present in test data point [True]
485 Text feature [likely] present in test data point [True]
491 Text feature [normal] present in test data point [True]
493 Text feature [genes] present in test data point [True]
495 Text feature [testing] present in test data point [True]
Out of the top  500  features  138 are present in query point
```

#### 4.3.3.2. For Incorrectly classified point

In [107]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0337 0.0591 0.0982 0.0895 0.0579 0.6342 0.0201 0.0024 0.0049]]
Actual Class : 3
--------------------------------------------------
50 Text feature [blue] present in test data point [True]
51 Text feature [e2] present in test data point [True]
105 Text feature [ligase] present in test data point [True]
106 Text feature [ubiquitin] present in test data point [True]
107 Text feature [values] present in test data point [True]
109 Text feature [interaction] present in test data point [True]
111 Text feature [direct] present in test data point [True]
113 Text feature [substitutions] present in test data point [True]
114 Text feature [substitution] present in test data point [True]
116 Text feature [s2] present in test data point [True]
117 Text feature [binds] present in test data point [True]
118 Text feature [classified] present in test data point [True]
119 Text feature [site] present in test data point [True]
120 Text feature [ring] present in test data point [True]
123 Text feature [deleterious] present in test data point [True]
125 Text feature [degradation] present in test data point [True]
130 Text feature [basis] present in test data point [True]
133 Text feature [mutated] present in test data point [True]
134 Text feature [significant] present in test data point [True]
199 Text feature [selection] present in test data point [True]
201 Text feature [2006] present in test data point [True]
202 Text feature [confer] present in test data point [True]
203 Text feature [suppression] present in test data point [True]
204 Text feature [substrate] present in test data point [True]
205 Text feature [free] present in test data point [True]
206 Text feature [concentration] present in test data point [True]
209 Text feature [resistance] present in test data point [True]
210 Text feature [predicted] present in test data point [True]
211 Text feature [000] present in test data point [True]
212 Text feature [enzyme] present in test data point [True]
213 Text feature [terminus] present in test data point [True]
215 Text feature [identified] present in test data point [True]
216 Text feature [showed] present in test data point [True]
217 Text feature [interactions] present in test data point [True]
219 Text feature [mutagenesis] present in test data point [True]
221 Text feature [individuals] present in test data point [True]
223 Text feature [expected] present in test data point [True]
226 Text feature [sensitive] present in test data point [True]
227 Text feature [odds] present in test data point [True]
228 Text feature [overall] present in test data point [True]
229 Text feature [recognition] present in test data point [True]
231 Text feature [60] present in test data point [True]
232 Text feature [interface] present in test data point [True]
236 Text feature [ovarian] present in test data point [True]
237 Text feature [indicates] present in test data point [True]
238 Text feature [double] present in test data point [True]
239 Text feature [loss] present in test data point [True]
242 Text feature [luciferase] present in test data point [True]
243 Text feature [applied] present in test data point [True]
244 Text feature [reduction] present in test data point [True]
246 Text feature [receptor] present in test data point [True]
247 Text feature [five] present in test data point [True]
248 Text feature [missense] present in test data point [True]
249 Text feature [strand] present in test data point [True]
250 Text feature [times] present in test data point [True]
251 Text feature [35] present in test data point [True]
252 Text feature [difference] present in test data point [True]
```

```
256 Text feature [2004] present in test data point [True]
257 Text feature [copy] present in test data point [True]
258 Text feature [type] present in test data point [True]
261 Text feature [none] present in test data point [True]
262 Text feature [tumors] present in test data point [True]
264 Text feature [software] present in test data point [True]
265 Text feature [frequently] present in test data point [True]
266 Text feature [binding] present in test data point [True]
268 Text feature [increased] present in test data point [True]
270 Text feature [structural] present in test data point [True]
271 Text feature [development] present in test data point [True]
274 Text feature [including] present in test data point [True]
276 Text feature [inhibit] present in test data point [True]
277 Text feature [characteristics] present in test data point [True]
279 Text feature [acid] present in test data point [True]
282 Text feature [altered] present in test data point [True]
283 Text feature [75] present in test data point [True]
284 Text feature [breast] present in test data point [True]
285 Text feature [resistant] present in test data point [True]
286 Text feature [relevant] present in test data point [True]
287 Text feature [2007] present in test data point [True]
288 Text feature [model] present in test data point [True]
289 Text feature [according] present in test data point [True]
291 Text feature [decrease] present in test data point [True]
292 Text feature [lower] present in test data point [True]
294 Text feature [factor] present in test data point [True]
295 Text feature [status] present in test data point [True]
298 Text feature [thus] present in test data point [True]
299 Text feature [four] present in test data point [True]
300 Text feature [associated] present in test data point [True]
301 Text feature [affect] present in test data point [True]
302 Text feature [formation] present in test data point [True]
305 Text feature [20] present in test data point [True]
307 Text feature [brca1] present in test data point [True]
308 Text feature [history] present in test data point [True]
311 Text feature [helix] present in test data point [True]
315 Text feature [family] present in test data point [True]
318 Text feature [red] present in test data point [True]
325 Text feature [greater] present in test data point [True]
326 Text feature [leading] present in test data point [True]
328 Text feature [significance] present in test data point [True]
329 Text feature [2000] present in test data point [True]
331 Text feature [statistical] present in test data point [True]
333 Text feature [acids] present in test data point [True]
334 Text feature [studies] present in test data point [True]
335 Text feature [classification] present in test data point [True]
336 Text feature [95] present in test data point [True]
337 Text feature [induce] present in test data point [True]
338 Text feature [majority] present in test data point [True]
339 Text feature [two] present in test data point [True]
340 Text feature [cause] present in test data point [True]
341 Text feature [given] present in test data point [True]
342 Text feature [determined] present in test data point [True]
344 Text feature [bind] present in test data point [True]
345 Text feature [clinically] present in test data point [True]
346 Text feature [wild] present in test data point [True]
347 Text feature [17] present in test data point [True]
348 Text feature [relatively] present in test data point [True]
352 Text feature [examined] present in test data point [True]
353 Text feature [combination] present in test data point [True]
356 Text feature [upon] present in test data point [True]
358 Text feature [concentrations] present in test data point [True]
359 Text feature [stability] present in test data point [True]
360 Text feature [time] present in test data point [True]
367 Text feature [used] present in test data point [True]
369 Text feature [1a] present in test data point [True]
370 Text feature [information] present in test data point [True]
371 Text feature [following] present in test data point [True]
372 Text feature [47] present in test data point [True]
375 Text feature [reaction] present in test data point [True]
376 Text feature [methods] present in test data point [True]
377 Text feature [factors] present in test data point [True]
378 Text feature [support] present in test data point [True]
379 Text feature [risk] present in test data point [True]
380 Text feature [subjected] present in test data point [True]
384 Text feature [sequencing] present in test data point [True]
385 Text feature [published] present in test data point [True]
```

```
386 Text feature [approximately] present in test data point [True]
389 Text feature [group] present in test data point [True]
391 Text feature [average] present in test data point [True]
392 Text feature [well] present in test data point [True]
394 Text feature [considered] present in test data point [True]
395 Text feature [introduction] present in test data point [True]
396 Text feature [developed] present in test data point [True]
397 Text feature [days] present in test data point [True]
399 Text feature [versus] present in test data point [True]
400 Text feature [proportion] present in test data point [True]
401 Text feature [cases] present in test data point [True]
404 Text feature [figures] present in test data point [True]
406 Text feature [example] present in test data point [True]
407 Text feature [single] present in test data point [True]
408 Text feature [disease] present in test data point [True]
409 Text feature [six] present in test data point [True]
410 Text feature [population] present in test data point [True]
411 Text feature [neutral] present in test data point [True]
412 Text feature [invitrogen] present in test data point [True]
413 Text feature [important] present in test data point [True]
414 Text feature [using] present in test data point [True]
415 Text feature [region] present in test data point [True]
416 Text feature [change] present in test data point [True]
417 Text feature [amino] present in test data point [True]
418 Text feature [form] present in test data point [True]
419 Text feature [set] present in test data point [True]
421 Text feature [per] present in test data point [True]
422 Text feature [structures] present in test data point [True]
423 Text feature [10] present in test data point [True]
425 Text feature [selected] present in test data point [True]
426 Text feature [culture] present in test data point [True]
427 Text feature [least] present in test data point [True]
428 Text feature [response] present in test data point [True]
429 Text feature [materials] present in test data point [True]
430 Text feature [interestingly] present in test data point [True]
431 Text feature [40] present in test data point [True]
434 Text feature [44] present in test data point [True]
435 Text feature [database] present in test data point [True]
436 Text feature [university] present in test data point [True]
438 Text feature [100] present in test data point [True]
439 Text feature [mm] present in test data point [True]
441 Text feature [frequent] present in test data point [True]
445 Text feature [pathway] present in test data point [True]
448 Text feature [embryonic] present in test data point [True]
450 Text feature [types] present in test data point [True]
451 Text feature [affinity] present in test data point [True]
452 Text feature [number] present in test data point [True]
453 Text feature [role] present in test data point [True]
454 Text feature [series] present in test data point [True]
455 Text feature [seven] present in test data point [True]
456 Text feature [low] present in test data point [True]
457 Text feature [correlated] present in test data point [True]
458 Text feature [variants] present in test data point [True]
460 Text feature [confirmed] present in test data point [True]
461 Text feature [mediated] present in test data point [True]
462 Text feature [identification] present in test data point [True]
463 Text feature [different] present in test data point [True]
465 Text feature [2005] present in test data point [True]
466 Text feature [ml] present in test data point [True]
467 Text feature [active] present in test data point [True]
470 Text feature [11] present in test data point [True]
471 Text feature [required] present in test data point [True]
472 Text feature [impact] present in test data point [True]
473 Text feature [sequence] present in test data point [True]
475 Text feature [blotting] present in test data point [True]
476 Text feature [residues] present in test data point [True]
477 Text feature [fig] present in test data point [True]
478 Text feature [approach] present in test data point [True]
479 Text feature [state] present in test data point [True]
480 Text feature [01] present in test data point [True]
481 Text feature [directed] present in test data point [True]
482 Text feature [18] present in test data point [True]
483 Text feature [conserved] present in test data point [True]
484 Text feature [controls] present in test data point [True]
485 Text feature [likely] present in test data point [True]
486 Text feature [activity] present in test data point [True]
489 Text feature [kit] present in test data point [True]
```

```
490 Text feature [involved] present in test data point [True]
491 Text feature [normal] present in test data point [True]
492 Text feature [targets] present in test data point [True]
493 Text feature [genes] present in test data point [True]
494 Text feature [screening] present in test data point [True]
495 Text feature [testing] present in test data point [True]
496 Text feature [relative] present in test data point [True]
497 Text feature [generated] present in test data point [True]
Out of the top  500  features  219 are present in query point
```

## 4.5 Random Forest Classifier

### 4.5.1. Hyper paramter tuning (With One hot Encoding)

In [108]:

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

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

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

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


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

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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
```

```
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2617453895093356
for n_estimators = 100 and max depth =  10
Log Loss : 1.3019463090580705
for n_estimators = 200 and max depth =  5
Log Loss : 1.2493140370153533
for n_estimators = 200 and max depth =  10
Log Loss : 1.2970076157971044
for n_estimators = 500 and max depth =  5
Log Loss : 1.2463308491627996
for n_estimators = 500 and max depth =  10
Log Loss : 1.2886169624222013
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2468160662896974
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2843507228906608
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2436768923560293
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2845348720758512
For values of best estimator =  2000 The train log loss is: 0.8318932278313419
For values of best estimator =  2000 The cross validation log loss is: 1.2436768923560293
For values of best estimator =  2000 The test log loss is: 1.1535319553112264
```

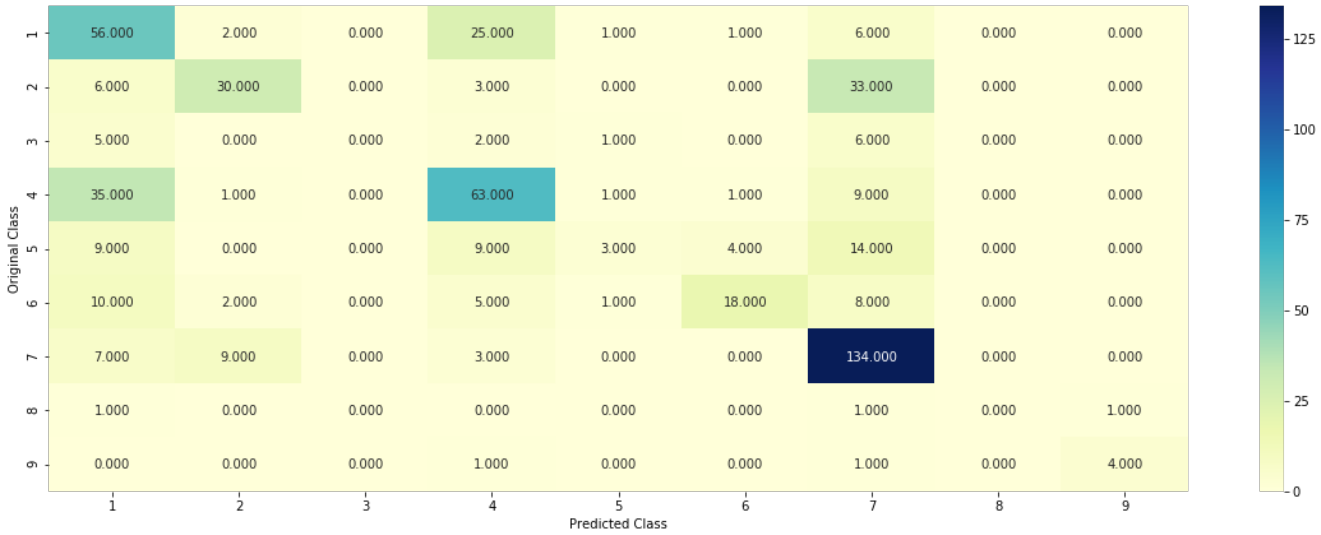
### 4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [109]:

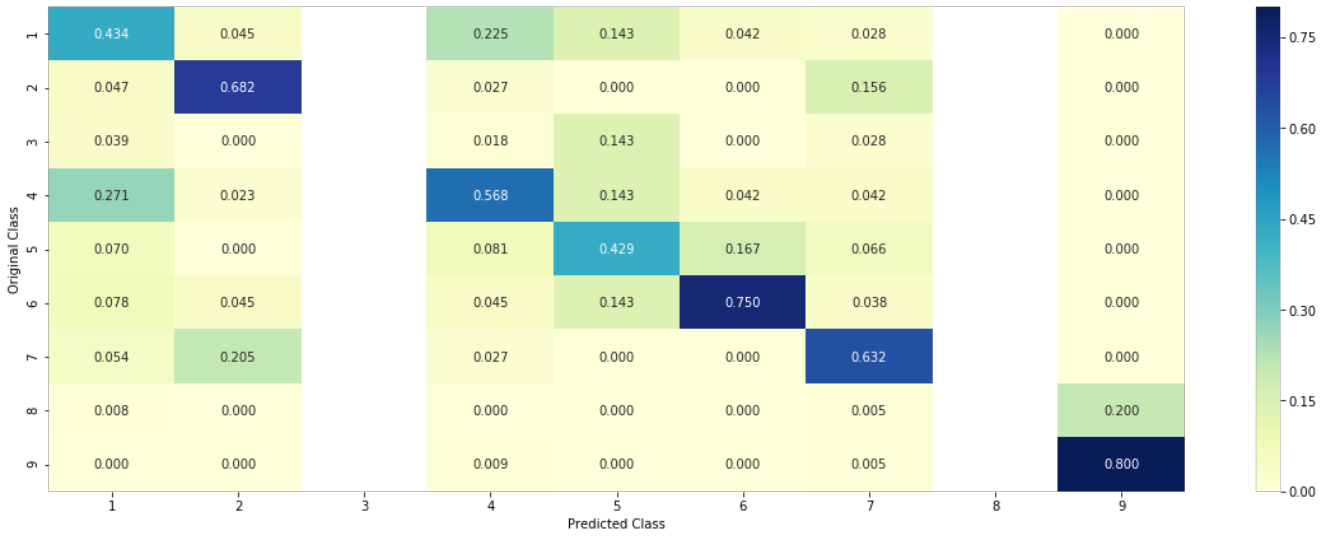
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

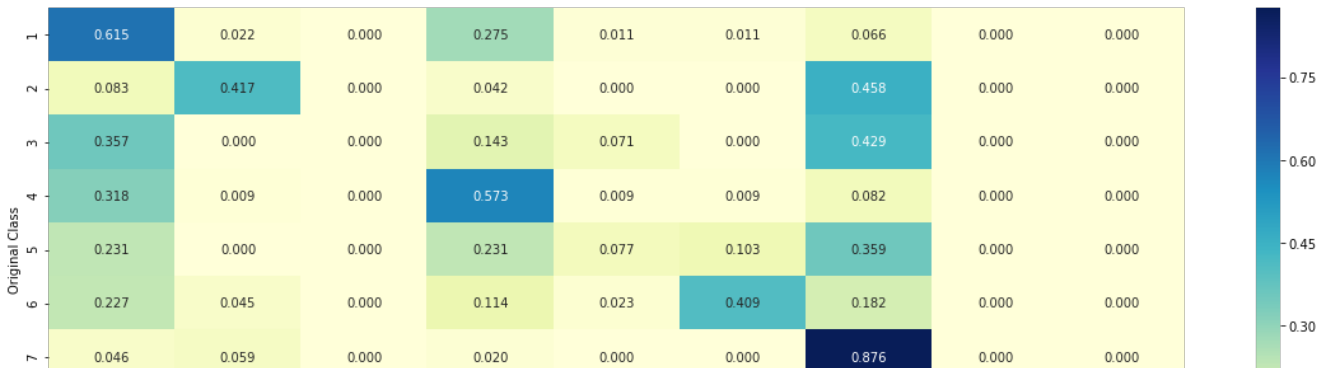
```
Log loss : 1.2436768923560293
Number of mis-classified points : 0.42105263157894735
-------------------- Confusion matrix --------------------
```

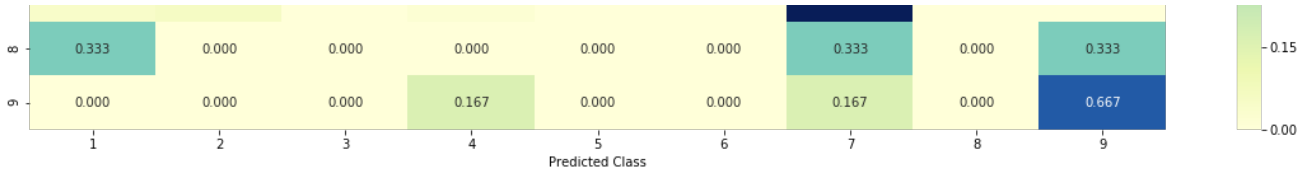



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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






### 4.5.3. Feature Importance

#### 4.5.3.1. Correctly Classified point

In [110]:

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

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0576 0.0074 0.0084 0.0364 0.0949 0.7784 0.0137 0.0015 0.0017]]
Actual Class : 6
--------------------------------------------------
2 Text feature [function] present in test data point [True]
10 Text feature [missense] present in test data point [True]
11 Text feature [loss] present in test data point [True]
13 Text feature [brca1] present in test data point [True]
15 Text feature [treatment] present in test data point [True]
18 Text feature [pathogenic] present in test data point [True]
21 Text feature [variants] present in test data point [True]
22 Text feature [receptor] present in test data point [True]
24 Text feature [deleterious] present in test data point [True]
26 Text feature [functional] present in test data point [True]
27 Text feature [cell] present in test data point [True]
28 Text feature [growth] present in test data point [True]
30 Text feature [cells] present in test data point [True]
35 Text feature [protein] present in test data point [True]
37 Text feature [brca2] present in test data point [True]
38 Text feature [classified] present in test data point [True]
41 Text feature [treated] present in test data point [True]
43 Text feature [neutral] present in test data point [True]
46 Text feature [therapeutic] present in test data point [True]
55 Text feature [brca] present in test data point [True]
56 Text feature [predicted] present in test data point [True]
63 Text feature [ovarian] present in test data point [True]
67 Text feature [ring] present in test data point [True]
68 Text feature [variant] present in test data point [True]
70 Text feature [patients] present in test data point [True]
72 Text feature [dna] present in test data point [True]
73 Text feature [sensitivity] present in test data point [True]
74 Text feature [57] present in test data point [True]
77 Text feature [odds] present in test data point [True]
78 Text feature [conserved] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
83 Text feature [information] present in test data point [True]
85 Text feature [lines] present in test data point [True]
86 Text feature [expected] present in test data point [True]
92 Text feature [affected] present in test data point [True]
Out of the top  100  features  35 are present in query point
```

#### 4.5.3.2. Inorrectly Classified point

In [111]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2399 0.0077 0.0511 0.3246 0.0934 0.2592 0.0166 0.0025 0.0049]]
Actuall Class : 3
--------------------------------------------------
2 Text feature [function] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [suppressor] present in test data point [True]
10 Text feature [missense] present in test data point [True]
11 Text feature [loss] present in test data point [True]
12 Text feature [transforming] present in test data point [True]
13 Text feature [brca1] present in test data point [True]
16 Text feature [inhibitor] present in test data point [True]
18 Text feature [pathogenic] present in test data point [True]
19 Text feature [stability] present in test data point [True]
21 Text feature [variants] present in test data point [True]
22 Text feature [receptor] present in test data point [True]
24 Text feature [deleterious] present in test data point [True]
26 Text feature [functional] present in test data point [True]
27 Text feature [cell] present in test data point [True]
28 Text feature [growth] present in test data point [True]
30 Text feature [cells] present in test data point [True]
32 Text feature [yeast] present in test data point [True]
35 Text feature [protein] present in test data point [True]
36 Text feature [inhibited] present in test data point [True]
37 Text feature [brca2] present in test data point [True]
38 Text feature [classified] present in test data point [True]
40 Text feature [expression] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [functions] present in test data point [True]
43 Text feature [neutral] present in test data point [True]
45 Text feature [repair] present in test data point [True]
47 Text feature [proteins] present in test data point [True]
48 Text feature [defective] present in test data point [True]
53 Text feature [resistance] present in test data point [True]
56 Text feature [predicted] present in test data point [True]
62 Text feature [response] present in test data point [True]
63 Text feature [ovarian] present in test data point [True]
67 Text feature [ring] present in test data point [True]
68 Text feature [variant] present in test data point [True]
70 Text feature [patients] present in test data point [True]
71 Text feature [damage] present in test data point [True]
72 Text feature [dna] present in test data point [True]
73 Text feature [sensitivity] present in test data point [True]
75 Text feature [expressing] present in test data point [True]
77 Text feature [odds] present in test data point [True]
78 Text feature [conserved] present in test data point [True]
79 Text feature [ligand] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
83 Text feature [information] present in test data point [True]
85 Text feature [lines] present in test data point [True]
86 Text feature [expected] present in test data point [True]
87 Text feature [activity] present in test data point [True]
88 Text feature [inhibition] present in test data point [True]
90 Text feature [assays] present in test data point [True]
92 Text feature [affected] present in test data point [True]
93 Text feature [assay] present in test data point [True]
96 Text feature [downstream] present in test data point [True]
Out of the top  100  features  53 are present in query point
```

### 4.5.3. Hyper paramter tuning (With Response Coding)

In [112]:

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

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

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

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


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

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
_train, 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 : 2.2305420892989427
for n_estimators = 10 and max depth =  3
Log Loss : 1.621725997666496
for n_estimators = 10 and max depth =  5
Log Loss : 1.6083580231874626
for n_estimators = 10 and max depth =  10
Log Loss : 1.9207665243865544
for n_estimators = 50 and max depth =  2
Log Loss : 1.9088645345411264
for n_estimators = 50 and max depth =  3
Log Loss : 1.6195077446371076
for n_estimators = 50 and max depth =  5
Log Loss : 1.5623230581508691
for n_estimators = 50 and max depth =  10
Log Loss : 1.8086318318209833
for n_estimators = 100 and max depth =  2
Log Loss : 1.741144063129107
for n_estimators = 100 and max depth =  3
Log Loss : 1.7218051744473226
for n_estimators = 100 and max depth =  5
Log Loss : 1.4531973318107547
for n_estimators = 100 and max depth =  10
Log Loss : 1.792142496254684
for n_estimators = 200 and max depth =  2
Log Loss : 1.7774868533595367
for n_estimators = 200 and max depth =  3
Log Loss : 1.7378974157364324
for n_estimators = 200 and max depth =  5
Log Loss : 1.5644453539121541
for n_estimators = 200 and max depth =  10
Log Loss : 1.8008705195714558
for n_estimators = 500 and max depth =  2
Log Loss : 1.8736465462619376
for n_estimators = 500 and max depth =  3
Log Loss : 1.7964978510113605
for n_estimators = 500 and max depth =  5
Log Loss : 1.5864556562226673
for n_estimators = 500 and max depth =  10
Log Loss : 1.820409047031725
for n_estimators = 1000 and max depth =  2
Log Loss : 1.840246160115029
for n_estimators = 1000 and max depth =  3
Log Loss : 1.789861123622554
for n_estimators = 1000 and max depth =  5
Log Loss : 1.5861162931513006
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8502620088518582
For values of best alpha =  100 The train log loss is: 0.056014921549048365
For values of best alpha =  100 The cross validation log loss is: 1.4531973318107547
For values of best alpha =  100 The test log loss is: 1.3814657331935223
```

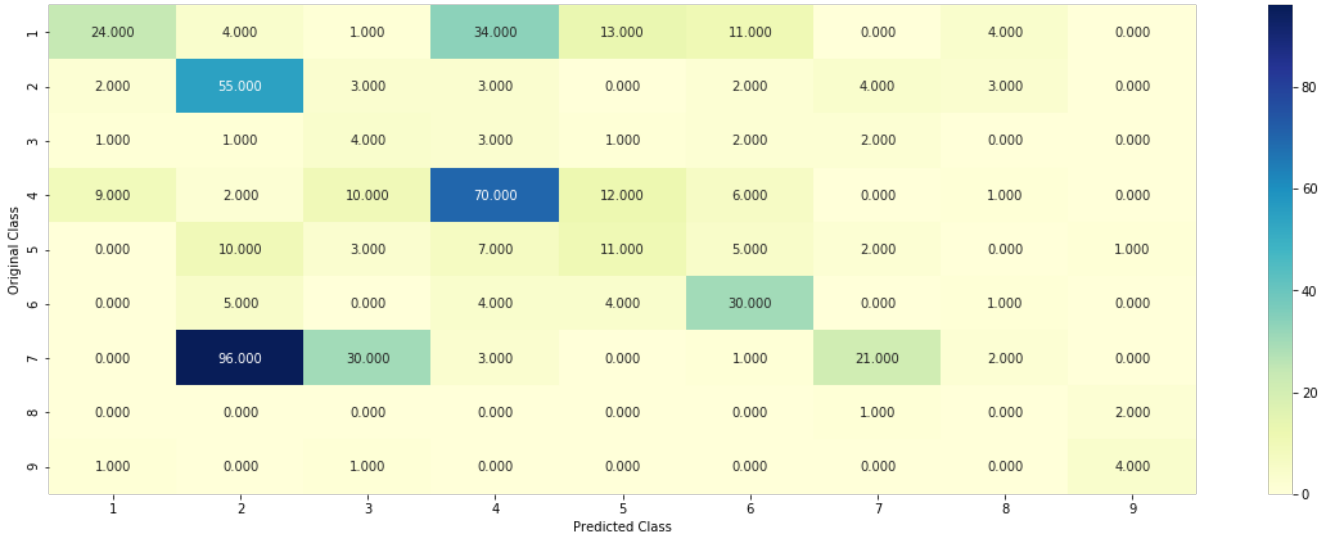
### 4.5.4. Testing model with best hyper parameters (Response Coding)

In [113]:

```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
```

```
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-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)
```

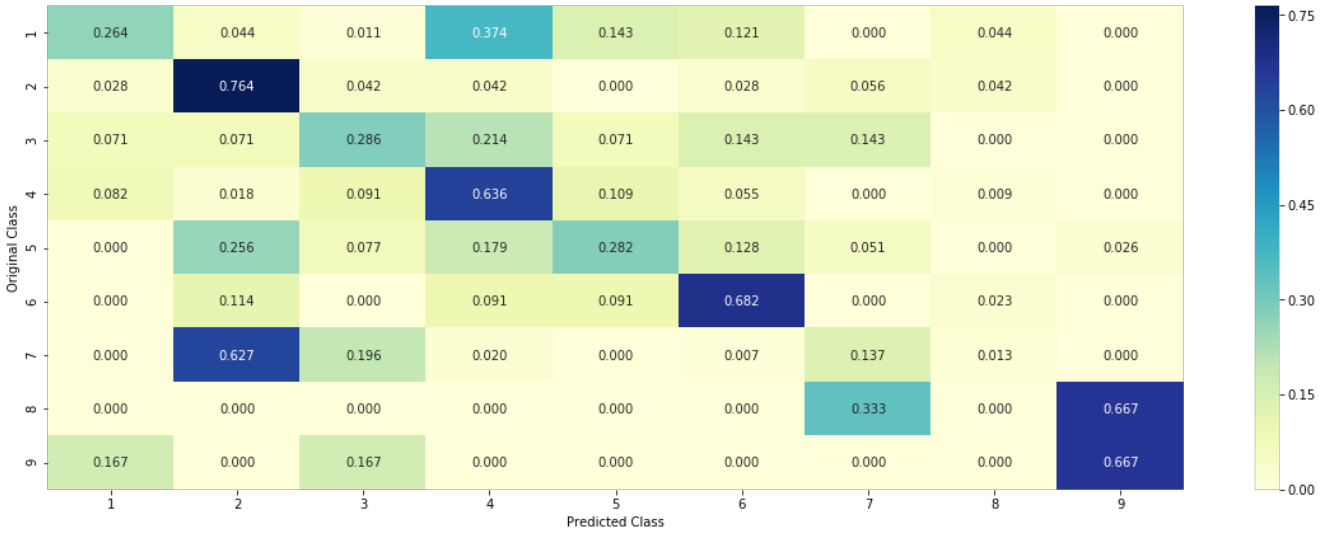
```
Log loss : 1.4531973318107547
Number of mis-classified points : 0.5883458646616542
-------------------- Confusion matrix --------------------
```




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


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

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




## 4.5.5. Feature Importance

#### 4.5.5.1. Correctly Classified point

In [114]:

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


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: [[0.0169 0.0033 0.0163 0.0176 0.2728 0.6629 0.0015 0.0049 0.0039]]
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
Gene is important feature
Text 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
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
```

#### 4.5.5.2. Incorrectly Classified point

In [115]:

```
test_point_index = 50
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 : 5
Predicted Class Probabilities: [[0.0312 0.0048 0.1955 0.0466 0.3825 0.3265 0.0021 0.006  0.0048]]
Actual Class : 3
--------------------------------------------------
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
Gene is important feature
Text 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
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
```

## 4.7 Stack the models

### 4.7.1 testing with hyper parameter tuning

In [116]:

```
# 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_i
ter=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: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#-------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

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


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

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

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

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


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

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


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
```

```
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=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, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

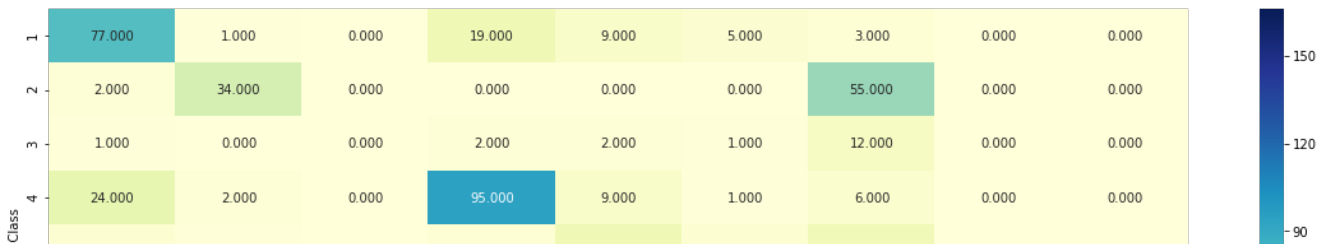
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.81
Naive Bayes : Log Loss: 1.26
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.515
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.207
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.456
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.961
```

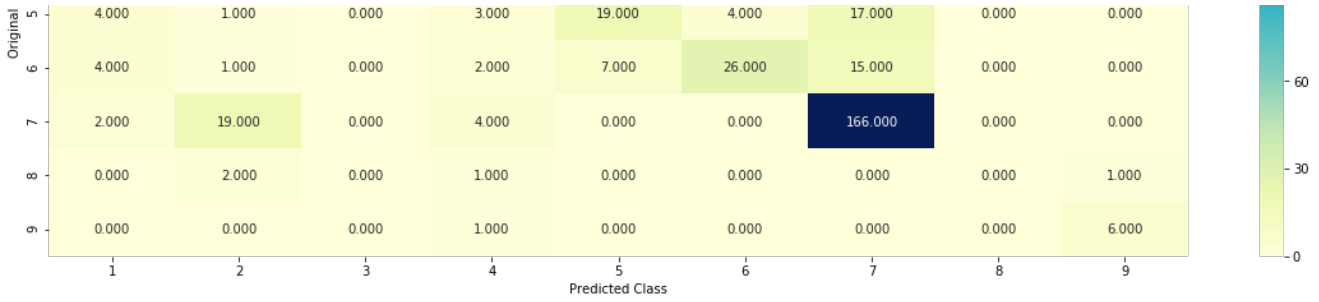
### 4.7.2 testing the model with the best hyper parameters

In [117]:

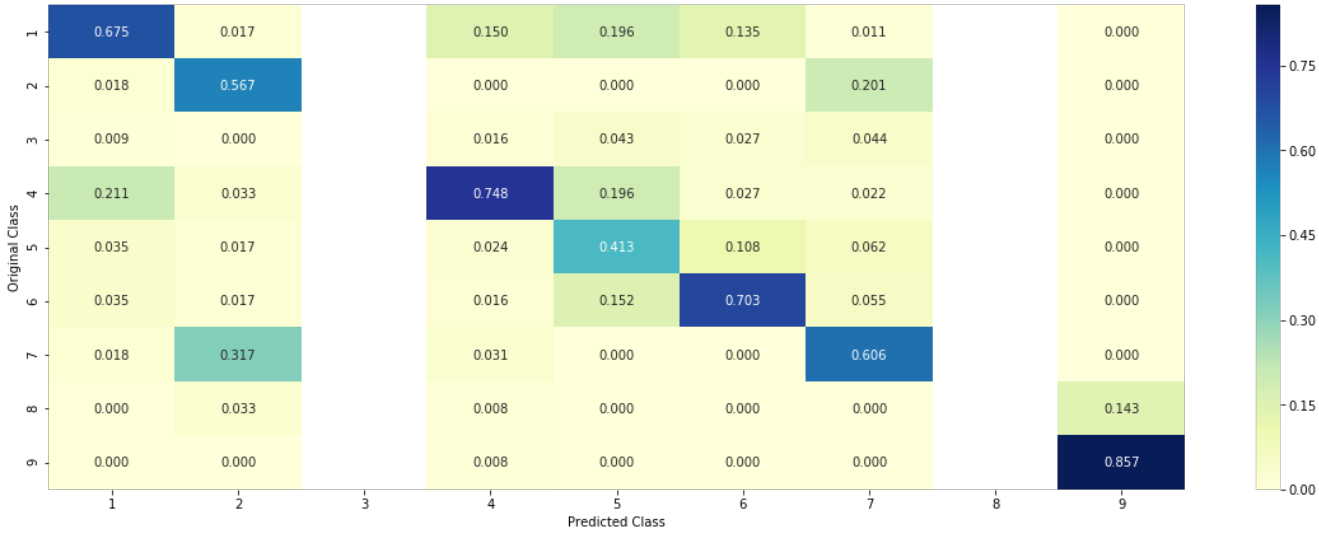
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5263055382187135
Log loss (CV) on the stacking classifier : 1.2069544665863272
Log loss (test) on the stacking classifier : 1.1324603115425407
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```

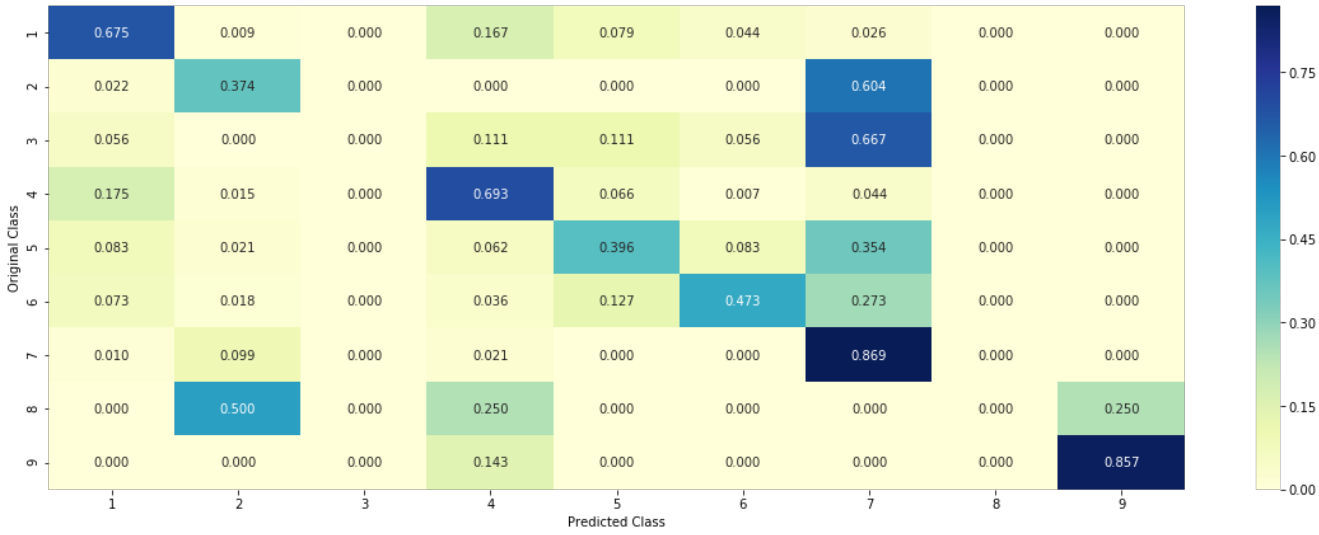





---------- Precision matrix (Columm Sum=1) ----------




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



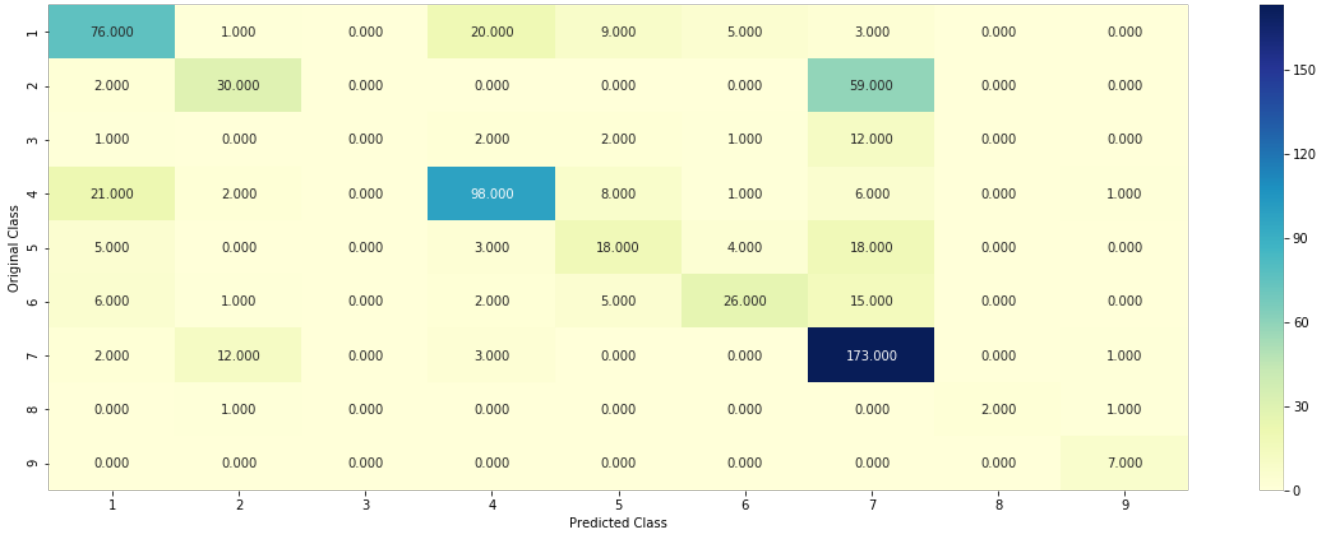

### 4.7.3 Maximum Voting classifier

In [118]:

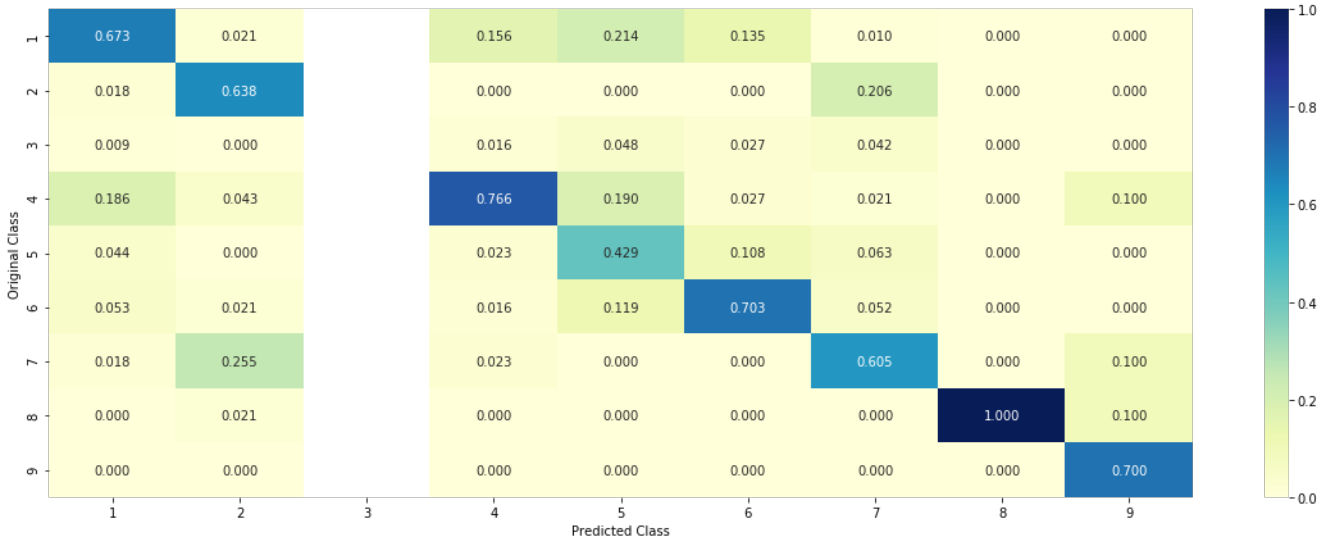
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log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

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

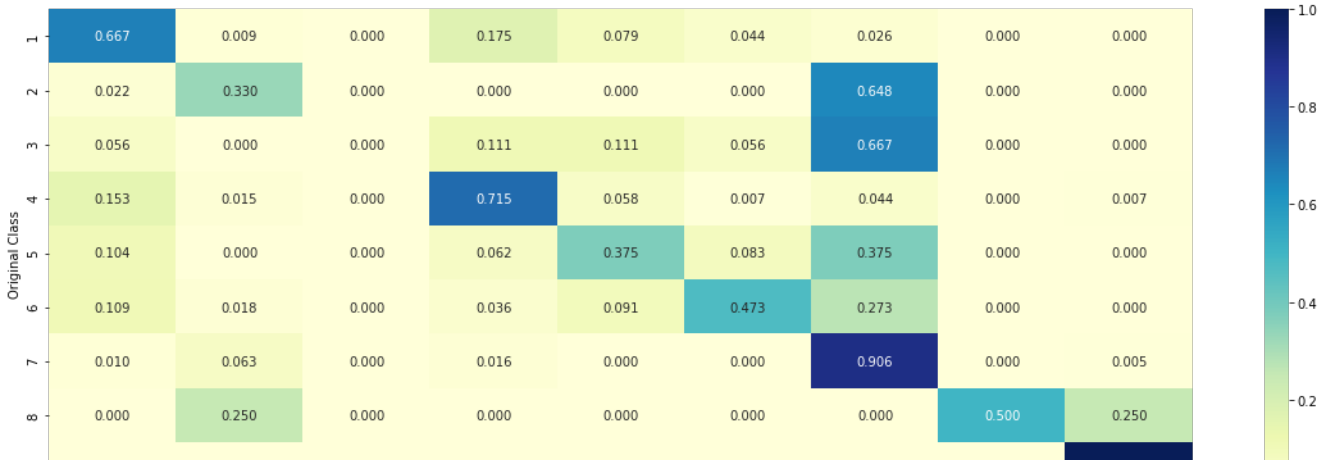
```
Log loss (train) on the VotingClassifier : 0.8205211369791904
Log loss (CV) on the VotingClassifier : 1.2358592154832424
Log loss (test) on the VotingClassifier : 1.1578788529527342
Number of missclassified point : 0.3533834586466165
-------------------- Confusion matrix --------------------
```

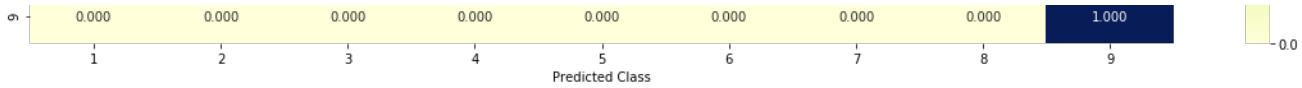



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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






# 5. Assignments

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

# Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [126]:

```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variation'].values;cv_variatio
n=cv_df['Variation'].values
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv_df['Gene'].values
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv_df['TEXT'].values

from sklearn.feature_extraction.text import CountVectorizer
encode=CountVectorizer(ngram_range=(1, 2))
train_variation=encode.fit_transform(train_variation);test_variation=encode.transform(test_variatic
n);cv_variation=encode.transform(cv_variation)
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);cv_gene=encode.t
ransform(cv_gene)
#train_text=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);cv_gene=encode.t
form(cv_gene)
encode=CountVectorizer(min_df=10,ngram_range=(1,2))
train_variation=normalize(train_variation,axis=0);test_variation=normalize(test_variation,axis=0);
cv_variation=normalize(cv_variation,axis=0);
train_gene=normalize(train_gene,axis=0);test_gene=normalize(test_gene,axis=0);cv_gene=normalize(cv
_gene,axis=0);
print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_text);cv_text=encode.t
ransform(cv_text)
train_text=normalize(train_text,axis=0);test_text=normalize(test_text,axis=0);cv_text=normalize(cv
_text,axis=0)
print(train_text.shape)
```

```
(2124, 239)
  (0, 62) 0.10660035817780521
(2124, 2054)
  (0, 712) 1.0
(2124, 230843)
```

In [127]:

```
from scipy.sparse import hstack
print(train_variation.shape,train_gene.shape,train_text.shape)
train_data=hstack([train_variation,train_gene,train_text]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text]).tocsr()
test_data=hstack([test_variation,test_gene,test_text]).tocsr()
print(train_data.shape,cv_data.shape,test_data.shape)
```

```
(2124, 2054) (2124, 239) (2124, 230843)
(2124, 233136) (532, 233136) (665, 233136)
```

# Logistic Regression with class balance

In [122]:

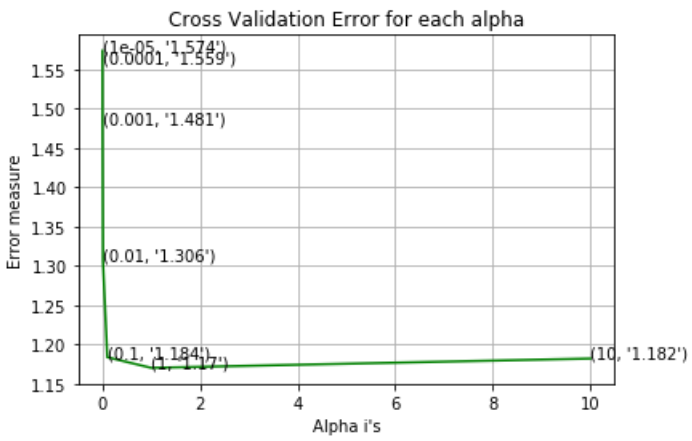
```
cv_scores=[]
alpha=[10 ** x for x in range(-5, 2)]
for c in alpha:
    print("for alpha =", c)
    lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_jobs=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_op=clf.predict_proba(cv_data)
    print(c,log_loss(y_cv, cv_op))
    cv_scores.append(log_loss(y_cv, cv_op))
```

```
for alpha = 1e-05
1e-05 1.5736072572156412
for alpha = 0.0001
0.0001 1.559368915140178
for alpha = 0.001
0.001 1.4808668441087112
for alpha = 0.01
0.01 1.3064123290187657
for alpha = 0.1
0.1 1.1836247822317656
for alpha = 1
1 1.1700946566667552
for alpha = 10
10 1.1819214906327575
```

In [123]:

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

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10] [1.5736072572156412, 1.559368915140178,
1.4808668441087112, 1.3064123290187657, 1.1836247822317656, 1.1700946566667552,
1.1819214906327575]
7 7
<class 'list'>
```




In [124]:

```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1)
```

```
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
print("Log Loss value for the test data is(10) ",log_loss(y_test, test_op))
```

```
Log Loss value for the test data is(10)  1.0810087459939697
```

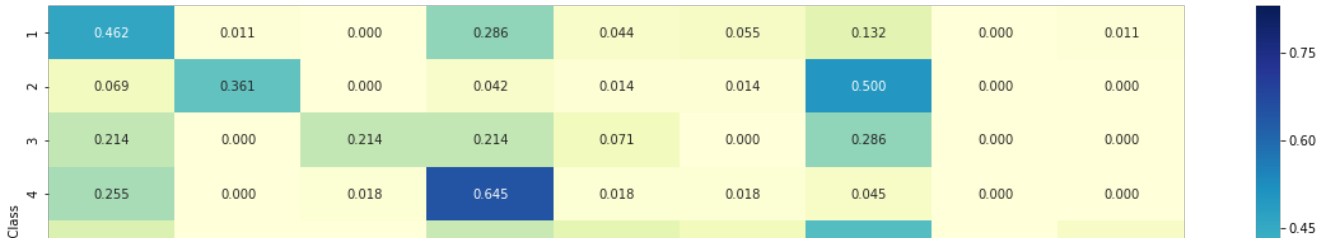
In [125]:

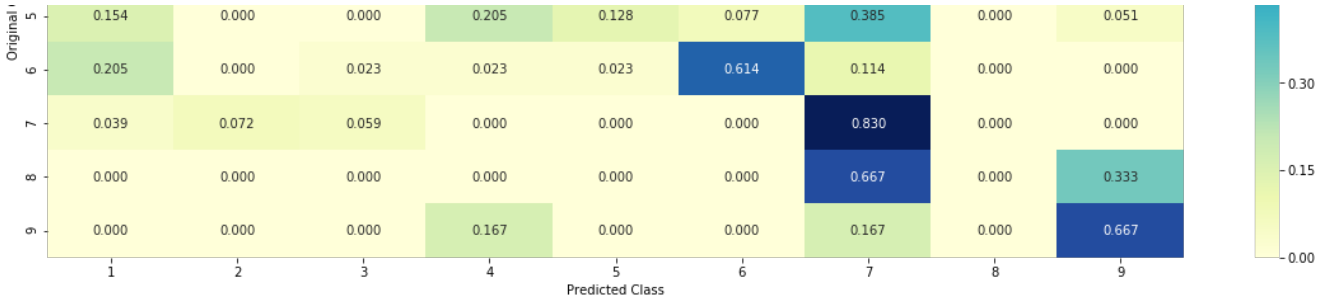
```
lr = LogisticRegression(random_state=0, C=10,class_weight='balanced',n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
predict_and_plot_confusion_matrix(train_data, y_train, cv_data,y_cv, clf)
```

```
Log loss : 1.1882869498296549
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```


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


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






# Logistic Regression without class balance

In [128]:

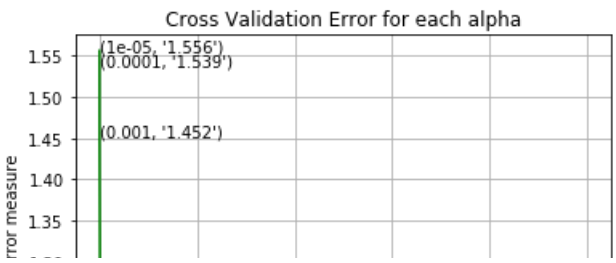
```
cv_scores=[]
alpha=[10 ** x for x in range(-5, 2)]
for c in alpha:
    print("for alpha =", c)
    lr = LogisticRegression(random_state=0, C=c,n_jobs=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_op=clf.predict_proba(cv_data)
    print(c,log_loss(y_cv, cv_op))
    cv_scores.append(log_loss(y_cv, cv_op))
```

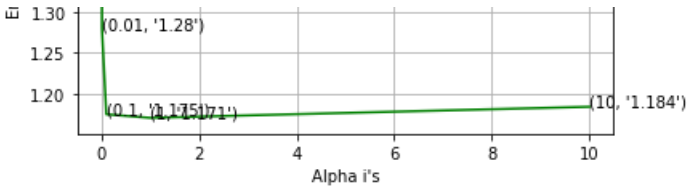
```
for alpha = 1e-05
1e-05 1.5561956833030874
for alpha = 0.0001
0.0001 1.5387345245637378
for alpha = 0.001
0.001 1.4521353735411526
for alpha = 0.01
0.01 1.2795339418307579
for alpha = 0.1
0.1 1.1752478764884722
for alpha = 1
1 1.170914939761208
for alpha = 10
10 1.1843738807350708
```

In [129]:

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

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10] [1.5561956833030874, 1.5387345245637378,
1.4521353735411526, 1.2795339418307579, 1.1752478764884722, 1.170914939761208, 1.1843738807350708]
7 7
<class 'list'>
```






In [130]:

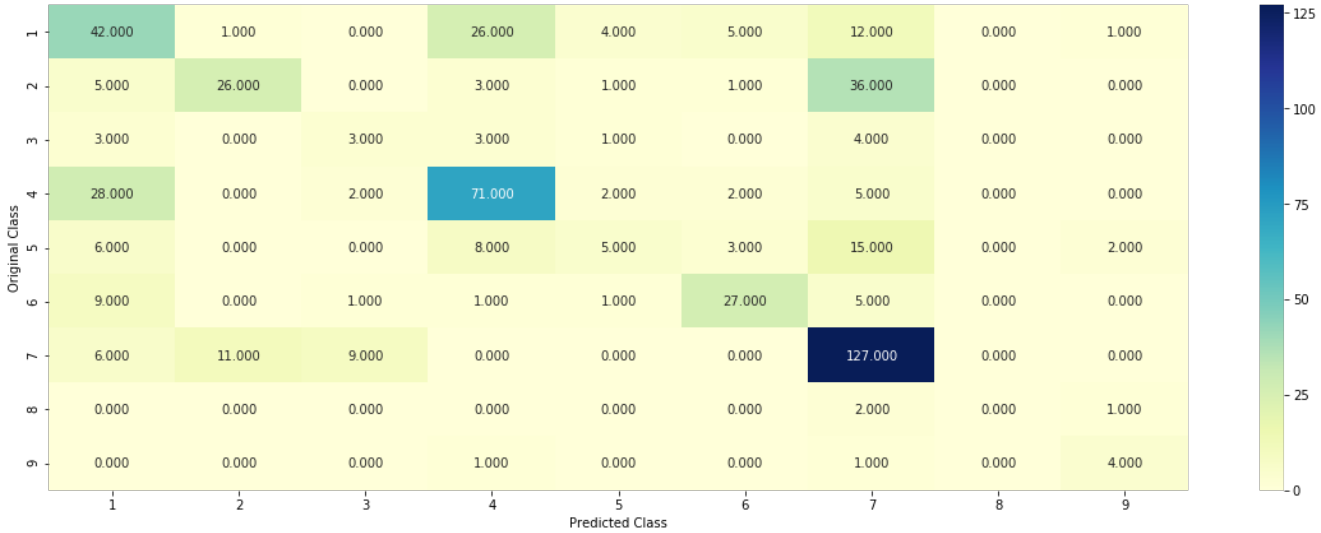
```
lr = LogisticRegression(random_state=0, C=10,n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
print("Log Loss value for the test data is(10) ",log_loss(y_test, test_op))
```

```
Log Loss value for the test data is(10)  1.0816900529487397
```

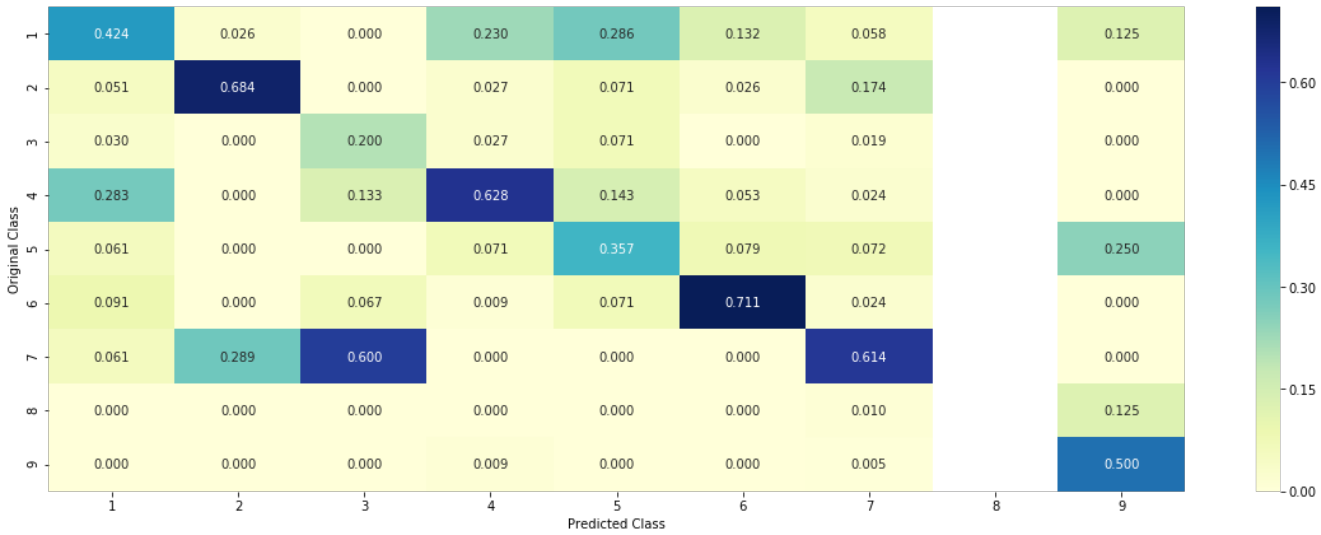
In [131]:

```
lr = LogisticRegression(random_state=0, C=10,n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
predict_and_plot_confusion_matrix(train_data, y_train, cv_data,y_cv, clf)
```

```
Log loss : 1.1902212439103876
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```




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



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


# feature engineering techniques to reduce log loss

In [132]:

```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)

x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train, test_size=0.2)
```

In [133]:

```
def get_gv_fea_dict(alpha, feature, df):
    value_count = x_train[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    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])
    return gv_fea
```

In [134]:

```
alpha = 1

# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_train))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_test))
```

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

In [135]:

```
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

## Variation Feature

In [136]:

```
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_train))

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_test))

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_cv))
```

In [137]:

```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'])
```

## Text Feature

In [138]:

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


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

In [139]:

```
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) 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 : 126190
```

In [140]:

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

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


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

In [141]:

```
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

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

In [142]:

```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['TEXT'])
```

# Features after feature engineering

In [143]:

```
gene_variation = []

for gene in data['Gene'].values:
    gene_variation.append(gene)

for variation in data['Variation'].values:
    gene_variation.append(variation)
```

In [144]:

```
tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
```

```
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
cv_text = tfidfVectorizer.transform(x_cv['TEXT'])
```

## Stack above three features

In [145]:

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

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(x_train['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(x_test['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(x_cv['Class']))


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

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

In [146]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 129383)
(number of data points * number of features) in test data =  (665, 129383)
(number of data points * number of features) in cross validation data = (532, 129383)
```

In [147]:

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

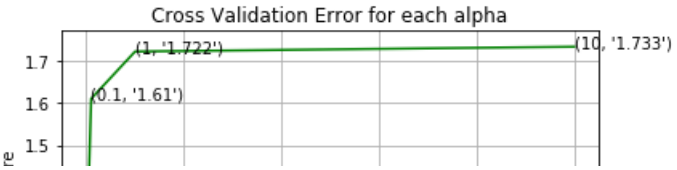
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
```

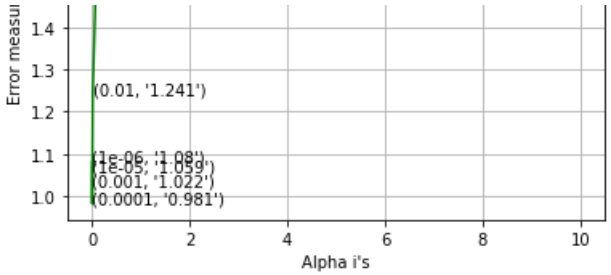
(number of data points * number of features) in cross validation data = (532, 27)

In [148]:

```
alpha = [10 ** x for x in range(-6, 2)]
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=41
)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.0796035140981501
for alpha = 1e-05
Log Loss : 1.0594540610989416
for alpha = 0.0001
Log Loss : 0.9811117885672768
for alpha = 0.001
Log Loss : 1.022242671477516
for alpha = 0.01
Log Loss : 1.2411181728826903
for alpha = 0.1
Log Loss : 1.6096991571447559
for alpha = 1
Log Loss : 1.7216164571557
for alpha = 10
Log Loss : 1.7330139160076867
```



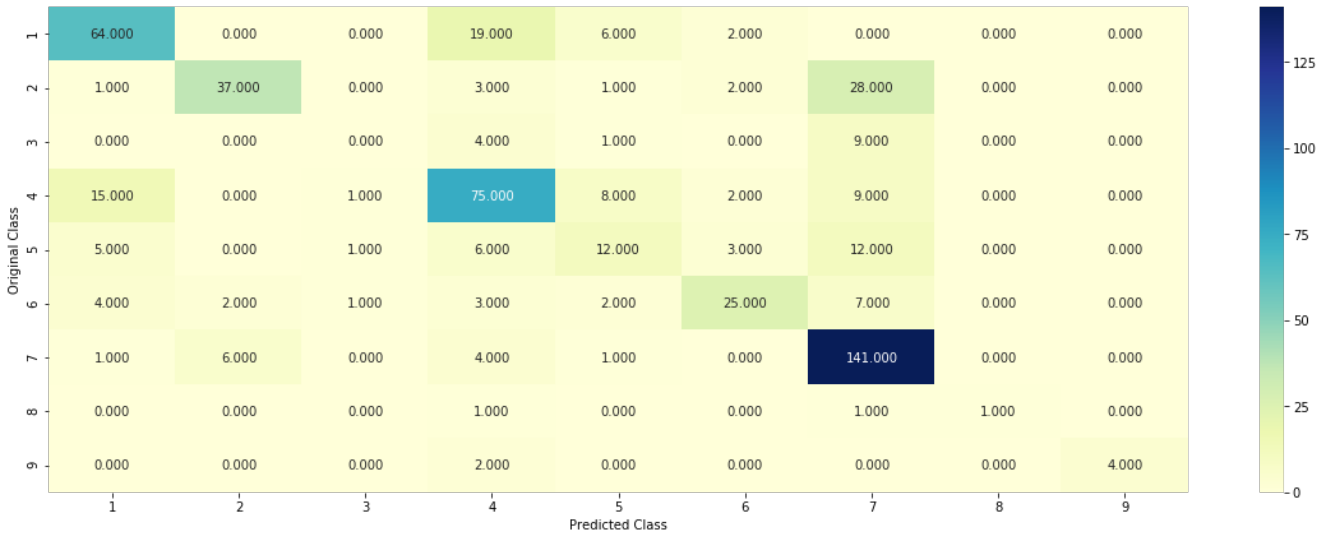



```
For values of best alpha =  0.0001 The train log loss is: 0.4431883563196341
For values of best alpha =  0.0001 The cross validation log loss is: 0.9740588000074776
For values of best alpha =  0.0001 The test log loss is: 1.0128919303025137
```

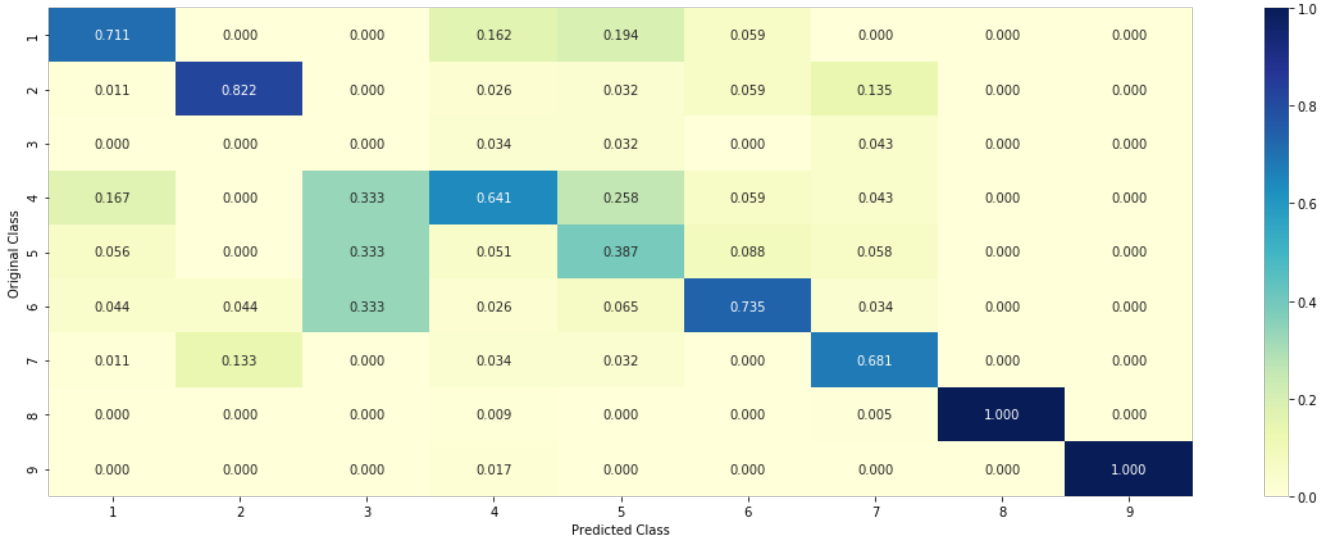
In [149]:

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

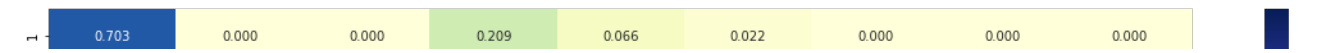
```
Log loss : 0.9789413907935669
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```

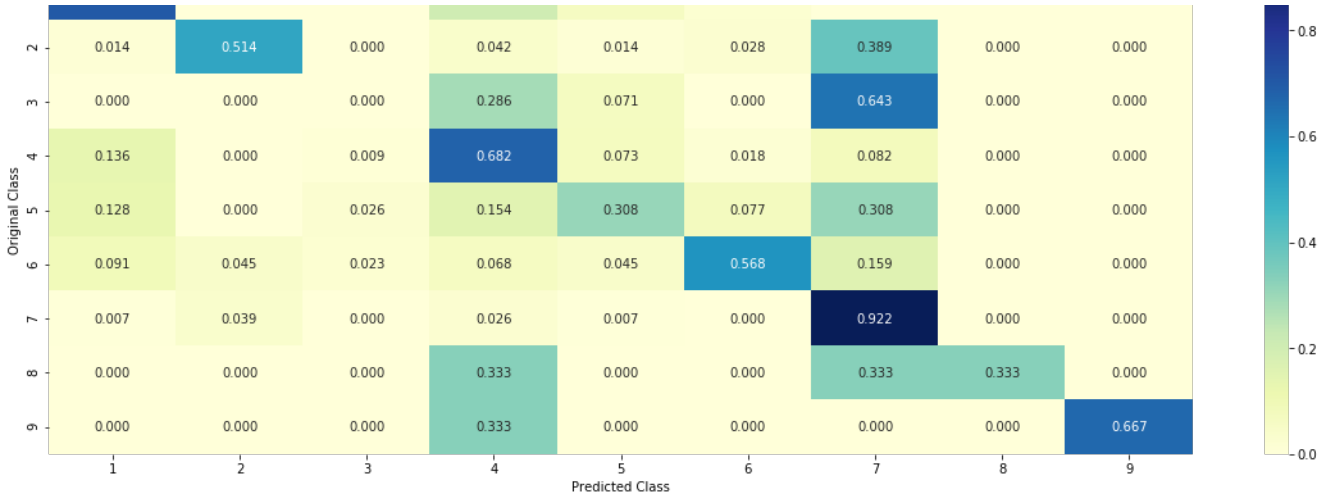



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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






# Final Results

In [2]:

```
from prettytable import PrettyTable

# Names of models
model=['Naive Bayes  ','KNN','Logistic Regression With Class balancing  ','Logistic Regression
Without Class balancing','Linear SVM  ','Random Forest Classifier With One hot Encoding','Random F
orest Classifier With Response Coding','Stack Models:LR+NB+SVM','Maximum Voting classifier','Count
Vectorizer Features, including both unigrams and bigrams with class balance',"CountVectorizer Feat
ures, including both unigrams and bigrams without class balance ",'after feature engineering']



train =[0.50,0.635,0.433,0.42,0.45, 0.83,0.05,0.53,0.83,1.08,1.08,0.44]
cv=[1.25,1.16,1.07,1.10,1.13,1.24,1.45,1.20,1.23,1.07,1.07,0.97]
test = [1.15,1.02,0.98,0.99,1.03,1.15,1.38,1.13,1.15,1.06,1.06,1.01]
mp=[41,41,37,37,36,42,58,36,35,42,42,32]
numbering=[1,2,3,4,5,6,7,8,9,10,11,12]
# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("model",model)
ptable.add_column("train",train)
ptable.add_column("cv",cv)
ptable.add_column("test",test)
ptable.add_column("% Misclassified Points",mp)

# Printing the Table
print(ptable)
```

```
+-------+-------------------------------------------------------------------------------------+---
+------+------+-------------------------+
| S.NO. |                                    model                                            | tr
|  cv  | test | % Misclassified Points |
+-------+-------------------------------------------------------------------------------------+---
+------+------+-------------------------+
|   1   |                                  Naive Bayes                                        |  0
| 1.25 | 1.15 |           41           |
|   2   |                                     KNN                                             | 0.
| 1.16 | 1.02 |           41           |
|   3   |                      Logistic Regression With Class balancing                       |
0.433 | 1.07 | 0.98 |           37           |
|   4   |                     Logistic Regression Without Class balancing                     |
0.42 | 1.1  | 0.99 |           37           |
|   5   |                                   Linear SVM                                        |  0
| 1.13 | 1.03 |           36           |
|   6   |                 Random Forest Classifier With One hot Encoding                      |  0
83 | 1.24 | 1.15 |           42           |
|   7   |                 Random Forest Classifier With Response Coding                       |  0
05 | 1.45 | 1.38 |           58           |
```

```
|   8   |                                Stack Models:LR+NB+SVM                                 |
0.53 | 1.2  | 1.13 |           36           |
|   9   |                              Maximum Voting classifier                                |  0
3 | 1.23 | 1.15 |           35           |
|   10  |   CountVectorizer Features, including both unigrams and bigrams with class balance   |  1
.08 | 1.07 | 1.06 |           42           |
|   11  | CountVectorizer Features, including both unigrams and bigrams without class balance  |
1.08 | 1.07 | 1.06 |           42           |
|   12  |                              after feature engineering                                |  0
4 | 0.97 | 1.01 |           32           |
+-------+-------------------------------------------------------------------------------------+---
+------+------+-----------------------+
```

## Conclusion

According to our ploblem statment: We need to predict the probability of each data-point belonging to each of the nine classes. i.e Classify the given genetic variations/mutations based on evidence from text-based clinical literature

Lets Start -> As we know we have dataset which contains ID,Gene,Variation,Test,Class and we have given class lables i.e 1-9 and genetic mutation has been classified on this.

So lets start with Exploratory Data Analysis but before that we will do some Preprocessing of text so that we will able to rectify that is there any row which has null value and after doing this we will split our whole data into Train,Cv and test and then we will do some EDA on these we will able to visialuze the distribution of data or the class in the train test and cv.

And after that lets apply some random models to that we will able to recify the worst performance of the models, and then we will start working on the models

Now after getting performance of the random models, lets do some featurization on our data set as we know we have dataset which contains ID,Gene,Variation,Test,Class from which we have Gene, variance and test as a features. So lets start with univariate Analysis on each features one by one and try to get the performance of each. Here we will work with two type of featurization i.e Response coding One hot coding Note : 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. And at the same time we will try to get how good is this feature in predicting y_i, and Is the feature stable across all the data sets (Test, Train, Cross validation), the distribution if the features and lot more.

After doing all above now we will Start with our Machine learning models but before that we will combine our all featurized features into one i.e Stacking the three types of features and then will apply different machine learning models on it and try to get the best out of it by apply hyperparameter tuning on it each of the models.

The most imp this is to visualise the confusion matrix, precison and recall using heat map which help us to visualize the performance of our models so that we will able to pick best out of these.

What ever we have implemented in pervious notebook we will do the same thing just with minor changes in order to see where we can impove the model performance or not so and to do that what we will apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams insteed of TFIDF what we have applied in pervious notebook and as we know when we will use CountVectorizer Features, including both unigrams and bigrams we know we will have the data with huge dim so instead of apply this on all the models we will only apply this on Logistic regression because as we know logisti reg works well on high dim data.

As we have seen in last two notebook we have worked with different featurization in order to improve our model performance. Now in this we will try some of the feature engineering techniques to reduce the log-loss to a value less than 1.0.

And after trying some feature engineering we got our test log loss less than 1. In this we are only working with Logistic reg because as we know our data is high dim and we know with high dim data logistic reg works well and here we got the performance.