

# 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=qxXRKVompl8>

### 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 contains 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 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 Learning 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 probabilities => 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 [5]:

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

from mlxtend.classifier import StackingClassifier

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

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

In [6]:

```
data = pd.read_csv('Cancerdata/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[6]:

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

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

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

### 3.1.2. Reading Text Data

In [8]:

```
# note the separator in this file
data_text = pd.read_csv("Cancerdata/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[8]:

	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 [9]:

```
# 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 [10]:

```

#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 : 44.11321568794881 seconds

```

In [11]:

```

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

```

Out[11]:

	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 [12]:

```

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

```

Out[12]:

	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 [13]:

```

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

```

In [15]:

```

result[result['ID']==1277]

```

Out[15]:

	ID	Gene	Variation	Class	TEXT
1277	1277	ARID5B	Truncating Mutations	1	ARID5B Truncating Mutations

### 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 [16]:

```
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 variable '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 variable '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 [17]:

```
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 [0]:

```
# 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 y_i in train data')
plt.grid()
plt.show()

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

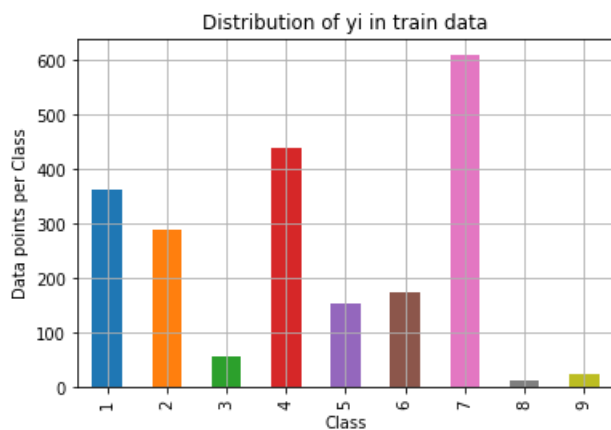
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of y_i in test data')
plt.grid()
```

```
plt.show()

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

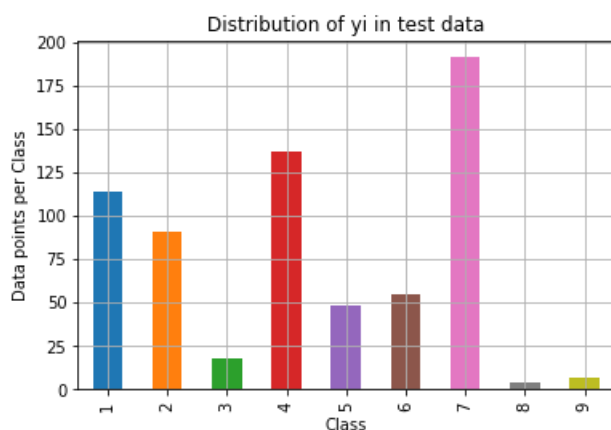
print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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



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

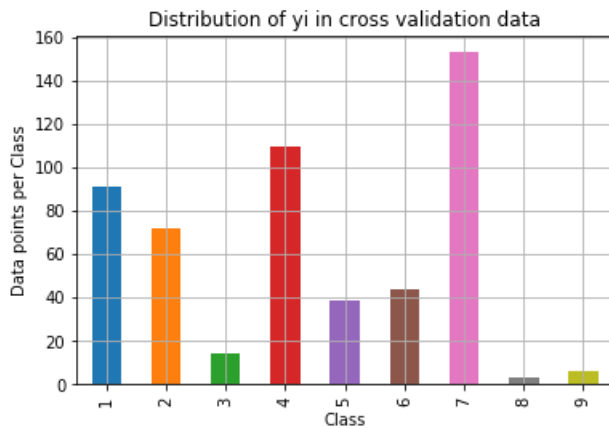
---



Number of data points in class 7 : 191 ( 28.722 %)  
Number of data points in class 4 : 137 ( 20.602 %)  
Number of data points in class 1 : 114 ( 17.143 %)  
Number of data points in class 2 : 91 ( 13.684 %)

Number of data points in class 6 : 55 ( 8.271 %)  
 Number of data points in class 5 : 48 ( 7.218 %)  
 Number of data points in class 3 : 18 ( 2.707 %)  
 Number of data points in class 9 : 7 ( 1.053 %)  
 Number of data points in class 8 : 4 ( 0.602 %)

---



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

## 3.2 Prediction using a 'Random' Model

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

In [18]:

```
# 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
    #divide 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 corresponds to columns and axis=1 corresponds to rows in two dimensional
    array
    # C.sum(axis = 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))
    #divide 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 corresponds to columns and axis=1 corresponds to rows in two dimensional
    array
    # C.sum(axis = 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 (Column 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 [19]:

```

# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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))

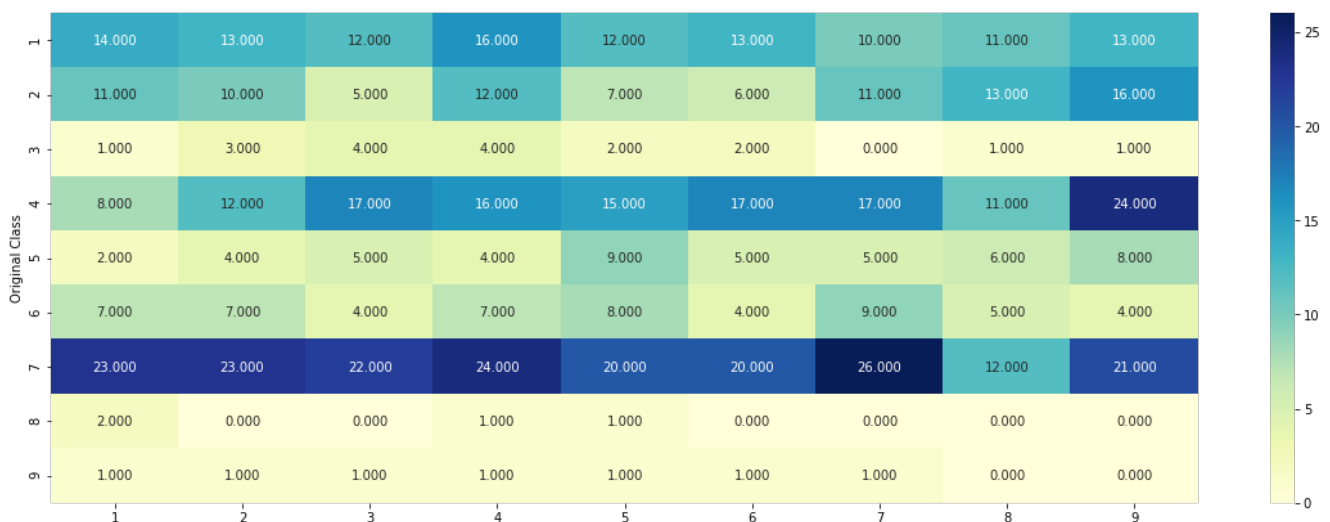
predicted_y = np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

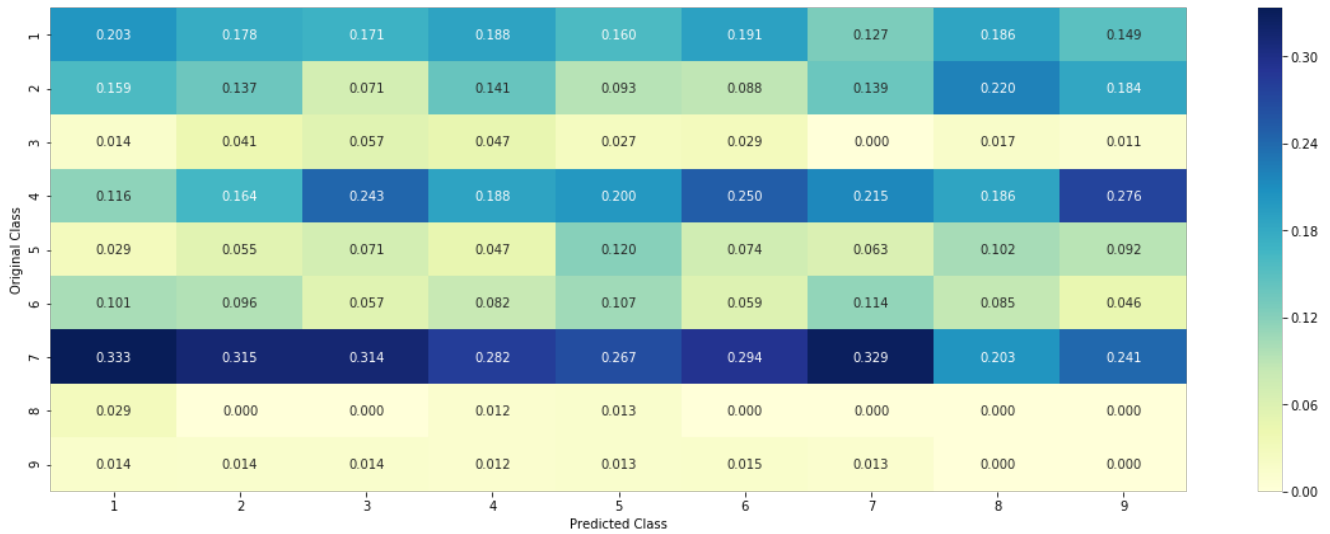
Log loss on Cross Validation Data using Random Model 2.511626449508402

Log loss on Test Data using Random Model 2.491750656286057

----- Confusion matrix -----



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



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



### 3.3 Univariate Analysis

In [20]:

```
# 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 occurrences of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha / number
of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# -----

# get_gv_fea_dict: Get Gene variation Feature Dict
def get_gv_fea_dict(alpha, feature, df):
```

```

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

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

# denominator will contain the number of time that particular feature occurred in whole data
for i, denominator in value_count.items():
    # vec will contain (p(yi=1/Gi) probability of gene/variation belongs to particular class
    # vec is 9 dimensional 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      MIR          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 occurred 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, 0.13636363636363636,
    5, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653061
    5, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.0561224489
    79591837],
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818181
    81818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878
    782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.0606060606
    6060608],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031446
    55, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081
    761006289],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284768211920529
    5, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556

```

```

0.00021000021012, 0.00021000021012, 0.00021000021012, 0.00021000021012, 0.000210000
2913912],
# 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.07333333333333333
34, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.06666666666666666, 0.066666666
66666666],
# ...
# }
gv_dict = get_gv_fea_dict(alpha, feature, df)
# value_count is similar in get_gv_fea_dict
value_count = train_df[feature].value_counts()

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

```

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

- $(\text{numerator} + 10 \cdot \alpha) / (\text{denominator} + 90 \cdot \alpha)$

### 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 [21]:

```

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

```

Number of Unique Genes : 238

```

BRCA1      164
TP53       104
EGFR        92
BRCA2       83
PTEN        82
KIT         65
BRAF        55
ERBB2       48
PIK3CA      42
ALK         39

```

Name: Gene, dtype: int64

In [22]:

```

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

```

Ans: There are 238 different categories of genes in the train data, and they are distributed as follows

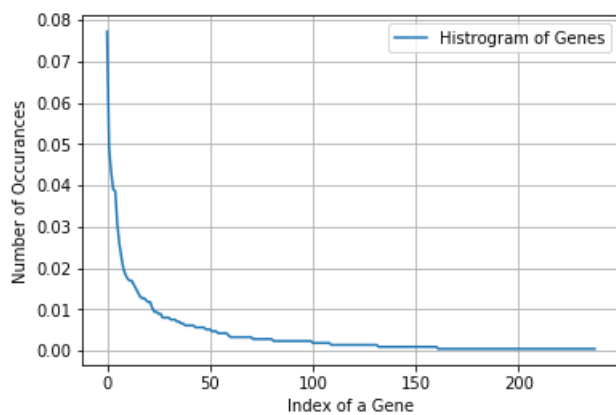
In [23]:

```

s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()

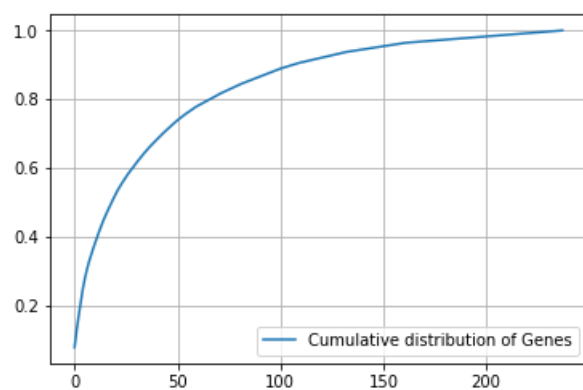
```

```
plt.legend()
plt.grid()
plt.show()
```



In [24]:

```
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.appliedaia.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 [25]:

```
#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 [27]:

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

train\_gene\_feature\_responseCoding is converted feature using response coding method. The shape of gene feature: (2124, 9)

In [96]:

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

In [97]:

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

Out[97]:

```
3307    RUNX1
180      EGFR
1734    MSH2
2252    PTEN
2182    PTEN
Name: Gene, dtype: object
```

In [98]:

```
gene_vectorizer.get_feature_names()
```

Out[98]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid2',
 'asx12',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
```

'cdk4',  
'cdk6',  
'cdk8',  
'cdkn1a',  
'cdkn1b',  
'cdkn2a',  
'cdkn2b',  
'cdkn2c',  
'chek2',  
'cic',  
'crebbp',  
'ctcf',  
'ctla4',  
'ctnnb1',  
'ddr2',  
'dicer1',  
'dnmt3a',  
'dnmt3b',  
'egfr',  
'eif1ax',  
'elf3',  
'ep300',  
'epas1',  
'epcam',  
'erbb2',  
'erbb3',  
'erbb4',  
'ercc2',  
'ercc3',  
'ercc4',  
'erg',  
'esr1',  
'etv1',  
'etv6',  
'ewsr1',  
'ezh2',  
'fam58a',  
'fanca',  
'fbxw7',  
'fgf19',  
'fgf3',  
'fgf4',  
'fgfr1',  
'fgfr2',  
'fgfr3',  
'fgfr4',  
'flt3',  
'foxa1',  
'foxl2',  
'foxo1',  
'fubp1',  
'gata3',  
'gli1',  
'gna11',  
'gnas',  
'h3f3a',  
'hla',  
'hnf1a',  
'hras',  
'idh1',  
'idh2',  
'igf1r',  
'ikbke',  
'ikzf1',  
'il7r',  
'inpp4b',  
'jak1',  
'jak2',  
'jun',  
'kdm5c',  
'kdm6a',  
'kdr',  
'keap1',  
'kit',  
'klf4',  
'kmt2b',  
'kmt2c',

'kmt2d',  
'knstrn',  
'kras',  
'lats1',  
'lats2',  
'map2k1',  
'map2k2',  
'map2k4',  
'map3k1',  
'mdm2',  
'med12',  
'mef2b',  
'men1',  
'met',  
'mga',  
'mlh1',  
'mpl',  
'msh2',  
'msh6',  
'mtor',  
'myc',  
'mycn',  
'myd88',  
'nf1',  
'nf2',  
'nfe2l2',  
'nfkb1a',  
'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',  
'rac1',  
'rad21',  
'rad50',  
'rad51b',  
'rad51c',  
'rad51d',  
'rad54l',  
'raf1',  
'rasa1',  
'rb1',  
'rbm10',  
'ret',  
'rheb',  
'rhoa',  
'riCTOR',  
'rit1',  
'rnf43',



```
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhd',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stag2',
'stat3',
'stk11',
'tcf3',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'xpo1',
'xrcc2']
```

In [99]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 238)
```

#### 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 [100]:

```
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_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=None, intercept_init=None) Fit linear model with Stochastic Gradient Descent
```

```

# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with stochastic gradient descent.
# predict(X) Predict class labels for samples in X.

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

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

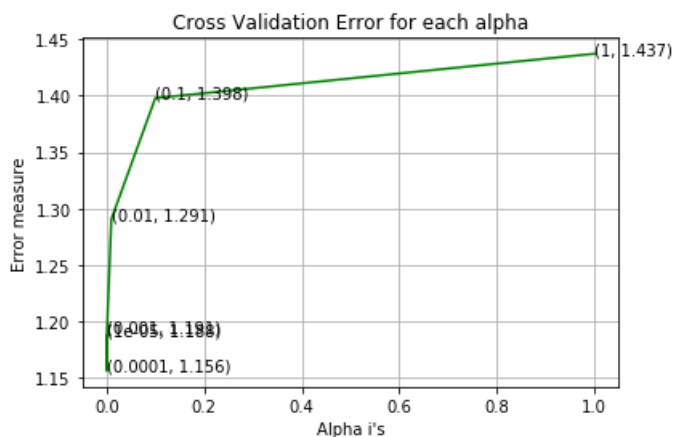
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```

For values of alpha = 1e-05 The log loss is: 1.1876292929381473  
 For values of alpha = 0.0001 The log loss is: 1.156037843129918  
 For values of alpha = 0.001 The log loss is: 1.1910451848145323  
 For values of alpha = 0.01 The log loss is: 1.2905515233975875  
 For values of alpha = 0.1 The log loss is: 1.397844880790661  
 For values of alpha = 1 The log loss is: 1.4372387199886898



For values of best alpha = 0.0001 The train log loss is: 0.998783816257262  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.156037843129918  
 For values of best alpha = 0.0001 The test log loss is: 1.2200553023449674

**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 [101]:

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

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

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

Q6. How many data points in Test and CV datasets are covered by the 238 genes in train dataset?

Ans

1. In test data 641 out of 665 : 96.39097744360903

2. In cross validation data 518 out of 532 : 97.36842105263158

### 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 [102]:

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

Number of Unique Variations : 1930

Truncating_Mutations	61
Amplification	46
Deletion	46
Fusions	20
G12V	4
Overexpression	4
Q61H	3
T58I	3
A146T	2
T167A	2

Name: Variation, dtype: int64

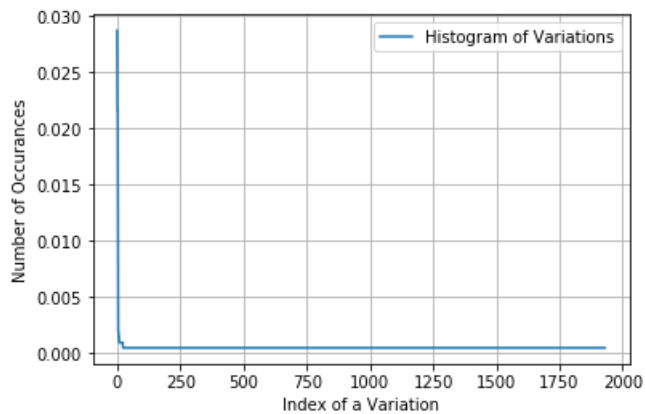
In [103]:

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

Ans: There are 1930 different categories of variations in the train data, and they are distributed as follows

In [104]:

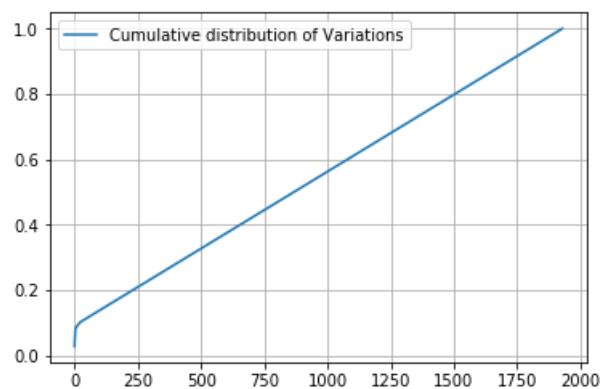
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [105]:

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

```
[0.0287194  0.05037665 0.0720339  ... 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 [106]:

```
# 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 [107]:

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

train\_variation\_feature\_responseCoding is a converted feature using the response coding method. The shape of Variation feature: (1000, 10)

train\_variation\_feature\_onehotCoding is a converted feature using the response coding method. The shape of Variation feature: (2124, 9)

In [108]:

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

In [109]:

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

train\_variation\_feature\_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 1959)

## Q10. How good is this Variation feature in predicting y\_i?

Let's build a model just like the earlier!

In [110]:

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

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

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

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

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

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

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

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
```

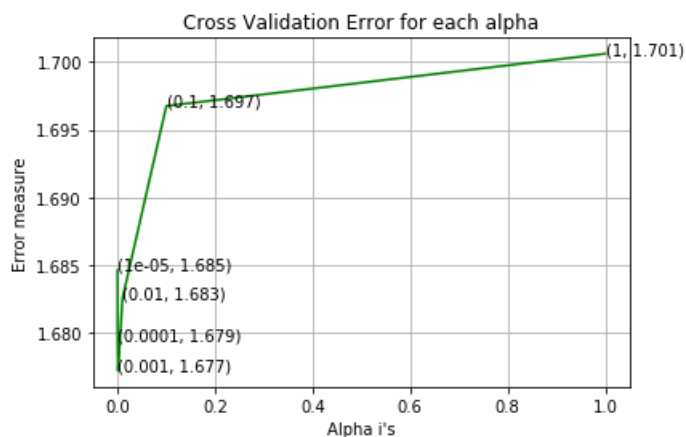
```

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

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

```

For values of alpha = 1e-05 The log loss is: 1.6846263149257228  
 For values of alpha = 0.0001 The log loss is: 1.67940730394681  
 For values of alpha = 0.001 The log loss is: 1.6771772848537378  
 For values of alpha = 0.01 The log loss is: 1.682532575742889  
 For values of alpha = 0.1 The log loss is: 1.696751582211109  
 For values of alpha = 1 The log loss is: 1.700612998962163



For values of best alpha = 0.001 The train log loss is: 1.1030115636926876  
 For values of best alpha = 0.001 The cross validation log loss is: 1.6771772848537378  
 For values of best alpha = 0.001 The test log loss is: 1.716426758460474

**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 [44]:

```

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

```

Q12. How many data points are covered by total 1930 genes in test and cross validation data sets?

Ans

1. In test data 66 out of 665 : 9.924812030075188
2. In cross validation data 57 out of 532 : 10.714285714285714

### 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 predicting y\_i?
5. Is the text feature stable across train test and CV datasets?

6. Is the text feature stable across train, test and CV datasets:

In [111]:

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

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

In [112]:

```
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 [154]:

```
# building a TfidfVectorizer with all the words that occurred minimum 3 times in train data
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
#print(train_text_feature_onehotCoding[0:50])
#Selecting top 1000 words using SelectKBest function
feature_scores = SelectKBest(chi2, k=1000).fit(train_text_feature_onehotCoding, y_train).scores_
feature_scores = np.nan_to_num(feature_scores)
#print(feature_scores)
feature_names = text_vectorizer.get_feature_names()
k_best_features = np.argsort(feature_scores.ravel(), (-1) * 1000)[(-1) * 1000:]
train_text_features = [feature_names[i] for i in k_best_features]
#print(train_text_features[0:200])
#print(top_features.describe())

#train_text_features= text_vectorizer.get_feature_names()

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

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

In [155]:

```
dict_list = []
# dict_list=[] contains 9 dictionaries 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 build 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 [156]:

```
#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 [157]:

```
# 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 [158]:

```
# 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 [159]:

```
#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 [160]:

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

In [161]:

```
# 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_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
```



```

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

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

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

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

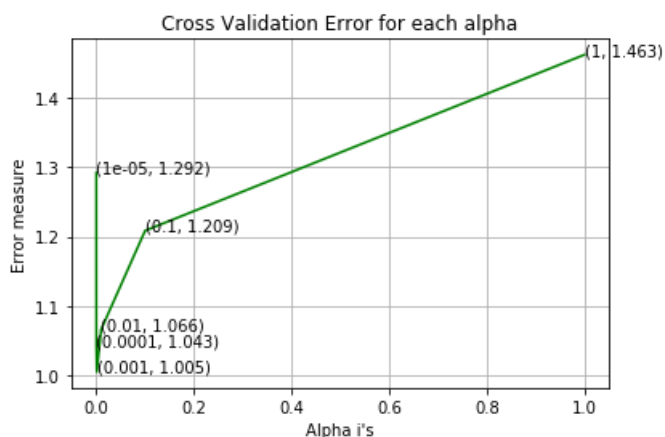
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```

For values of alpha = 1e-05 The log loss is: 1.292163793259139  
 For values of alpha = 0.0001 The log loss is: 1.0433596492025494  
 For values of alpha = 0.001 The log loss is: 1.0045244826636495  
 For values of alpha = 0.01 The log loss is: 1.065842438726074  
 For values of alpha = 0.1 The log loss is: 1.2085824148370075  
 For values of alpha = 1 The log loss is: 1.4628874049504519



For values of best alpha = 0.001 The train log loss is: 0.6279737189167154  
 For values of best alpha = 0.001 The cross validation log loss is: 1.0045244826636495  
 For values of best alpha = 0.001 The test log loss is: 1.1315351766432674

for values of best alpha = 0.001 The test log loss is: 1.1315351/694320/4

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

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

In [166]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()
    feature_scores = SelectKBest(chi2, k=1000).fit(df_text_fea, df['Class']).scores_
    feature_scores = np.nan_to_num(feature_scores)
    #print(feature_scores)
    #feature_names = text_vectorizer.get_feature_names()
    k_best_features = np.argpartition(feature_scores.ravel(), (-1) * 1000)[(-1) * 1000:]
    top_text_features = [df_text_features[i] for i in k_best_features]

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(top_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 [167]:

```
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")
```

2.499 % of word of test data appeared in train data  
2.752 % of word of Cross Validation appeared in train data

## 4. Machine Learning Models

In [168]:

```
#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 [169]:

```
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 [170]:

```
# this function will be used just for naive bayes
```

```

# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

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

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

```

## Stacking the three types of features

In [171]:

```

# merging gene, variance and text features

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

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

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

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

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

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_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))

```

```
ing))
```

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

In [172]:

```
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, 56700)  
(number of data points \* number of features) in test data = (665, 56700)  
(number of data points \* number of features) in cross validation data = (532, 56700)

In [173]:

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

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

## 4.1. Base Line Model

### 4.1.1. Naive Bayes

#### 4.1.1.1. Hyper parameter tuning

In [174]:

```
# 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/
```

```

ithm-1/
# -----

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

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

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

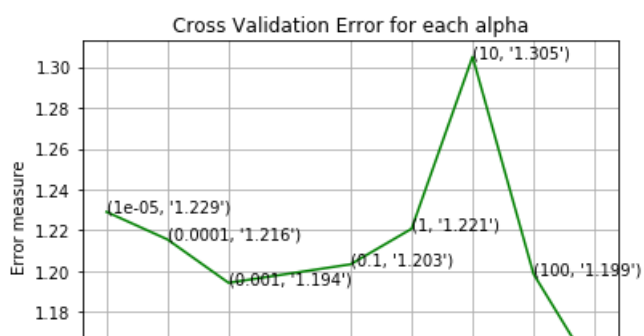
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

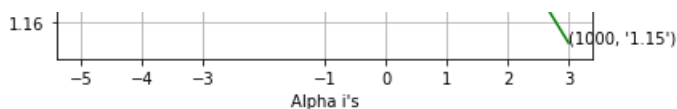
```

```

for alpha = 1e-05
Log Loss : 1.2290354329787514
for alpha = 0.0001
Log Loss : 1.2155044493766882
for alpha = 0.001
Log Loss : 1.194278460114686
for alpha = 0.1
Log Loss : 1.203336568791036
for alpha = 1
Log Loss : 1.2207367880188669
for alpha = 10
Log Loss : 1.30502298711941
for alpha = 100
Log Loss : 1.1985641155851097
for alpha = 1000
Log Loss : 1.1496820219784276

```





For values of best alpha = 1000 The train log loss is: 0.925237205804723

For values of best alpha = 1000 The cross validation log loss is: 1.1496820219784276

For values of best alpha = 1000 The test log loss is: 1.2426499585420714

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

In [175]:

```
# 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/
# -----

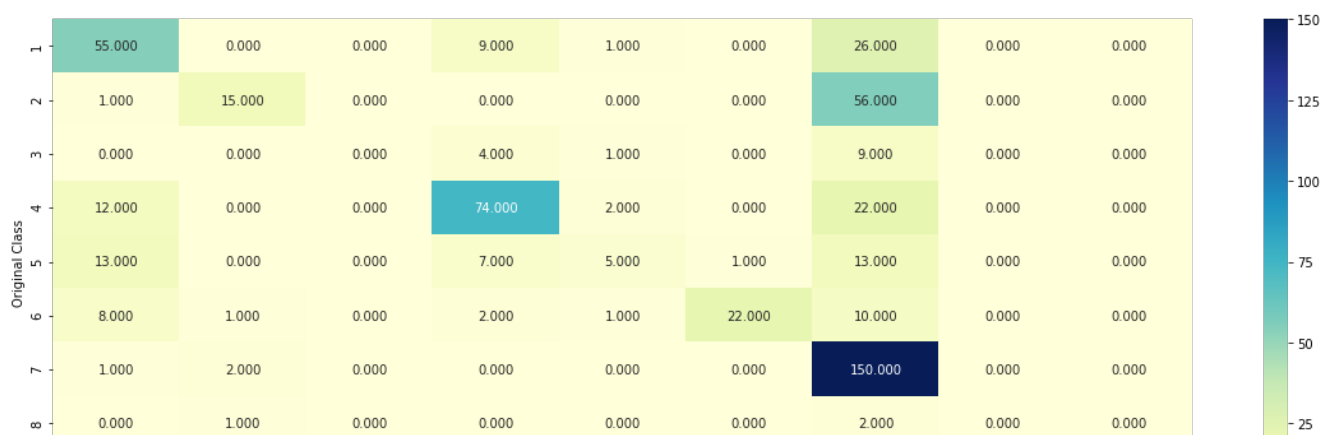
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
# -----
# some of the methods of CalibratedClassifierCV()
# 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
# -----

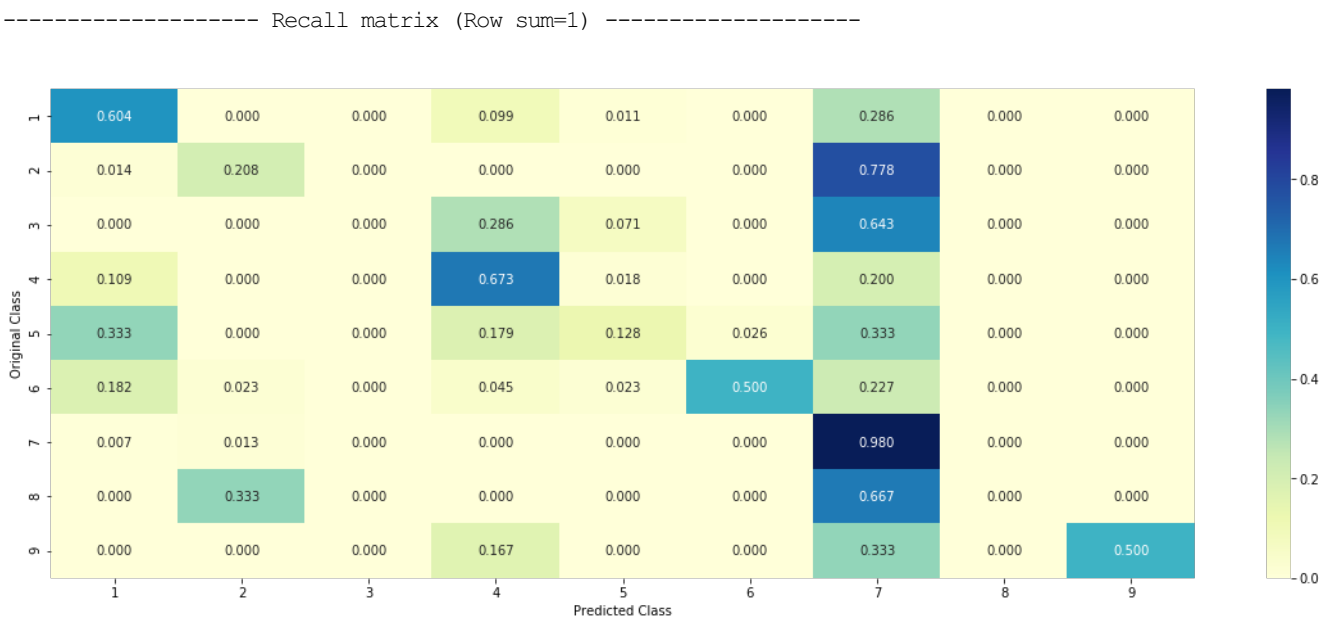
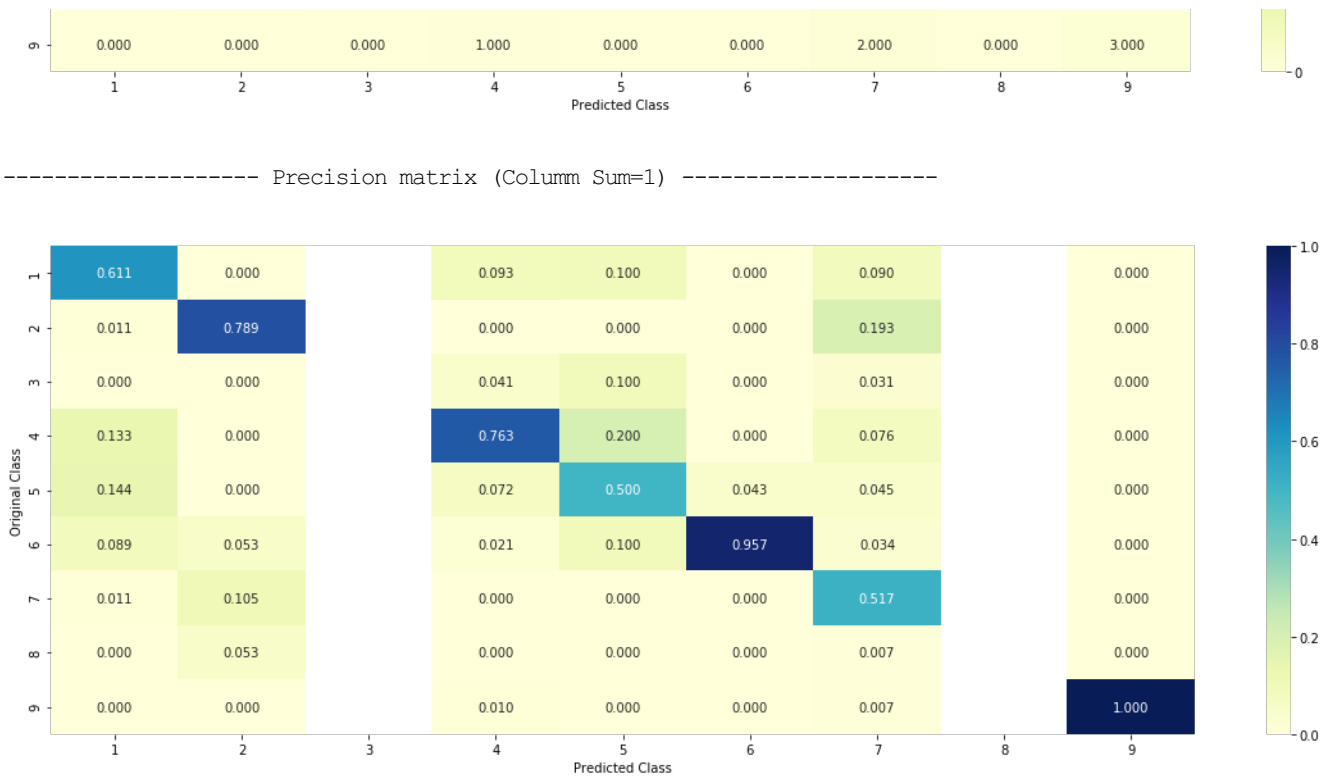
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilities we 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 : 1.1496820219784276

Number of missclassified point : 0.39097744360902253

----- Confusion matrix -----





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

In [176]:

```
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 : 7

Predicted Class Probabilities: [[0.0753 0.0653 0.0086 0.0756 0.0273 0.0255 0.7154 0.0045 0.0027]]

Actual Class : 2

15 Most Important Features [sorted] associated to test data point [Index]

15 Text feature [cells] present in test data point [True]  
16 Text feature [cell] present in test data point [True]  
17 Text feature [activated] present in test data point [True]  
19 Text feature [factor] present in test data point [True]  
20 Text feature [kinase] present in test data point [True]  
22 Text feature [activation] present in test data point [True]  
23 Text feature [contrast] present in test data point [True]  
24 Text feature [expressing] present in test data point [True]  
25 Text feature [presence] present in test data point [True]  
26 Text feature [shown] present in test data point [True]  
27 Text feature [signaling] present in test data point [True]  
28 Text feature [growth] present in test data point [True]  
29 Text feature [phosphorylation] present in test data point [True]  
30 Text feature [also] present in test data point [True]  
31 Text feature [recently] present in test data point [True]  
33 Text feature [however] present in test data point [True]  
34 Text feature [found] present in test data point [True]  
35 Text feature [10] present in test data point [True]  
36 Text feature [suggest] present in test data point [True]  
37 Text feature [inhibitor] present in test data point [True]  
38 Text feature [independent] present in test data point [True]  
39 Text feature [compared] present in test data point [True]  
40 Text feature [higher] present in test data point [True]  
41 Text feature [addition] present in test data point [True]  
42 Text feature [increased] present in test data point [True]  
43 Text feature [treated] present in test data point [True]  
44 Text feature [mutations] present in test data point [True]  
45 Text feature [mechanism] present in test data point [True]  
46 Text feature [similar] present in test data point [True]  
47 Text feature [well] present in test data point [True]  
48 Text feature [previously] present in test data point [True]  
49 Text feature [potential] present in test data point [True]  
50 Text feature [figure] present in test data point [True]  
51 Text feature [treatment] present in test data point [True]  
52 Text feature [showed] present in test data point [True]  
53 Text feature [tyrosine] present in test data point [True]  
54 Text feature [demonstrated] present in test data point [True]  
56 Text feature [followed] present in test data point [True]  
57 Text feature [3b] present in test data point [True]  
58 Text feature [without] present in test data point [True]  
59 Text feature [interestingly] present in test data point [True]  
60 Text feature [consistent] present in test data point [True]  
61 Text feature [constitutive] present in test data point [True]  
66 Text feature [1a] present in test data point [True]  
67 Text feature [confirmed] present in test data point [True]  
69 Text feature [reported] present in test data point [True]  
70 Text feature [total] present in test data point [True]  
71 Text feature [activating] present in test data point [True]  
72 Text feature [mutant] present in test data point [True]  
73 Text feature [using] present in test data point [True]  
74 Text feature [serum] present in test data point [True]  
75 Text feature [furthermore] present in test data point [True]  
76 Text feature [including] present in test data point [True]  
77 Text feature [respectively] present in test data point [True]  
78 Text feature [may] present in test data point [True]  
79 Text feature [sensitive] present in test data point [True]  
80 Text feature [absence] present in test data point [True]  
81 Text feature [described] present in test data point [True]  
82 Text feature [obtained] present in test data point [True]  
83 Text feature [report] present in test data point [True]  
84 Text feature [mutation] present in test data point [True]  
85 Text feature [expression] present in test data point [True]  
87 Text feature [inhibition] present in test data point [True]  
89 Text feature [fig] present in test data point [True]  
90 Text feature [pathways] present in test data point [True]  
91 Text feature [3a] present in test data point [True]  
92 Text feature [performed] present in test data point [True]  
94 Text feature [increase] present in test data point [True]  
95 Text feature [proliferation] present in test data point [True]  
96 Text feature [antibodies] present in test data point [True]  
97 Text feature [inhibitors] present in test data point [True]  
98 Text feature [recent] present in test data point [True]  
99 Text feature [two] present in test data point [True]  
Out of the top 100 features 73 are present in query point



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

In [178]:

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

Predicted Class : 7

Predicted Class Probabilities: [[0.1045 0.212 0.0174 0.2032 0.0416 0.0325 0.3798 0.0059 0.0032]]

Actual Class : 7

```
-----
15 Text feature [cells] present in test data point [True]
16 Text feature [cell] present in test data point [True]
17 Text feature [activated] present in test data point [True]
19 Text feature [factor] present in test data point [True]
20 Text feature [kinase] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
22 Text feature [activation] present in test data point [True]
23 Text feature [contrast] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [presence] present in test data point [True]
26 Text feature [shown] present in test data point [True]
27 Text feature [signaling] present in test data point [True]
28 Text feature [growth] present in test data point [True]
29 Text feature [phosphorylation] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [recently] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [found] present in test data point [True]
35 Text feature [10] present in test data point [True]
36 Text feature [suggest] present in test data point [True]
37 Text feature [inhibitor] present in test data point [True]
38 Text feature [independent] present in test data point [True]
41 Text feature [addition] present in test data point [True]
42 Text feature [increased] present in test data point [True]
43 Text feature [treated] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [mechanism] present in test data point [True]
46 Text feature [similar] present in test data point [True]
47 Text feature [well] present in test data point [True]
48 Text feature [previously] present in test data point [True]
49 Text feature [potential] present in test data point [True]
50 Text feature [figure] present in test data point [True]
51 Text feature [treatment] present in test data point [True]
54 Text feature [demonstrated] present in test data point [True]
55 Text feature [constitutively] present in test data point [True]
56 Text feature [followed] present in test data point [True]
57 Text feature [3b] present in test data point [True]
58 Text feature [without] present in test data point [True]
60 Text feature [consistent] present in test data point [True]
61 Text feature [constitutive] present in test data point [True]
62 Text feature [enhanced] present in test data point [True]
66 Text feature [1a] present in test data point [True]
67 Text feature [confirmed] present in test data point [True]
68 Text feature [various] present in test data point [True]
69 Text feature [reported] present in test data point [True]
70 Text feature [total] present in test data point [True]
71 Text feature [activating] present in test data point [True]
72 Text feature [mutant] present in test data point [True]
73 Text feature [using] present in test data point [True]
74 Text feature [serum] present in test data point [True]
75 Text feature [furthermore] present in test data point [True]
76 Text feature [including] present in test data point [True]
77 Text feature [respectively] present in test data point [True]
78 Text feature [may] present in test data point [True]
80 Text feature [absence] present in test data point [True]
81 Text feature [described] present in test data point [True]
82 Text feature [elaborated] present in test data point [True]
```

```

82 Text feature [obtained] present in test data point [True]
83 Text feature [report] present in test data point [True]
84 Text feature [mutation] present in test data point [True]
85 Text feature [expression] present in test data point [True]
87 Text feature [inhibition] present in test data point [True]
89 Text feature [fig] present in test data point [True]
90 Text feature [pathways] present in test data point [True]
91 Text feature [3a] present in test data point [True]
92 Text feature [performed] present in test data point [True]
93 Text feature [approximately] present in test data point [True]
94 Text feature [increase] present in test data point [True]
95 Text feature [proliferation] present in test data point [True]
96 Text feature [antibodies] present in test data point [True]
97 Text feature [inhibitors] present in test data point [True]
98 Text feature [recent] present in test data point [True]
99 Text feature [two] present in test data point [True]
Out of the top 100 features 72 are present in query point

```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

In [179]:

```

# 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-neighbors-geometric-intuition-with-a-toy-example-1/
#-----

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

alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    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 probabilities we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):

```

```

ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

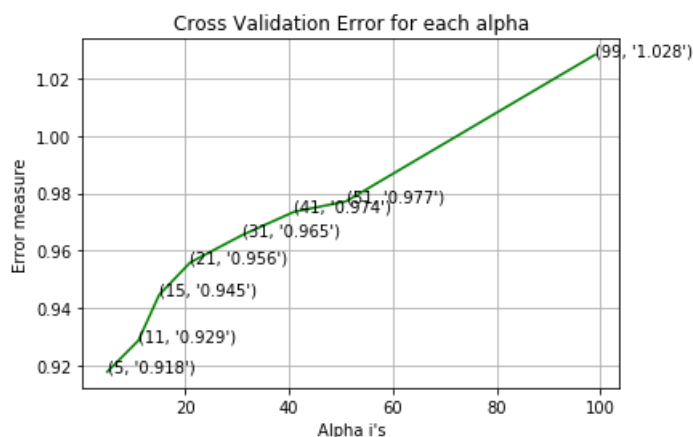
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 5
Log Loss : 0.9179123623438137
for alpha = 11
Log Loss : 0.9287858847330114
for alpha = 15
Log Loss : 0.9446633765282862
for alpha = 21
Log Loss : 0.9559507451499635
for alpha = 31
Log Loss : 0.9654357600716932
for alpha = 41
Log Loss : 0.9735418296140261
for alpha = 51
Log Loss : 0.977148007647987
for alpha = 99
Log Loss : 1.0281383071152987

```



```

For values of best alpha = 5 The train log loss is: 0.4965916554552862
For values of best alpha = 5 The cross validation log loss is: 0.9179123623438137
For values of best alpha = 5 The test log loss is: 1.0755631051180516

```

## 4.2.2. Testing the model with best hyper paramters

In [180]:

```

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

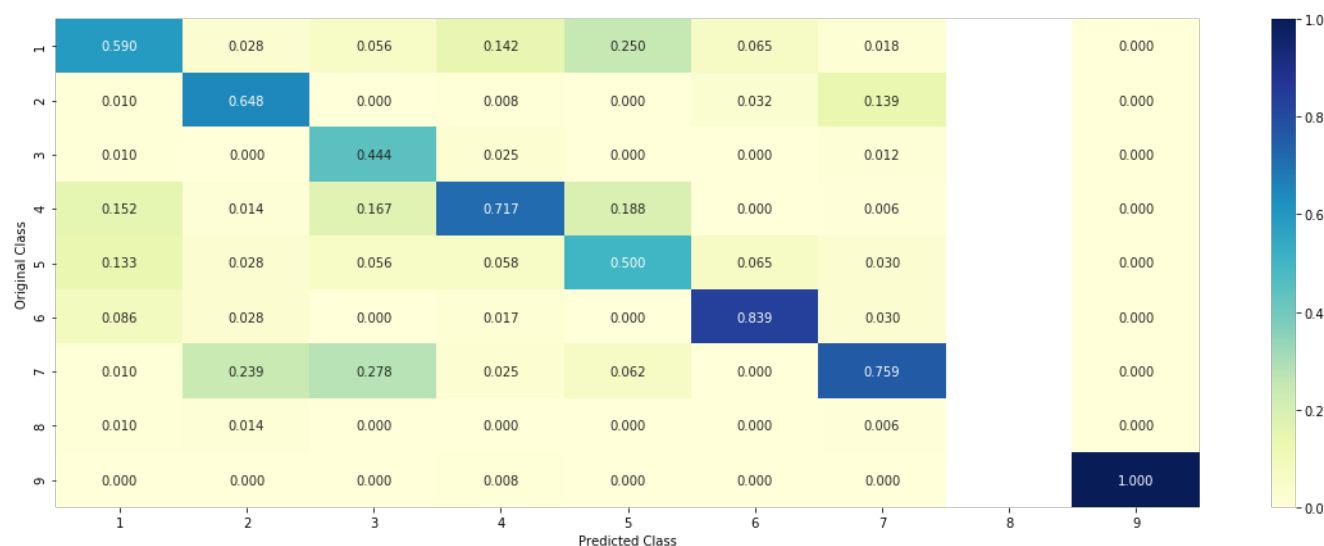
Log loss : 0.9179123623438137

Number of mis-classified points : 0.3101503759398496

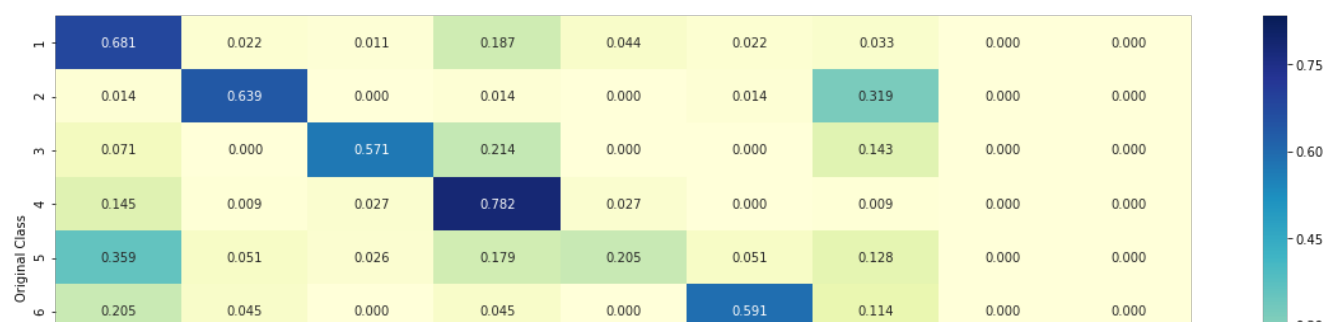
----- Confusion matrix -----

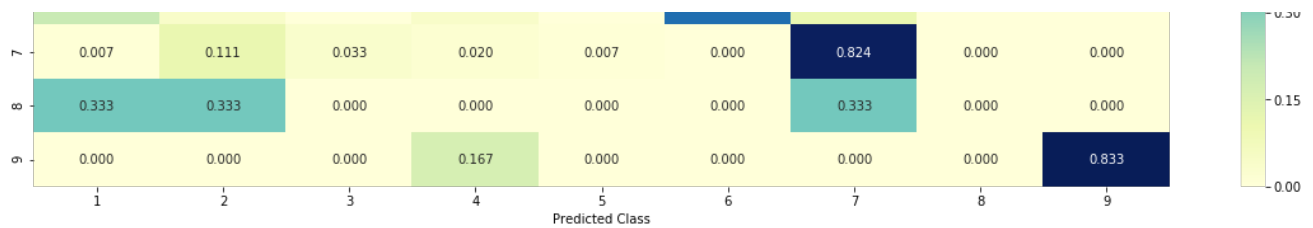


----- Precision matrix (Column Sum=1) -----



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





### 4.2.3. Sample Query point -1

In [181]:

```
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 : 7  
 Actual Class : 2  
 The 5 nearest neighbours of the test points belongs to classes [7 7 7 7 7]  
 Fequency of nearest points : Counter({7: 5})

### 4.2.4. Sample Query Point-2

In [182]:

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

test_point_index = 100

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

Predicted Class : 7  
 Actual Class : 7  
 the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [7 2 2 7 7]  
 Fequency of nearest points : Counter({7: 3, 2: 2})

## 4.3. Logistic Regression

### 4.3.1. With Class balancing

#### 4.3.1.1. Hyper paramter tuning

In [183]:

```
# 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=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
rn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# 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 probabilities we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

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

```

```

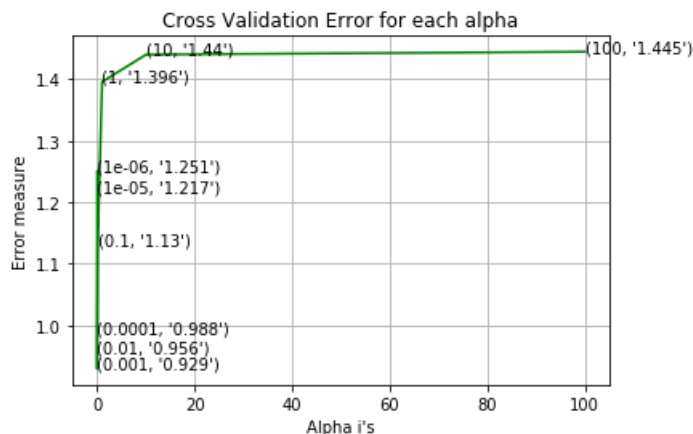
for alpha = 1e-06
Log Loss : 1.2505982675935539

```

```

for alpha = 1e-05
Log Loss : 1.2170652128525883
for alpha = 0.0001
Log Loss : 0.988405331215221
for alpha = 0.001
Log Loss : 0.9294179136812935
for alpha = 0.01
Log Loss : 0.955551977128828
for alpha = 0.1
Log Loss : 1.130096066350409
for alpha = 1
Log Loss : 1.3956038453788127
for alpha = 10
Log Loss : 1.439912697242982
for alpha = 100
Log Loss : 1.444588556836223

```



For values of best alpha = 0.001 The train log loss is: 0.5385867659816423  
For values of best alpha = 0.001 The cross validation log loss is: 0.9294179136812935  
For values of best alpha = 0.001 The test log loss is: 1.0789101779381822

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

In [184]:

```

# 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=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

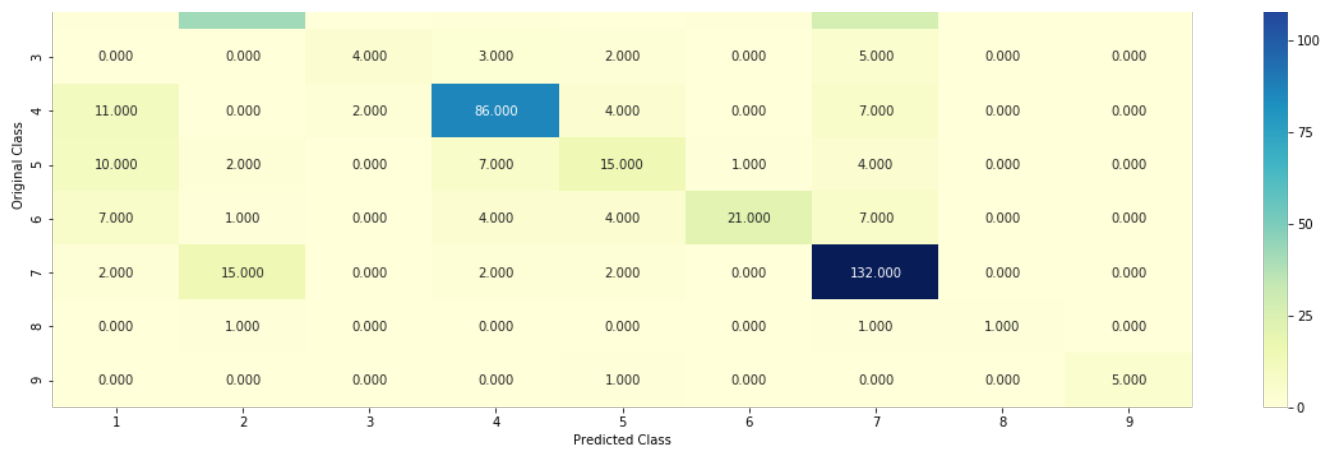
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuiti
on-1/
#-----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random
state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,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

```

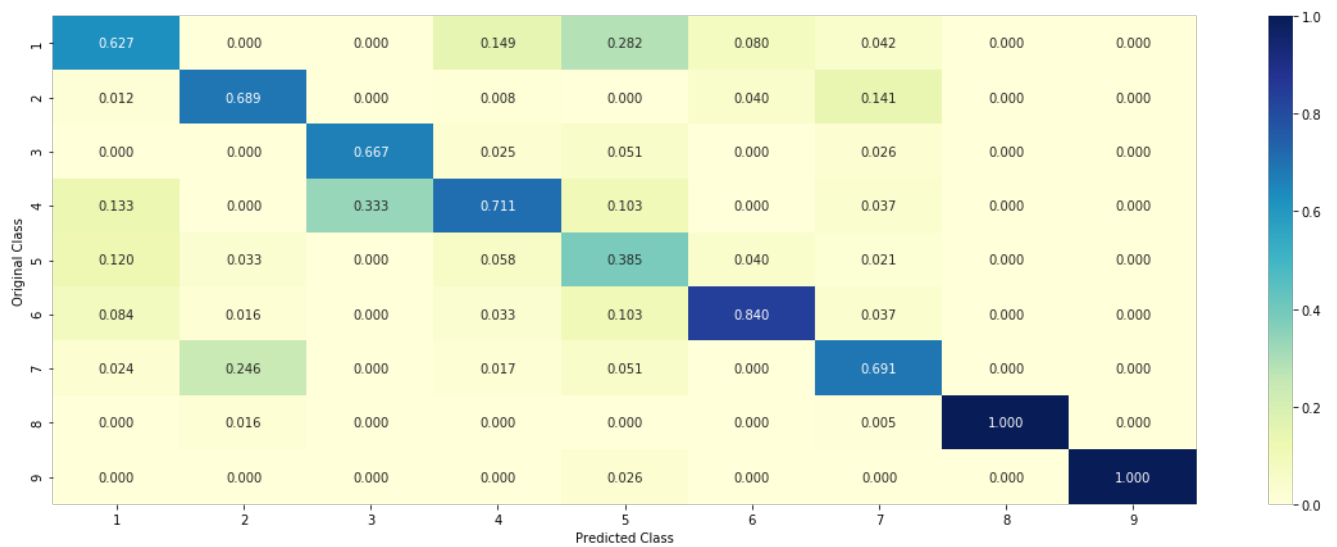
Log loss : 0.9294179136812935  
Number of mis-classified points : 0.32706766917293234

----- Confusion matrix -----

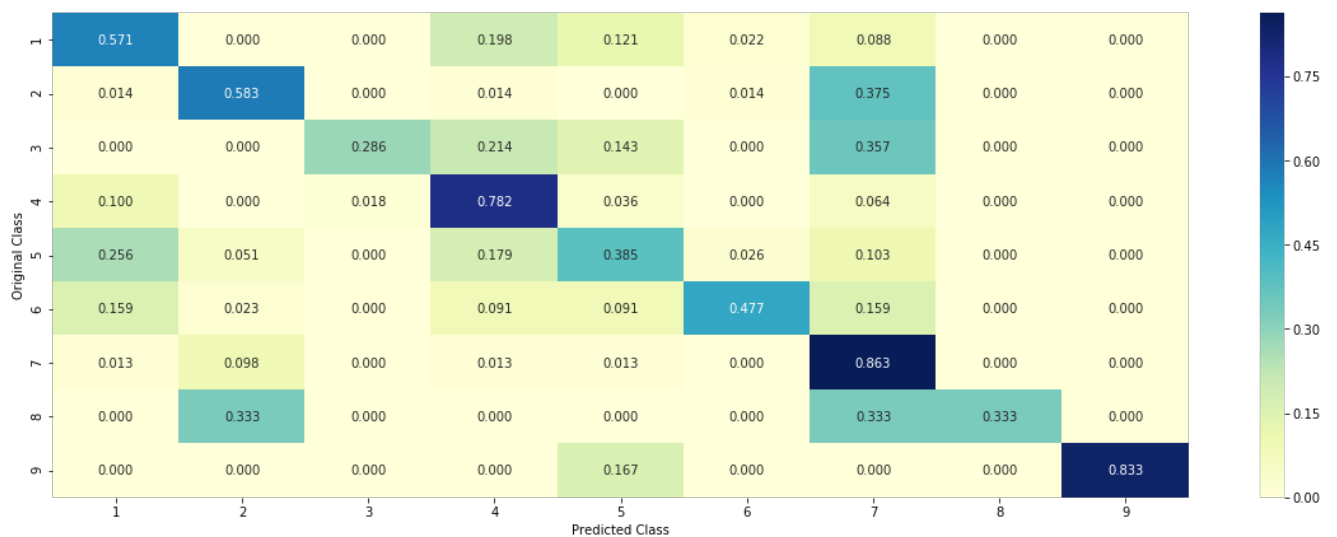
1	52.000	0.000	0.000	18.000	11.000	2.000	8.000	0.000	0.000
2	1.000	42.000	0.000	1.000	0.000	1.000	27.000	0.000	0.000



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



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



#### 4.3.1.3. Feature Importance

In [185]:

```
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([increasingorder_ind, "Gene", "Yes"])
elif i < 18:
    tabulte_list.append([increasingorder_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([increasingorder_ind, train_text_features[i], yes_no])
increasingorder_ind += 1
print(word_present, "most important features are present in our query point")
print("-"*50)
print("The features that are most important of the ", predicted_cls[0], " class:")
print(tabulate(tabulte_list, headers=["Index", "Feature name", "Present or Not"]))

```

#### 4.3.1.3.1. Incorrectly Classified point

In [186]:

```

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

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.063  0.058  0.0029 0.0646 0.0096 0.0116 0.7639 0.0152 0.0111]]
Actual Class : 2

```

```

-----
22 Text feature [constitutive] present in test data point [True]
38 Text feature [activated] present in test data point [True]
73 Text feature [serum] present in test data point [True]
79 Text feature [temporary] present in test data point [True]
84 Text feature [nf] present in test data point [True]
118 Text feature [doses] present in test data point [True]
128 Text feature [proliferate] present in test data point [True]
148 Text feature [biopsy] present in test data point [True]
184 Text feature [activation] present in test data point [True]
186 Text feature [gauge] present in test data point [True]
202 Text feature [bypass] present in test data point [True]
209 Text feature [murine] present in test data point [True]
222 Text feature [xenograft] present in test data point [True]
248 Text feature [activating] present in test data point [True]
271 Text feature [transforming] present in test data point [True]
289 Text feature [oncogene] present in test data point [True]
297 Text feature [transformed] present in test data point [True]
313 Text feature [cysteine] present in test data point [True]
440 Text feature [hydropathy] present in test data point [True]
449 Text feature [cd79a] present in test data point [True]
Out of the top 500 features 20 are present in query point

```

#### 4.3.1.3.2. Correctly Classified point

In [187]:

```

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))

```

```

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

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0516 0.3425 0.0114 0.1288 0.0269 0.0167 0.4064 0.0087 0.0071]]

Actual Class : 7

```

-----
22 Text feature [constitutive] present in test data point [True]
38 Text feature [activated] present in test data point [True]
44 Text feature [mitogen] present in test data point [True]
46 Text feature [constitutively] present in test data point [True]
73 Text feature [serum] present in test data point [True]
108 Text feature [downstream] present in test data point [True]
128 Text feature [proliferate] present in test data point [True]
184 Text feature [activation] present in test data point [True]
190 Text feature [stems] present in test data point [True]
209 Text feature [murine] present in test data point [True]
213 Text feature [subcutaneous] present in test data point [True]
230 Text feature [3t3] present in test data point [True]
248 Text feature [activating] present in test data point [True]
271 Text feature [transforming] present in test data point [True]
280 Text feature [subcutaneously] present in test data point [True]
289 Text feature [oncogene] present in test data point [True]
297 Text feature [transformed] present in test data point [True]
322 Text feature [refractory] present in test data point [True]
497 Text feature [hyperplasia] present in test data point [True]
Out of the top 500 features 19 are present in query point

```

## 4.3.2. Without Class balancing

### 4.3.2.1. Hyper paramter tuning

In [188]:

```

# 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/
#-----

# 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)

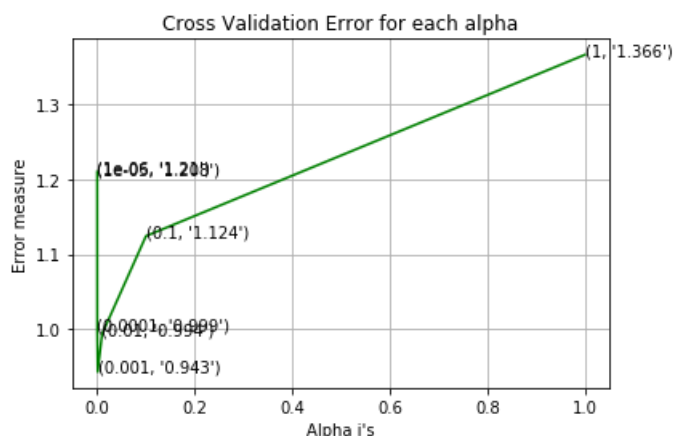
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_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 : 1.210022417091008
for alpha = 1e-05
Log Loss : 1.2077734057414014
for alpha = 0.0001
Log Loss : 0.999137425677682
for alpha = 0.001
Log Loss : 0.9432172777584038
for alpha = 0.01
Log Loss : 0.9936971838165556
for alpha = 0.1
Log Loss : 1.1243007169847556
for alpha = 1
Log Loss : 1.366059447456724

```



```

For values of best alpha = 0.001 The train log loss is: 0.5344285120747495
For values of best alpha = 0.001 The cross validation log loss is: 0.9432172777584038
For values of best alpha = 0.001 The test log loss is: 1.0927553636184113

```

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

In [189]:

```
# 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.

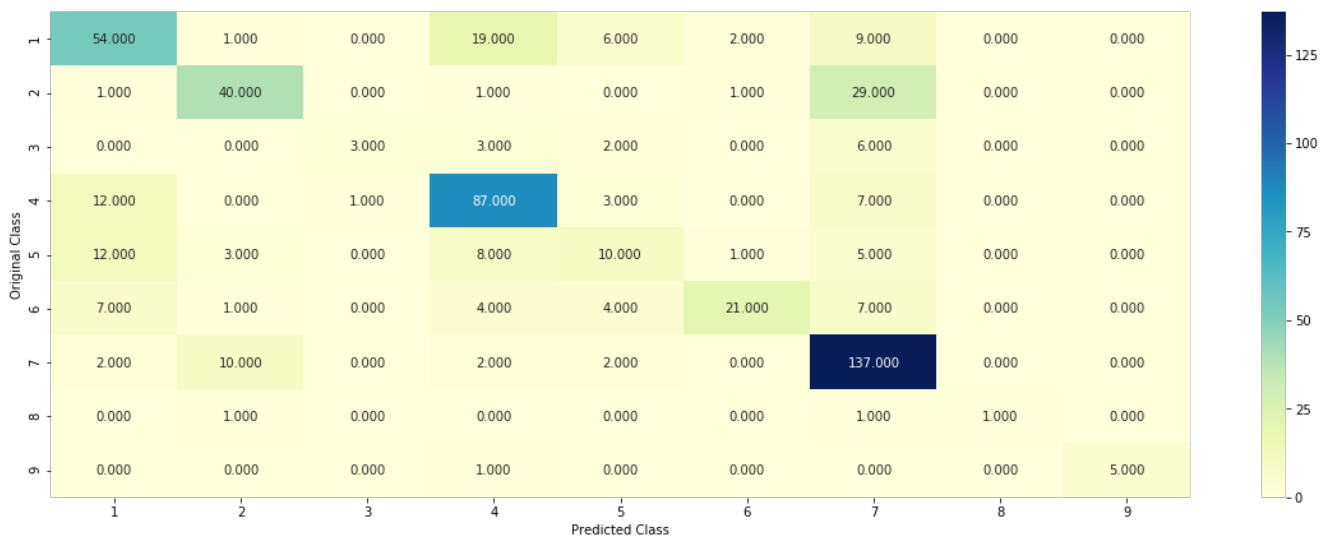
#-----
# video link:
#-----

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)
```

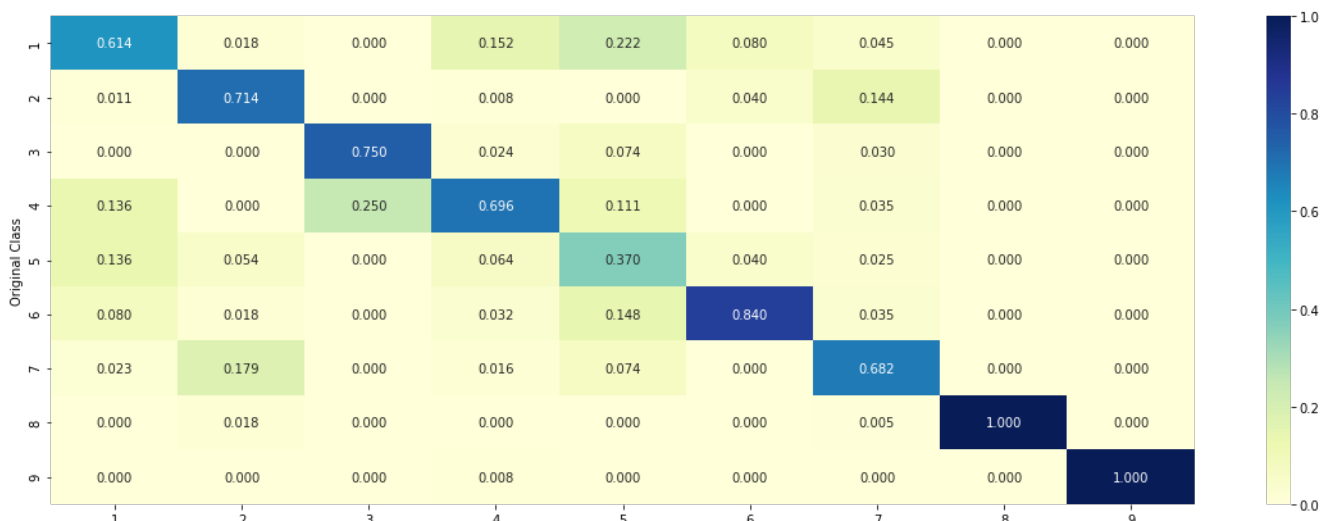
Log loss : 0.9432172777584038

Number of mis-classified points : 0.32706766917293234

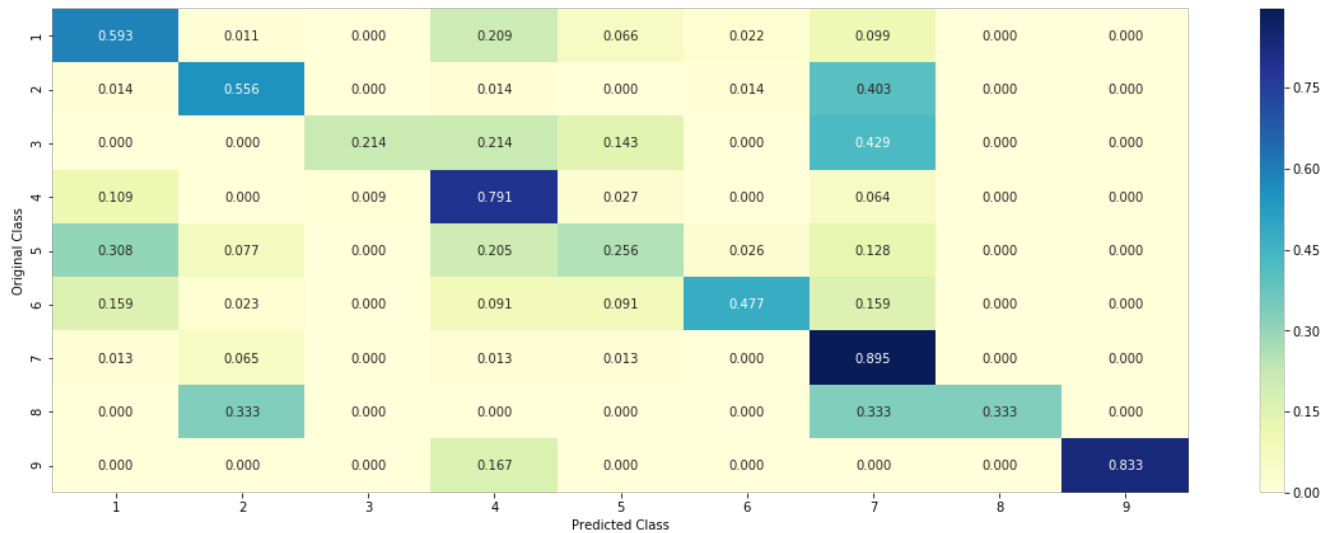
----- Confusion matrix -----



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



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



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

In [190]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_) [predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 5  
 Predicted Class Probabilities: [[0.215 0.0956 0.0429 0.1996 0.22 0.0371 0.168 0.0199 0.0019]]  
 Actual Class : 1

-----  
 182 Text feature [antigenicity] present in test data point [True]  
 399 Text feature [885] present in test data point [True]  
 403 Text feature [nonionic] present in test data point [True]  
 Out of the top 500 features 3 are present in query point

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

In [191]:

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

Predicted Class : 7  
 Predicted Class Probabilities: [[0.0502 0.3333 0.0096 0.128 0.0242 0.0151 0.4287 0.0079 0.0028]]

Actual Class : 7

```
-----
37 Text feature [constitutive] present in test data point [True]
91 Text feature [activated] present in test data point [True]
96 Text feature [constitutively] present in test data point [True]
111 Text feature [mitogen] present in test data point [True]
135 Text feature [serum] present in test data point [True]
140 Text feature [downstream] present in test data point [True]
196 Text feature [activation] present in test data point [True]
201 Text feature [proliferate] present in test data point [True]
222 Text feature [3t3] present in test data point [True]
224 Text feature [activating] present in test data point [True]
236 Text feature [transforming] present in test data point [True]
278 Text feature [transformed] present in test data point [True]
303 Text feature [subcutaneous] present in test data point [True]
305 Text feature [subcutaneously] present in test data point [True]
312 Text feature [murine] present in test data point [True]
324 Text feature [stems] present in test data point [True]
328 Text feature [signaling] present in test data point [True]
346 Text feature [oncogene] present in test data point [True]
391 Text feature [refractory] present in test data point [True]
422 Text feature [ectopically] present in test data point [True]
436 Text feature [expressing] present in test data point [True]
466 Text feature [transform] present in test data point [True]
494 Text feature [enhanced] present in test data point [True]
Out of the top 500 features 23 are present in query point
```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper parameter tuning

In [192]:

```
# read more about support vector machines with linear kernels 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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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

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

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

```

clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

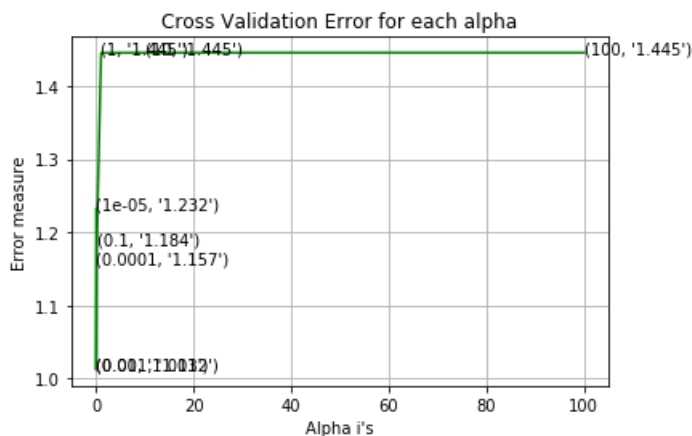
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for C = 1e-05
Log Loss : 1.2318174467690552
for C = 0.0001
Log Loss : 1.1571944407968615
for C = 0.001
Log Loss : 1.011959561311857
for C = 0.01
Log Loss : 1.0134227076887183
for C = 0.1
Log Loss : 1.18434867419452
for C = 1
Log Loss : 1.4450503054602675
for C = 10
Log Loss : 1.4452972534705137
for C = 100
Log Loss : 1.4452840637088957

```



```

For values of best alpha = 0.001 The train log loss is: 0.5920124165653546
For values of best alpha = 0.001 The cross validation log loss is: 1.011959561311857
For values of best alpha = 0.001 The test log loss is: 1.1438726111425028

```

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

In [193]:

```
# read more about support vector machines with linear kernels 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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

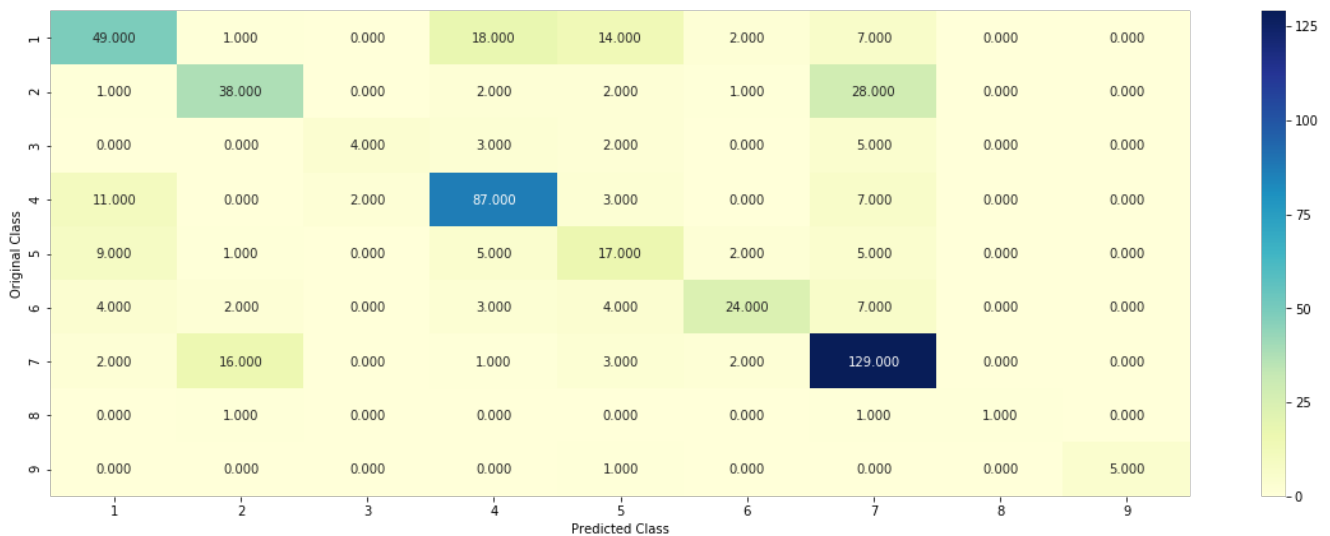
# Some of methods of SVM()
# fit(X, y, [sample weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42,class_weight='balanced')
predict and plot confusion matrix(train x onehotCoding, train y,cv x onehotCoding,cv y, clf)
```

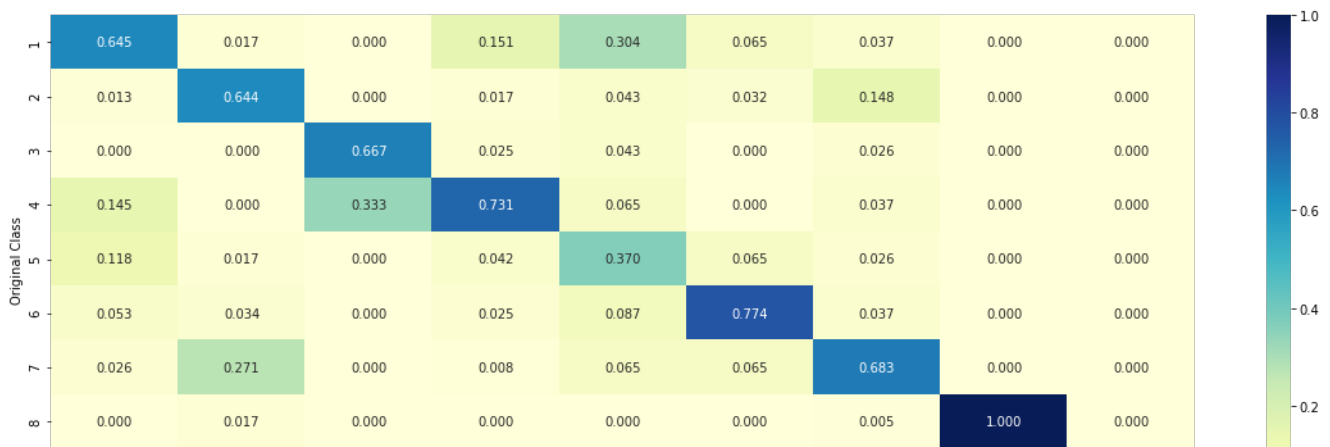
Log loss : 1.011959561311857

Number of mis-classified points : 0.33458646616541354

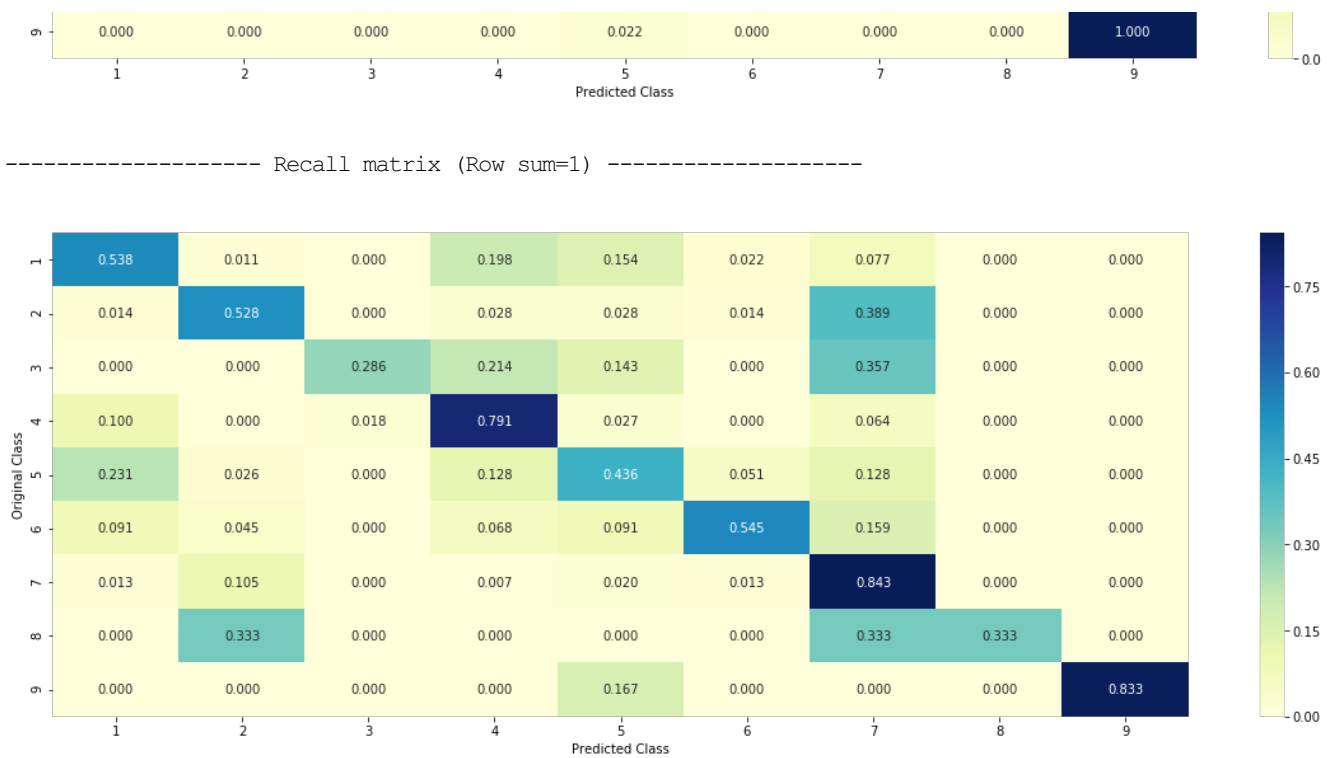
```
----- Confusion matrix -----
```



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







### 4.3.3. Feature Importance

#### 4.3.3.1. For Incorrectly classified point

In [194]:

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

Predicted Class : 9  
Predicted Class Probabilities: [[0.0503 0.0699 0.0129 0.0373 0.1391 0.0121 0.1398 0.0284 0.5103]]  
Actual Class : 8

```
-----
14 Text feature [idhlr132h] present in test data point [True]
23 Text feature [splitless] present in test data point [True]
24 Text feature [5ms] present in test data point [True]
25 Text feature [7890a] present in test data point [True]
26 Text feature [ramped] present in test data point [True]
27 Text feature [g515a] present in test data point [True]
55 Text feature [bc012846] present in test data point [True]
56 Text feature [ab55271] present in test data point [True]
57 Text feature [redissolving] present in test data point [True]
58 Text feature [bc009244] present in test data point [True]
61 Text feature [orbitrap] present in test data point [True]
64 Text feature [na93lv] present in test data point [True]
109 Text feature [hydroxyglutarate] present in test data point [True]
129 Text feature [reductive] present in test data point [True]
159 Text feature [idh3] present in test data point [True]
167 Text feature [nadp] present in test data point [True]
170 Text feature [kolker] present in test data point [True]
178 Text feature [proligo] present in test data point [True]
180 Text feature [carboxylation] present in test data point [True]
```

```

197 Text feature [r140q] present in test data point [True]
201 Text feature [r172k] present in test data point [True]
215 Text feature [interconvert] present in test data point [True]
236 Text feature [potb7] present in test data point [True]
253 Text feature [idh2] present in test data point [True]
299 Text feature [mitochondrial] present in test data point [True]
300 Text feature [citrate] present in test data point [True]
319 Text feature [siebert] present in test data point [True]
336 Text feature [nad] present in test data point [True]
341 Text feature [colman] present in test data point [True]
421 Text feature [mardis] present in test data point [True]
429 Text feature [organic] present in test data point [True]
433 Text feature [orient] present in test data point [True]
446 Text feature [idh1] present in test data point [True]
455 Text feature [evaporation] present in test data point [True]
463 Text feature [idh] present in test data point [True]
469 Text feature [ketoglutarate] present in test data point [True]
470 Text feature [citric] present in test data point [True]
477 Text feature [intrasample] present in test data point [True]
480 Text feature [latini] present in test data point [True]
Out of the top 500 features 39 are present in query point

```

#### 4.3.3.2. For Incorrectly classified point

In [195]:

```

test_point_index = 102
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 : 2
Predicted Class Probabilities: [[0.0522 0.3478 0.0187 0.1505 0.0456 0.0335 0.3375 0.0068 0.0074]]
Actual Class : 7

```

```

-----
184 Text feature [fused] present in test data point [True]
489 Text feature [movement] present in test data point [True]
Out of the top 500 features 2 are present in query point

```

## 4.5 Random Forest Classifier

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

In [196]:

```

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

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

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

```

```

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# 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 : 1.2149931739464976
for n_estimators = 100 and max depth = 10
Log Loss : 1.1499868331487566
for n_estimators = 200 and max depth = 5
Log Loss : 1.2087093181058821
for n_estimators = 200 and max depth = 10
Log Loss : 1.146433856854096
for n_estimators = 500 and max depth = 5
Log Loss : 1.191177448428674
for n_estimators = 500 and max depth = 10

```

```

for n_estimators = 500 and max depth = 10
Log Loss : 1.135156575295882
for n_estimators = 1000 and max depth = 5
Log Loss : 1.1959581979908445
for n_estimators = 1000 and max depth = 10
Log Loss : 1.1368138475615197
for n_estimators = 2000 and max depth = 5
Log Loss : 1.1929543915051388
for n_estimators = 2000 and max depth = 10
Log Loss : 1.133502318305912
For values of best estimator = 2000 The train log loss is: 0.6535074527032497
For values of best estimator = 2000 The cross validation log loss is: 1.133502318305912
For values of best estimator = 2000 The test log loss is: 1.1725063304634953

```

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

In [197]:

```

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

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

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

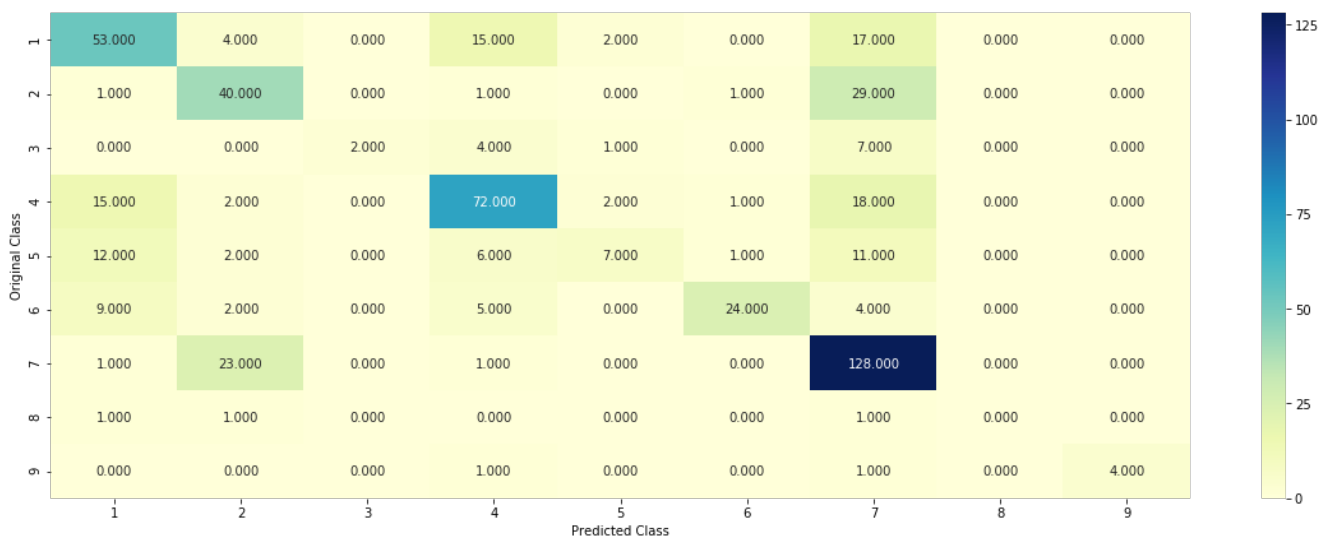
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and
-their-construction-2/
# -----

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

```

Log loss : 1.133502318305912  
Number of mis-classified points : 0.37969924812030076

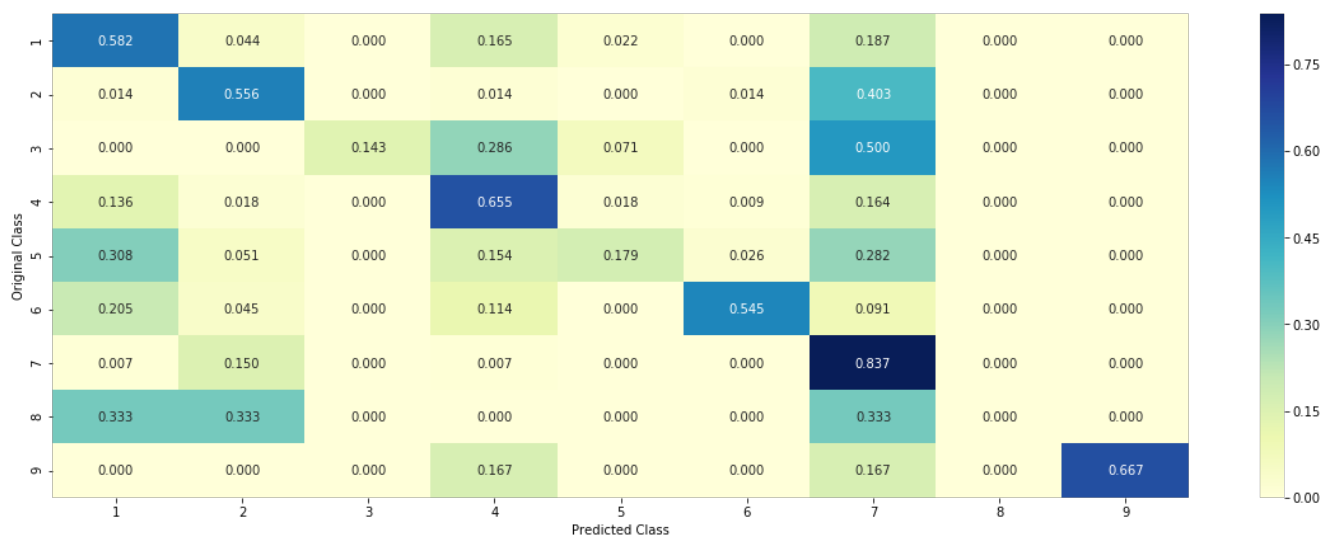
----- Confusion matrix -----



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



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



## 4.5.3. Feature Importance

### 4.5.3.1. Incorrectly Classified point

In [213]:

```
# 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 = RandomForestClassifier(n_estimators=2000, 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 : 7
Predicted Class Probabilities: [[0.0935 0.1696 0.0281 0.1175 0.0586 0.0608 0.4638 0.0069 0.0014]]
Actual Class : 2

```

```

-----
0 Text feature [activation] present in test data point [True]
1 Text feature [activated] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [kinase] present in test data point [True]
4 Text feature [tyrosine] present in test data point [True]
5 Text feature [nonsense] present in test data point [True]
6 Text feature [transforming] present in test data point [True]
7 Text feature [unstable] present in test data point [True]
8 Text feature [therapy] present in test data point [True]
9 Text feature [inhibitor] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [function] present in test data point [True]
12 Text feature [akt] present in test data point [True]
14 Text feature [loss] present in test data point [True]
15 Text feature [trials] present in test data point [True]
16 Text feature [growth] present in test data point [True]
17 Text feature [phosphorylation] present in test data point [True]
18 Text feature [inhibited] present in test data point [True]
19 Text feature [pathogenic] present in test data point [True]
21 Text feature [survival] present in test data point [True]
23 Text feature [receptor] present in test data point [True]
24 Text feature [signaling] present in test data point [True]
25 Text feature [suppressor] present in test data point [True]
28 Text feature [constitutive] present in test data point [True]
30 Text feature [missense] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
33 Text feature [inhibitors] present in test data point [True]
34 Text feature [variants] present in test data point [True]
35 Text feature [kinases] present in test data point [True]
36 Text feature [treated] present in test data point [True]
38 Text feature [yeast] present in test data point [True]
39 Text feature [cell] present in test data point [True]
41 Text feature [inhibition] present in test data point [True]
43 Text feature [drug] present in test data point [True]
44 Text feature [oncogenic] present in test data point [True]
50 Text feature [proliferation] present in test data point [True]
51 Text feature [amplification] present in test data point [True]
52 Text feature [dose] present in test data point [True]
55 Text feature [stability] present in test data point [True]
58 Text feature [lines] present in test data point [True]
63 Text feature [factor] present in test data point [True]
69 Text feature [activate] present in test data point [True]
71 Text feature [functional] present in test data point [True]
72 Text feature [serum] present in test data point [True]
75 Text feature [atp] present in test data point [True]
76 Text feature [proteins] present in test data point [True]
78 Text feature [effective] present in test data point [True]
80 Text feature [phosphatase] present in test data point [True]
92 Text feature [expressing] present in test data point [True]
93 Text feature [patients] present in test data point [True]
97 Text feature [cells] present in test data point [True]
98 Text feature [daily] present in test data point [True]
Out of the top 100 features 52 are present in query point

```

#### 4.5.3.2. Inorrectly Classified point

In [199]:

```

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

```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1052 0.1699 0.0238 0.1837 0.0528 0.0445 0.4021 0.0097 0.0083]]
Actual Class : 7
```

```
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activation] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [activated] present in test data point [True]
5 Text feature [loss] present in test data point [True]
6 Text feature [inhibitor] present in test data point [True]
7 Text feature [constitutive] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
9 Text feature [inhibitors] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [treatment] present in test data point [True]
12 Text feature [transforming] present in test data point [True]
13 Text feature [growth] present in test data point [True]
15 Text feature [suppressor] present in test data point [True]
16 Text feature [function] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
20 Text feature [receptor] present in test data point [True]
22 Text feature [defective] present in test data point [True]
23 Text feature [akt] present in test data point [True]
24 Text feature [constitutively] present in test data point [True]
25 Text feature [erk] present in test data point [True]
27 Text feature [inhibited] present in test data point [True]
31 Text feature [downstream] present in test data point [True]
32 Text feature [extracellular] present in test data point [True]
33 Text feature [serum] present in test data point [True]
36 Text feature [kinases] present in test data point [True]
38 Text feature [proliferation] present in test data point [True]
41 Text feature [inhibition] present in test data point [True]
42 Text feature [stability] present in test data point [True]
43 Text feature [treated] present in test data point [True]
45 Text feature [expressing] present in test data point [True]
49 Text feature [patients] present in test data point [True]
50 Text feature [activate] present in test data point [True]
51 Text feature [cells] present in test data point [True]
52 Text feature [amplification] present in test data point [True]
54 Text feature [mitogen] present in test data point [True]
57 Text feature [protein] present in test data point [True]
58 Text feature [functional] present in test data point [True]
59 Text feature [lines] present in test data point [True]
60 Text feature [cell] present in test data point [True]
66 Text feature [factor] present in test data point [True]
71 Text feature [retained] present in test data point [True]
72 Text feature [proteins] present in test data point [True]
73 Text feature [3t3] present in test data point [True]
74 Text feature [effective] present in test data point [True]
77 Text feature [lung] present in test data point [True]
78 Text feature [phosphorylated] present in test data point [True]
83 Text feature [phosphatase] present in test data point [True]
84 Text feature [response] present in test data point [True]
85 Text feature [harboring] present in test data point [True]
87 Text feature [predicted] present in test data point [True]
90 Text feature [splice] present in test data point [True]
92 Text feature [resistant] present in test data point [True]
93 Text feature [ras] present in test data point [True]
94 Text feature [oncogene] present in test data point [True]
95 Text feature [variant] present in test data point [True]
Out of the top 100 features 56 are present in query point
```

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

In [200]:

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



```

m_state=False,
# class_weight=None)

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

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

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

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

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y_train, 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 : 2.0675584337507495
for n_estimators = 10 and max depth = 3
Log Loss : 1.6938777946898587
for n_estimators = 10 and max depth = 5
Log Loss : 1.6244226603373448
for n_estimators = 10 and max depth = 10
Log Loss : 1.5136705760148659
for n_estimators = 50 and max depth = 2
Log Loss : 1.695669640779793
for n_estimators = 50 and max depth = 3
Log Loss : 1.387052379371738
for n_estimators = 50 and max depth = 5
Log Loss : 1.4739279829314453
for n_estimators = 50 and max depth = 10
Log Loss : 1.6820322705067516
for n_estimators = 100 and max depth = 2
Log Loss : 1.5130647195232825
for n_estimators = 100 and max depth = 3
Log Loss : 1.4596845337504605
for n_estimators = 100 and max depth = 5
Log Loss : 1.338599727999531
for n_estimators = 100 and max depth = 10
Log Loss : 1.6854626665564127
for n_estimators = 200 and max depth = 2
Log Loss : 1.6012159006767979
for n_estimators = 200 and max depth = 3
Log Loss : 1.5028501651898674
for n_estimators = 200 and max depth = 5
Log Loss : 1.4027180767712204
for n_estimators = 200 and max depth = 10
Log Loss : 1.7321055705353572
for n_estimators = 500 and max depth = 2
Log Loss : 1.7100512440083484
for n_estimators = 500 and max depth = 3
Log Loss : 1.578816086040387
for n_estimators = 500 and max depth = 5
Log Loss : 1.381780319829746
for n_estimators = 500 and max depth = 10
Log Loss : 1.7826627898365277
for n_estimators = 1000 and max depth = 2
Log Loss : 1.6649221946648411
for n_estimators = 1000 and max depth = 3
Log Loss : 1.5740293241640144
for n_estimators = 1000 and max depth = 5
Log Loss : 1.3688587276367348
for n_estimators = 1000 and max depth = 10
Log Loss : 1.80633892451625
For values of best alpha = 100 The train log loss is: 0.060906836234414896
For values of best alpha = 100 The cross validation log loss is: 1.338599727999531
For values of best alpha = 100 The test log loss is: 1.3834770558599507

```

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

In [201]:

```

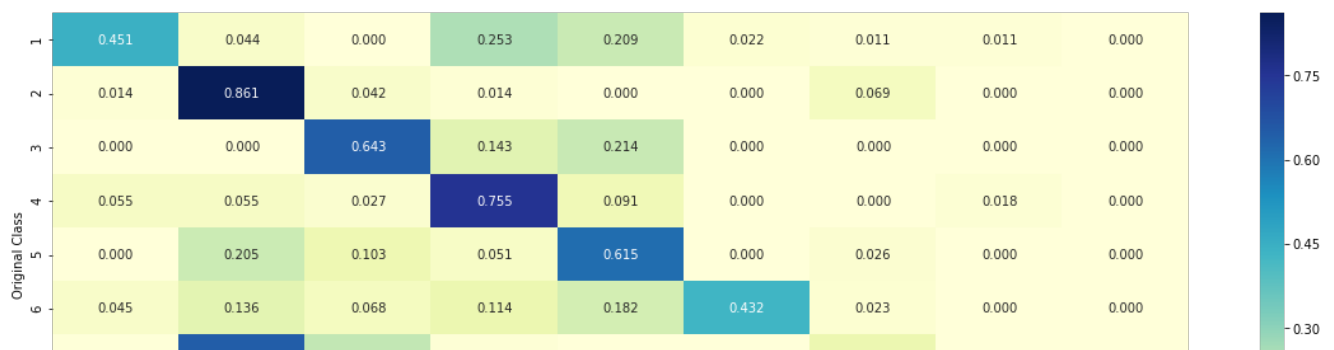
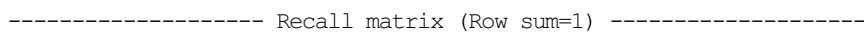
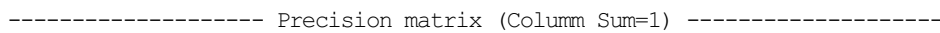
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True,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

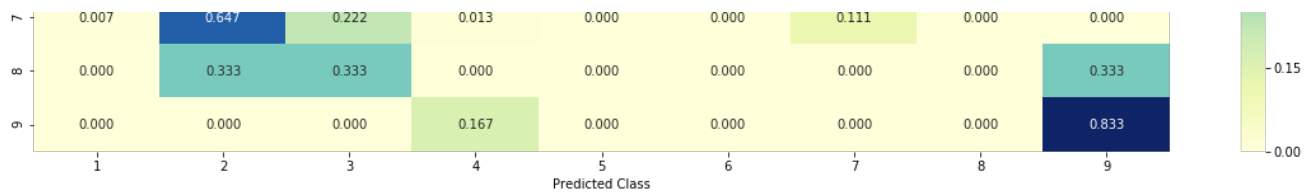
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

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

```

```
Log loss : 1.338599727999531
Number of mis-classified points : 0.5112781954887218
----- Confusion matrix -----
```





## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point

In [202]:

```
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 : 2
Predicted Class Probabilities: [[0.0355 0.2758 0.229 0.0462 0.0281 0.0522 0.2624 0.053 0.0177]]
Actual Class : 2
```

```
-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

### 4.5.5.2. Incorrectly Classified point

#### 4.3.3.2. Incorrectly Classified point

In [203]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_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 : 2

Predicted Class Probabilities: [[0.0175 0.5357 0.103 0.0219 0.0239 0.0348 0.2299 0.0258 0.0076]]

Actual Class : 7

```
-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

## 4.7 Stack the models

### 4.7.1 testing with hyper parameter tuning

In [204]:

```
# 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=N
one, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-----
```

```

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

# read more about support vector machines with linear kernels 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, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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

# read more about support vector machines with linear kernels 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_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

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

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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)

```

```

sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)
print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.p
redict_proba(cv_x_onehotCoding))))
log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
if best_alpha > log_error:
    best_alpha = log_error

```

Logistic Regression : Log Loss: 0.93  
Support vector machines : Log Loss: 1.45  
Naive Bayes : Log Loss: 1.19

-----  
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.177  
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.030  
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.470  
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.014  
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.044  
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.289

## 4.7.2 testing the model with the best hyper parameters

In [205]:

```

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

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

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

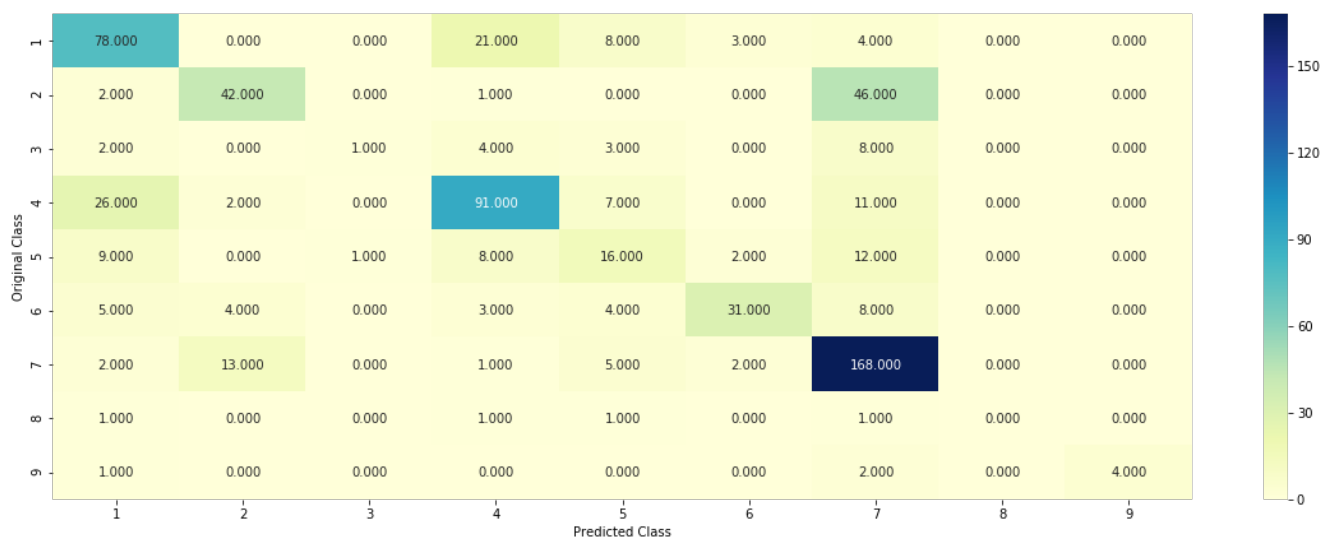
log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

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

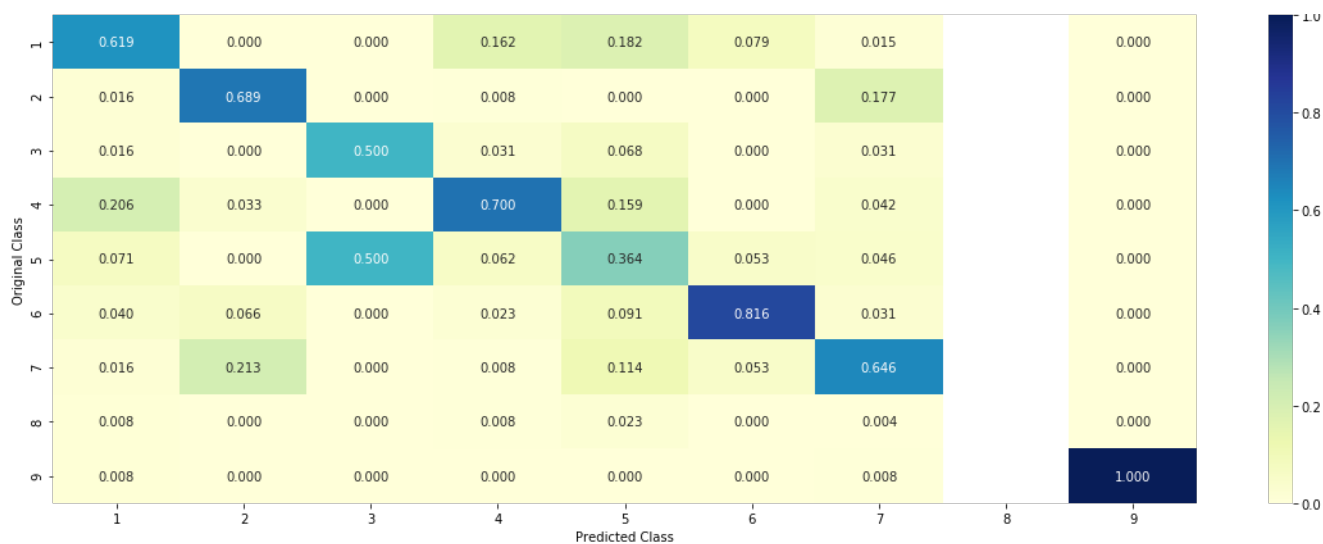
```

Log loss (train) on the stacking classifier : 0.6142670446584532  
Log loss (CV) on the stacking classifier : 1.0143639250886207  
Log loss (test) on the stacking classifier : 1.1171554340124765  
Number of missclassified point : 0.3518796992481203

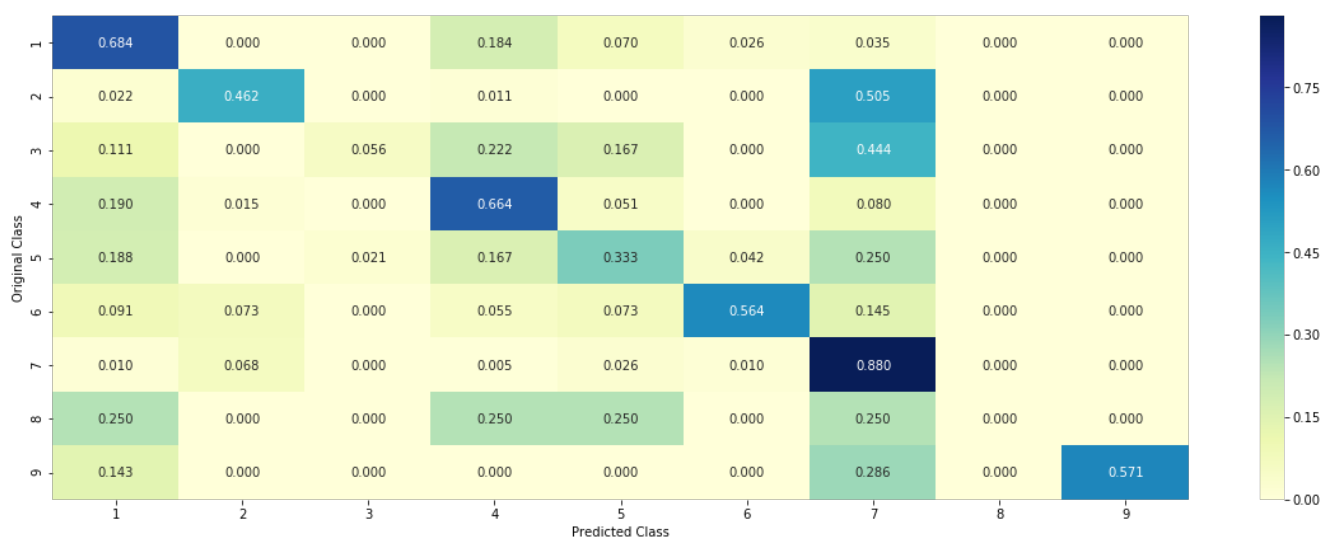
----- Confusion matrix -----



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



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



## 4.7.3 Maximum Voting classifier

In [206]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vc1f = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vc1f.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vc1f.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vc1f.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vc1f.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vc1f.predict(test_x_onehotCoding) - test_y) / test_y.shape[0]))
plot_confusion_matrix(test_y=test_y, predict_y=vc1f.predict(test_x_onehotCoding))
```

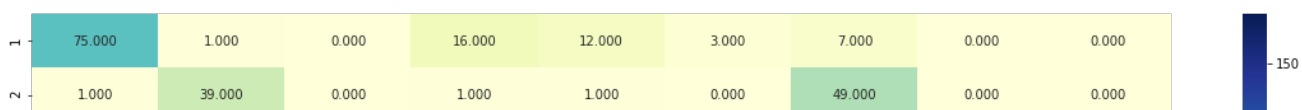
Log loss (train) on the VotingClassifier : 0.8488224530523184

Log loss (CV) on the VotingClassifier : 1.0628974227418848

Log loss (test) on the VotingClassifier : 1.173371591550275

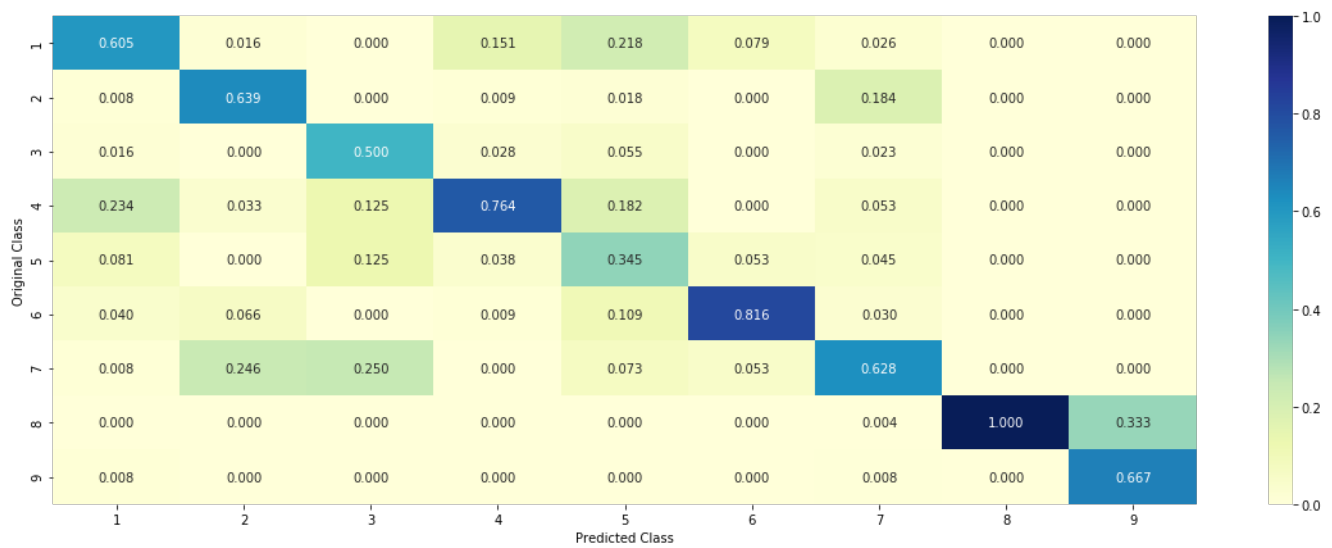
Number of missclassified point : 0.3669172932330827

----- Confusion matrix -----

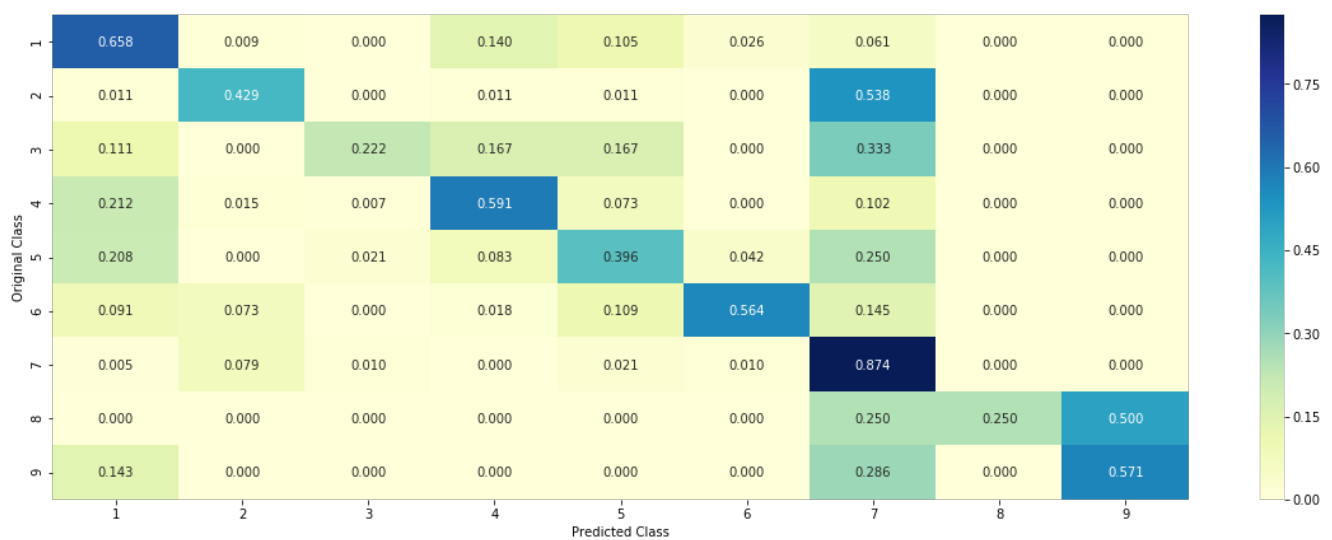




Precision matrix (Column Sum=1)



Recall matrix (Row sum=1)



In [222]:

```
#TASK 3 #Logistic Regression with Class Balancing(Count Vectorizer unigrams and bigrams)
```

In [210]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data and with unigrams and bigrams
text_vectorizer_2 = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding_2 = text_vectorizer_2.fit_transform(train_df['TEXT'])
```



```

train_text_feature_onehotCoding_2 = text_vectorizer_2.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features_2 = text_vectorizer_2.get_feature_names()

# train_text_feature_onehotCoding_2.sum(axis=0).A1 will sum every row and returns (1*number of features)
vector
train_text_fea_counts_2 = train_text_feature_onehotCoding_2.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occurred
text_fea_dict_2 = dict(zip(list(train_text_features_2),train_text_fea_counts_2))

print("Total number of unique words in train data :", len(train_text_features_2))

```

Total number of unique words in train data : 801863

In [214]:

```

dict_list = []
# dict_list=[] contains 9 dictionaries 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 build on whole training text data
total_dict = extract_dictionary_paddle(train_df)

confuse_array = []
for i in train_text_features_2:
    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 [215]:

```

# don't forget to normalize every feature
train_text_feature_onehotCoding_2 = normalize(train_text_feature_onehotCoding_2, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding_2 = text_vectorizer_2.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_onehotCoding_2 = normalize(test_text_feature_onehotCoding_2, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding_2 = text_vectorizer_2.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding_2 = normalize(cv_text_feature_onehotCoding_2, axis=0)

```

In [216]:

```

#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 [217]:

```

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_2)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding_2)).tocsr()

```

```
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding_2)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding_2)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [218]:

```
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, 804060)
(number of data points * number of features) in test data = (665, 804060)
(number of data points * number of features) in cross validation data = (532, 804060)
```

In [219]:

```
# 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/
#-----

# 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 probabilities 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.title('Cross Validation Error for each alpha')
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

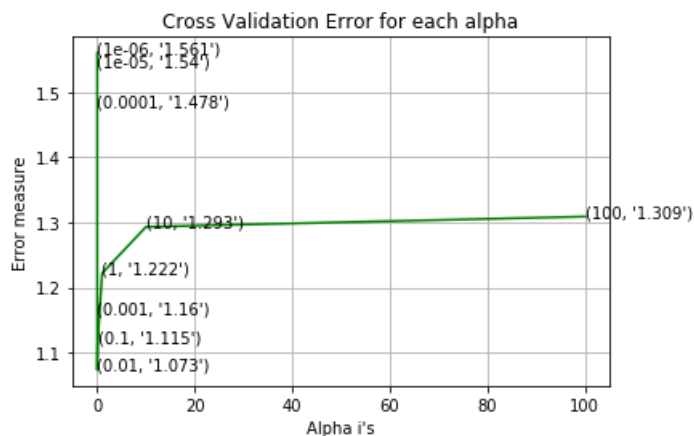
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, 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 : 1.5606839693203471
for alpha = 1e-05
Log Loss : 1.5400430046448301
for alpha = 0.0001
Log Loss : 1.4779039591507768
for alpha = 0.001
Log Loss : 1.159808653282564
for alpha = 0.01
Log Loss : 1.0733769173639796
for alpha = 0.1
Log Loss : 1.115281381169979
for alpha = 1
Log Loss : 1.2217755562497732
for alpha = 10
Log Loss : 1.2931007825540082
for alpha = 100
Log Loss : 1.3088964857905945

```



```

For values of best alpha = 0.01 The train log loss is: 0.7211246394286942
For values of best alpha = 0.01 The cross validation log loss is: 1.0733769173639796
For values of best alpha = 0.01 The test log loss is: 1.2002921203795176

```

In [223]:

```

# 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)

```

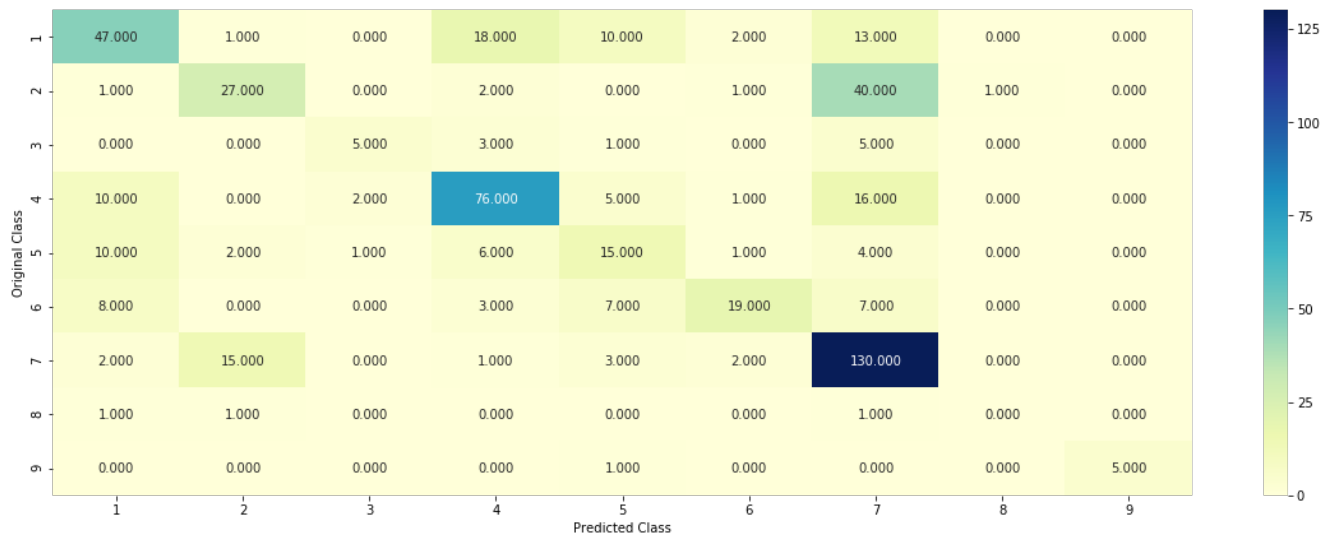
```
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

#-----
# 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)
```

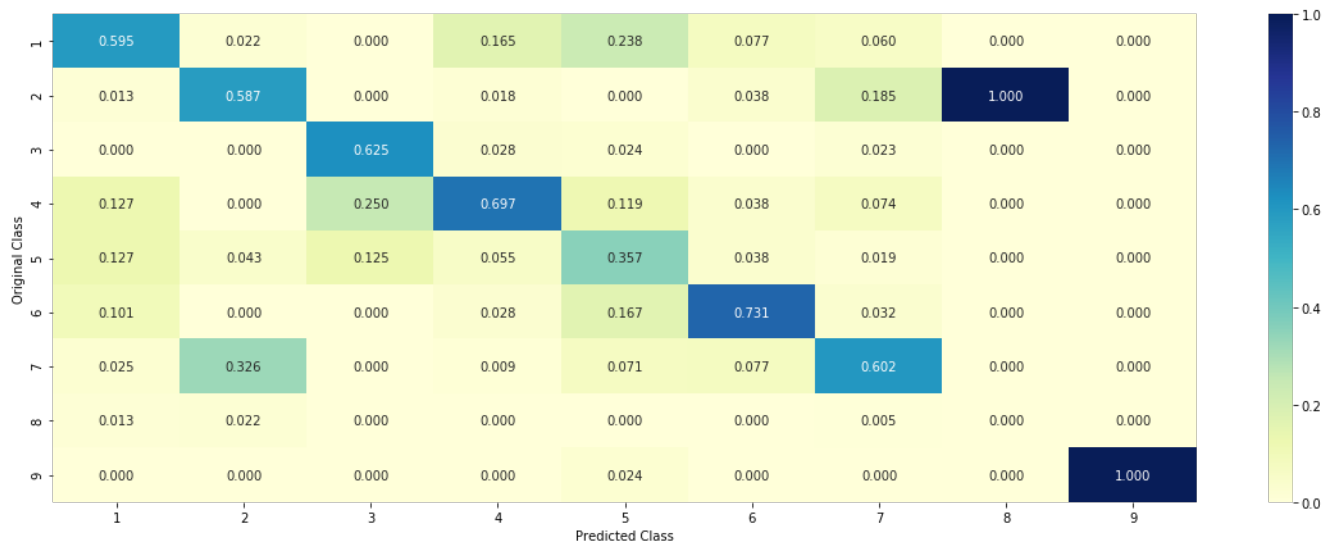
Log loss : 1.0733769173639796

Number of mis-classified points : 0.39097744360902253

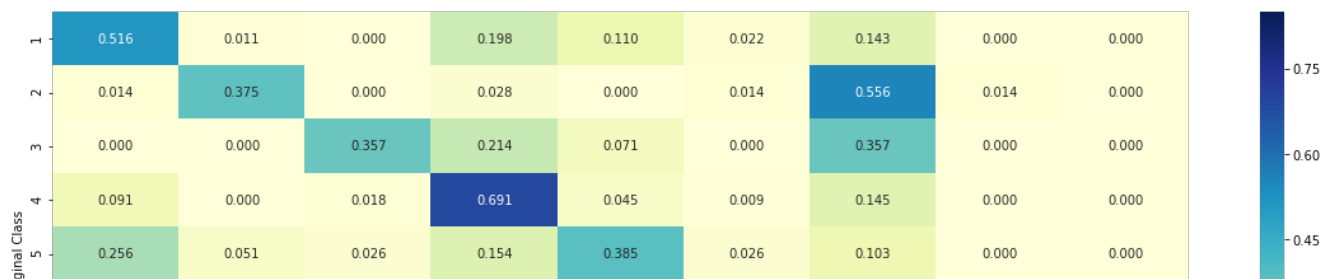
----- Confusion matrix -----

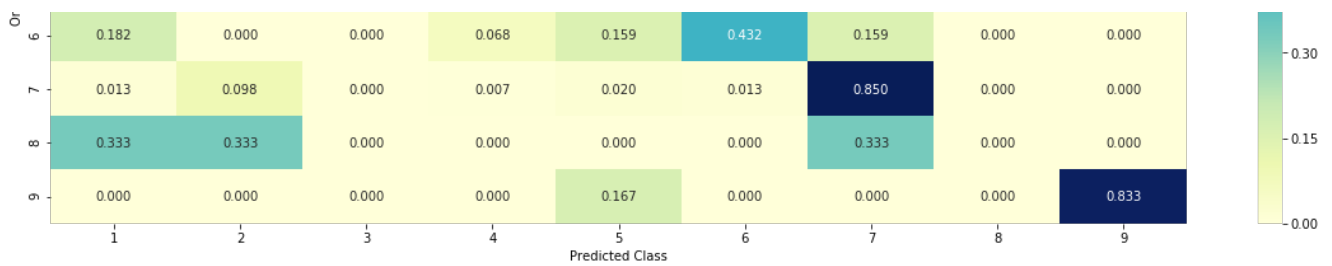


----- Precision matrix (Column Sum=1) -----



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





In [4]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Train loss", "CV Loss", "Test Loss", "%Misclassified"]

x.add_row(["Naive Bayes", "0.92", "1.14", "1.24", "39%"])
x.add_row(["KNN", "0.49", "0.917", "1.075", "31%"])
x.add_row(["LR with Class Balancing", "0.538", "0.929", "1.07", "32%"])
x.add_row(["LR without Class Balancing", "0.534", "0.943", "1.09", "32%"])
x.add_row(["Linear SVM", "0.59", "1.011", "1.14", "33%"])
x.add_row(["Random Forest(One hot encoding)", "0.65", "1.13", "1.17", "37%"])
x.add_row(["Random Forest(Response encoding)", "0.06", "1.33", "1.38", "51%"])
x.add_row(["LR with CountVectorizer(unigrams/bigrams)", "0.72", "1.07", "1.2", "39%"])

print(x)
```

Model	Train loss	CV Loss	Test Loss	%Misclassified
Naive Bayes	0.92	1.14	1.24	39%
KNN	0.49	0.917	1.075	31%
LR with Class Balancing	0.538	0.929	1.07	32%
LR without Class Balancing	0.534	0.943	1.09	32%
Linear SVM	0.59	1.011	1.14	33%
Random Forest(One hot encoding)	0.65	1.13	1.17	37%
Random Forest(Response encoding)	0.06	1.33	1.38	51%
LR with CountVectorizer(unigrams/bigrams)	0.72	1.07	1.2	39%

Observation: Task 1 and Task 2: Applied TFIDF vectorizer and 1000 best features are selected using SelectKBest method. TASK 3: Logistic Regression with Class Balancing(Count Vectorizer unigrams and bigrams)

Comparing all the models, the log loss value is reduced when model is trained on TFIDF/LR with Class balancing.