

# 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 [4]:

```
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.preprocessing import MinMaxScaler
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
import nltk
```

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

In [6]:

```
data = pd.read_csv('training_variants.csv')
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	ID	Gene	Variation	Class
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

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

Fields are

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

### 3.1.2. Reading Text Data

In [8]:

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

Number of data points : 3321

Number of features : 2

Features : ['ID' 'TEXT']

Out[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...

In [9]:

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

Out[9]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

+++++

## Task 3: Feature Engineering

+++++

Research & Ideas obtained from these kernels/blogs.

1. <https://www.kaggle.com/osciart/redefining-treatment-0-57456-modified>
2. <https://www.kaggle.com/lalitparihar44/detailed-text-based-feature-engineering>
3. <https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/>

### Gene + Variation Feature

In [12]:

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

Out[12]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

In [13]:

```
result['Gene_Variation'] = result['Gene'] + " " + result["Variation"]
result.head()
```

Out[13]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V

In [14]:

```
result.loc[result['TEXT'].isnull(), 'TEXT'] = result['Gene'] + ' '+result['Variation']
result[result['ID']==1109]
```

Out[14]:

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

### Gene Count Feature

In [15]:

```
result['Gene_Share'] = result.apply(lambda r: sum([1 for w in r['Gene'].split() if w in r['TEXT'].split()]), axis=1)
result.head()
```

Out[15]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1

## Variation Count Feature

In [16]:

```
result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['Variation'].split(' ') if w in r['TEXT'].split(' ')]), axis=1)
result["Variation_Share"].value_counts()
```

Out[16]:

```
1    1676
0    1572
2      59
3      10
5        2
4         2
```

Name: Variation\_Share, dtype: int64

## Count of Words Feature

In [17]:

```
result["Word_Count"] = result["TEXT"].apply(lambda x: len(x.split()))
result.head()
```

Out[17]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5722
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5722
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572
					Oncoegenic mutations in				

4	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count
					B...				

Text Count > 5000 Yes or no feature

```
In [18]:  
  
result["Word_Count_5000"] = result["Word_Count"].apply(lambda x: 1 if x > 5000 else 0)  
result.head()  
  
Out[18]:
```

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5722	1
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5722	1
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1

Character Count Feature

```
In [19]:  
  
result['Character_Count'] = result['TEXT'].apply(lambda x: len(str(x)))  
result.head()  
  
Out[19]:
```

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1
1	1	CBL	W802*	2	Abstract Background Non-small	CBL W802*	1	1	5722	1

	ID	Gene	Variation	Class	Non-small cell lung TEXT	CBL W802 Gene_Variation	Gene_Share	Variation_Share	5722 Word_Count	Word_Count_5
					canc...					
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5722	1
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1

### Average Length of Words used in statements

In [20]:

```
result['Avg_length'] = result['Character_Count'] / result['Word_Count']
result.head()
```

Out[20]:

	ID	Gene	Variation	Class	TEXT	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5
0	0	FAM58A	Truncating Mutations	1	Cyclin- dependent kinases (CDKs) regulate a var...	FAM58A Truncating Mutations	1	1	6089	1
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	CBL W802*	1	1	5722	1
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...	CBL Q249E	1	1	5722	1
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...	CBL N454D	1	1	5572	1
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...	CBL L399V	1	1	6202	1

### 3.1.4. Preprocessing of text

In [21]:



In [21]:

```
# 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 [23]:

```
#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 : 145.4743300371905 seconds

In [24]:

```
#removing unprocessed "TEXT" from results
result.drop("TEXT", axis=1, inplace=True)

# Joining Text which is processed :
result = pd.merge(result, data_text,on='ID', how='left')
result.head()
```

Out[24]:

	ID	Gene	Variation	Class	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Character
0	0	FAM58A	Truncating Mutations	1	FAM58A Truncating Mutations	1	1	6089	1	39765
1	1	CBL	W802*	2	CBL W802*	1	1	5722	1	36831
2	2	CBL	Q249E	2	CBL Q249E	1	1	5722	1	36831
3	3	CBL	N454D	3	CBL N454D	1	1	5572	1	36308

	ID	Gene	Variation	Class	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Character
4	4	CBL	L399V	4	CBL L399V	1	1	6202	1	41427

In [25]:

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

Out[25]:

	ID	Gene	Variation	Class	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Character
1109	1109	FANCA	S1088F	1	FANCA S1088F	1	1	2	0	12
1277	1277	ARID5B	Truncating Mutations	1	ARID5B Truncating Mutations	1	2	3	0	27
1407	1407	FGFR3	K508M	6	FGFR3 K508M	1	1	2	0	17
1639	1639	FLT1	Amplification	6	FLT1 Amplification	1	1	2	0	18
2755	2755	BRAF	G596C	7	BRAF G596C	1	1	2	0	10

In [26]:

```
result.loc[result['TEXT'].isnull(), 'TEXT'] = result['Gene'] + ' ' + result['Variation']
result[result['ID']==1109]
```

Out[26]:

	ID	Gene	Variation	Class	Gene_Variation	Gene_Share	Variation_Share	Word_Count	Word_Count_5000	Character
1109	1109	FANCA	S1088F	1	FANCA S1088F	1	1	2	0	12

### 3.1.5. Test, Train and Cross Validation Split

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

In [27]:

```
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
result.Gene_Variation = result.Gene_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,
, random_state = 1)
# 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,
, random_state = 1)
```

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

In [28]:

```
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

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

In [30]:

```
# 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()

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()

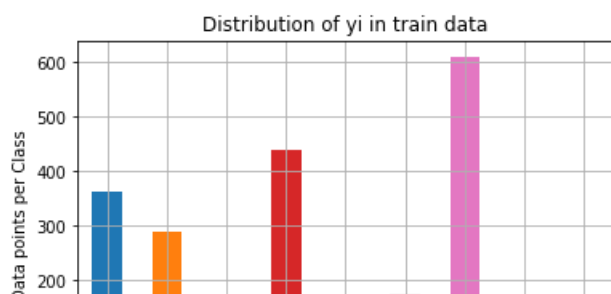
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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

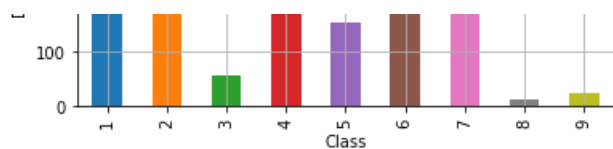
print('-'*80)
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of  $y_i$  in test data')
plt.grid()
plt.show()

# 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.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

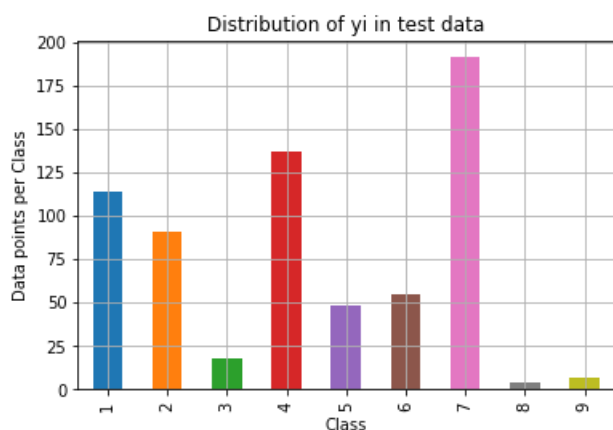
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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





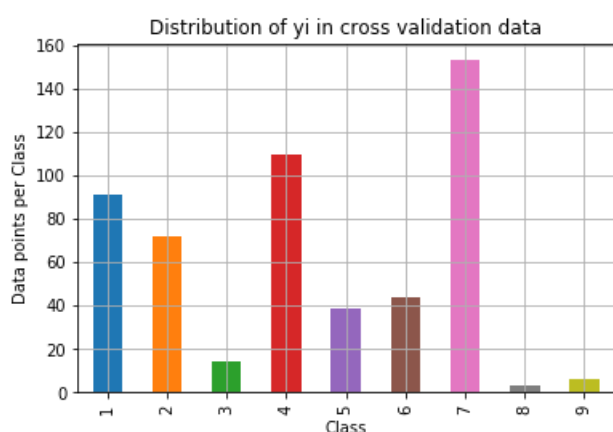
Number of data points in class 7 : 609 ( 28.672 %)  
 Number of data points in class 4 : 439 ( 20.669 %)  
 Number of data points in class 1 : 363 ( 17.09 %)  
 Number of data points in class 2 : 289 ( 13.606 %)  
 Number of data points in class 6 : 176 ( 8.286 %)  
 Number of data points in class 5 : 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 [31]:

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

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

    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 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))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #      [3, 4]]
    # C.sum(axis = 0)  axis=0 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 [32]:

```
# 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))

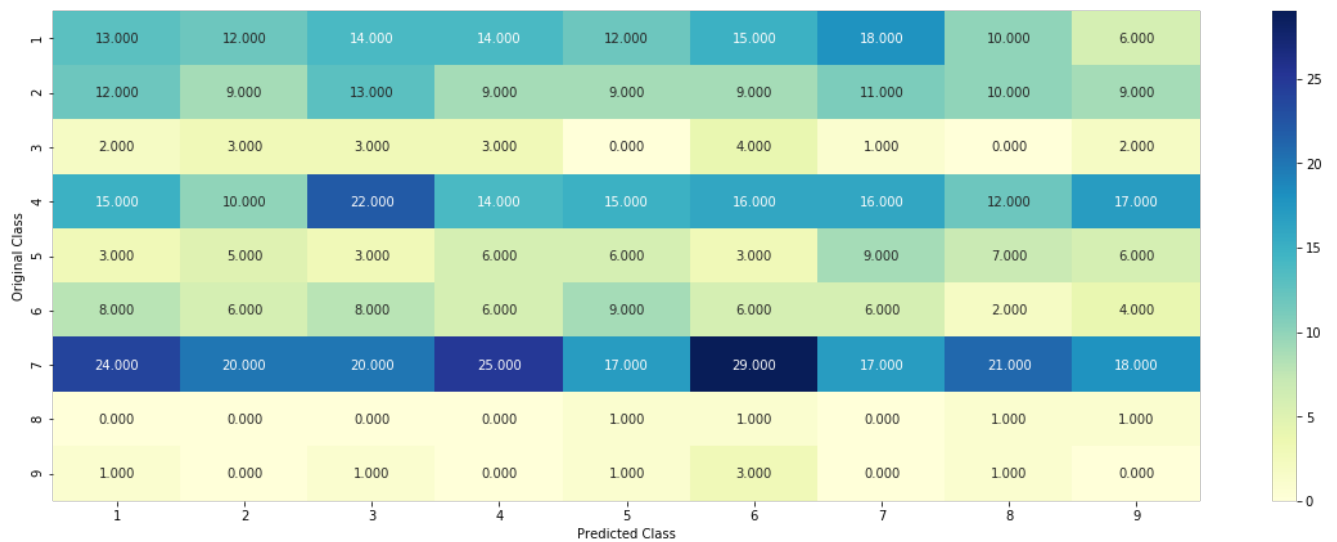
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

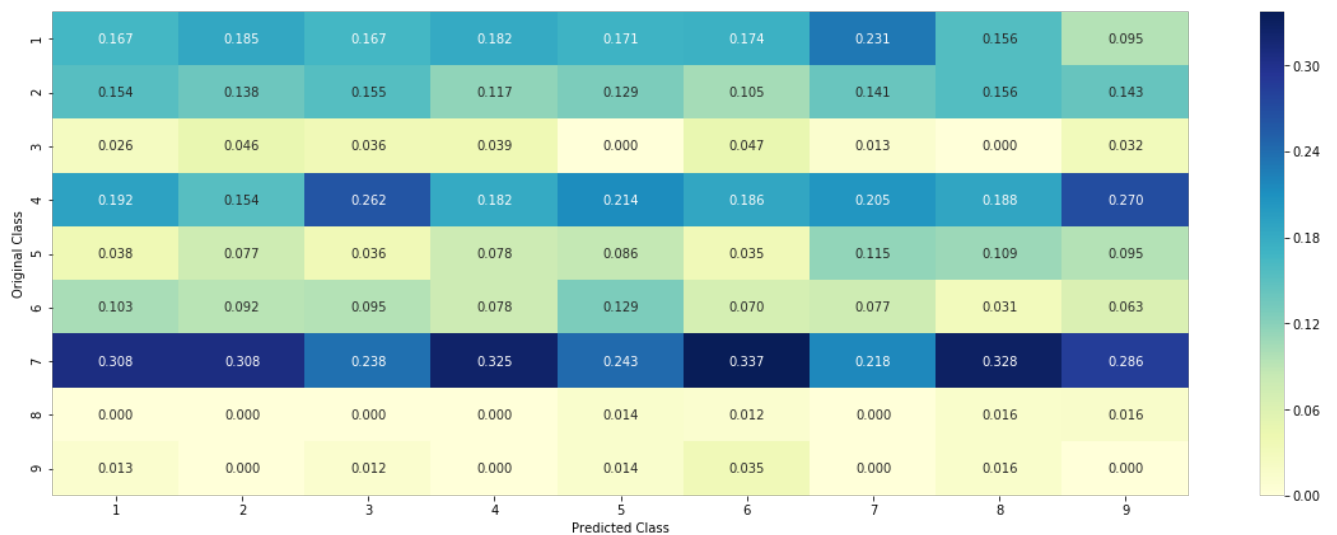
Log loss on Cross Validation Data using Random Model 2.502853362401248

Log loss on Test Data using Random Model 2.4790889720578173

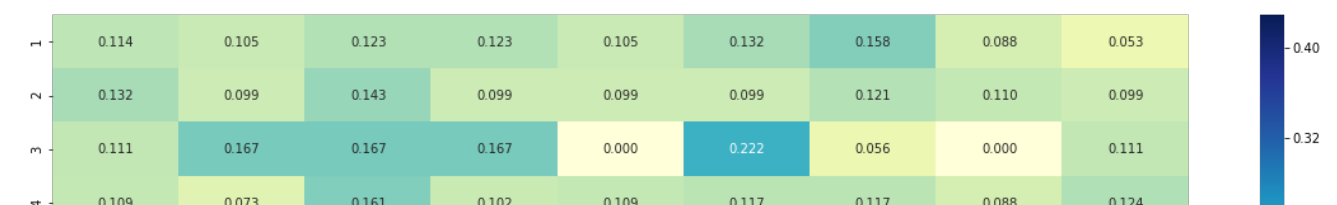
----- Confusion matrix -----

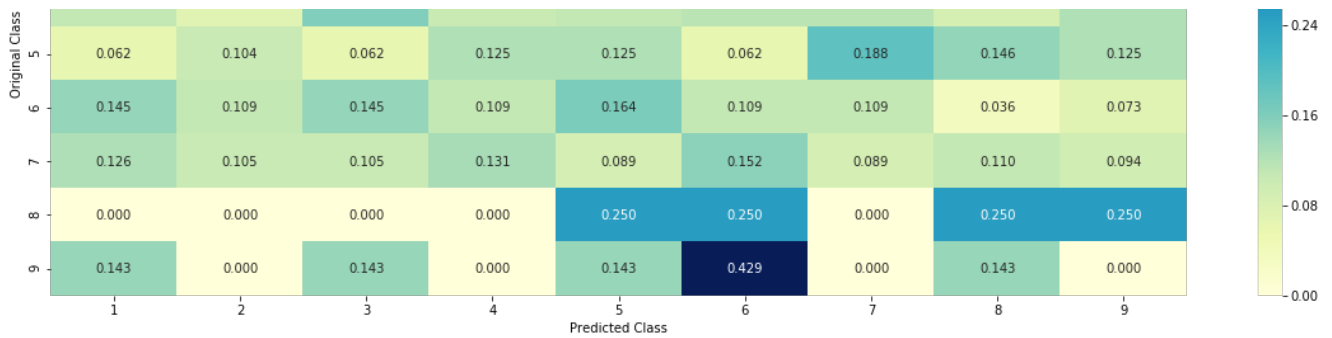


----- Precision matrix (Column Sum=1) -----



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





### 3.3 Univariate Analysis

In [33]:

```
# 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 times 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 times 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      M1R        1
    # 2432    2432    BRCA1      L1657P      1
    # 2567    2567    BRCA1      T1685A      1
    # 2583    2583    BRCA1      E1660G      1
    # 2634    2634    BRCA1      W1718L      1
    # cls_cnt.shape[0] will return the number of rows

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

    # cls_cnt.shape[0](numerator) will contain the number of time that particular feature 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, 0.0681818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0625, 0.0681818181818177, 0.0681818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.139393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],
    # ...
    # }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

### 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 [34]:

```
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 : 235

BRCA1 168

TP53 111

EGFR 86

PTEN 86

BRCA2 80

BRAF 65

KIT 60

ALK 44

ERBB2 39

PDGFRA 38

Name: Gene, dtype: int64

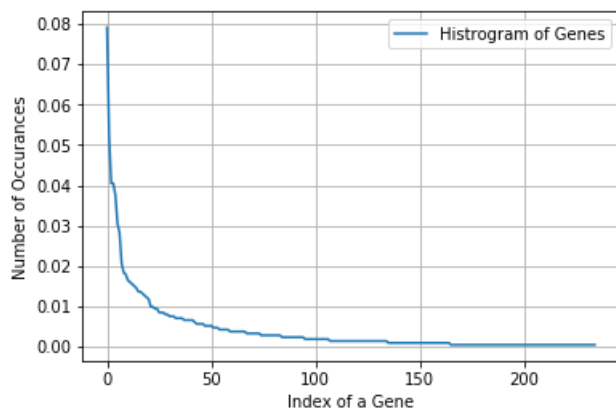
In [35]:

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

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

In [36]:

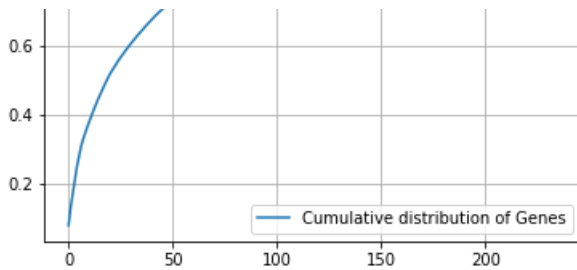
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [37]:

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





### Q3. How to featurize this Gene feature ?

**Ans.** there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [38]:

```
#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 [39]:

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

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

In [40]:

```
# 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 [41]:

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

Out[41]:

```
364      EPAS1
927      PDGFRA
2353     AURKA
2644     BRCA1
489      TP53
Name: Gene, dtype: object
```

In [42]:

```
gene_vectorizer.get_feature_names()
```

Out[42]:

```
['abl1',  
 'acvr1',  
 'ago2',  
 'akt1',  
 'akt2',  
 'akt3',  
 'alk',  
 'apc',  
 'ar',  
 'araf',  
 'arid1a',  
 'arid1b',  
 'arid2',  
 'arid5b',  
 'asx11',  
 'asx12',  
 'atm',  
 'atr',  
 'atrx',  
 'aurka',  
 'axin1',  
 'axl',  
 'b2m',  
 'bap1',  
 'bard1',  
 'bcl10',  
 'bcl2l11',  
 'bcor',  
 'braf',  
 'brca1',  
 'brca2',  
 'brd4',  
 'brip1',  
 'btk',  
 'card11',  
 'carm1',  
 'casp8',  
 'cbl',  
 'ccnd1',  
 'ccnd2',  
 'ccnd3',  
 'ccne1',  
 'cdh1',  
 'cdk12',  
 'cdk4',  
 'cdk6',  
 'cdkn1a',  
 'cdkn1b',  
 'cdkn2a',  
 'cdkn2b',  
 'cdkn2c',  
 'cebpa',  
 'chek2',  
 'cic',  
 'crebbp',  
 'ctcf',  
 'ctla4',  
 'ctnnb1',  
 'ddr2',  
 'dicer1',  
 'dnmt3a',  
 'dnmt3b',  
 'egfr',  
 'elf3',  
 'ep300',  
 'epas1',  
 'epcam',  
 'erbb2',  
 'erbb3',  
 'erbb4',  
 'ercc2',  
 'ercc3',  
 'ercc4',  
 'erg',  
 'errfil',  
 'esr1',  
 'etv1']
```

'etv6',  
'ewsr1',  
'ezh2',  
'fanca',  
'fat1',  
'fbxw7',  
'fgf4',  
'fgfr1',  
'fgfr2',  
'fgfr3',  
'fgfr4',  
'flt1',  
'flt3',  
'foxa1',  
'foxl2',  
'foxp1',  
'fubp1',  
'gata3',  
'gli1',  
'gnaq',  
'gnas',  
'h3f3a',  
'hist1h1c',  
'hla',  
'hras',  
'idh1',  
'idh2',  
'igflr',  
'ikbke',  
'ikzf1',  
'jak1',  
'jak2',  
'jun',  
'kdm5c',  
'kdm6a',  
'kdr',  
'keap1',  
'kit',  
'kmt2a',  
'kmt2b',  
'kmt2c',  
'kmt2d',  
'knstrn',  
'kras',  
'lats2',  
'map2k1',  
'map2k2',  
'map2k4',  
'map3k1',  
'mapk1',  
'mdm2',  
'mdm4',  
'med12',  
'mef2b',  
'men1',  
'met',  
'mlh1',  
'mpl',  
'msh2',  
'msh6',  
'mtor',  
'myc',  
'mycn',  
'myd88',  
'ncor1',  
'nfl1',  
'nf2',  
'nfe2l2',  
'nfkb1a',  
'nkx2',  
'notch1',  
'notch2',  
'nras',  
'ntrk1',  
'ntrk2',  
'ntrk3',  
'ntrk3'.

'pax8',  
'pbrml',  
'pdgfra',  
'pdgfrb',  
'pik3ca',  
'pik3cb',  
'pik3cd',  
'pik3r1',  
'pik3r2',  
'pik3r3',  
'pim1',  
'pms1',  
'pms2',  
'pole',  
'ppm1d',  
'ppp2r1a',  
'ppp6c',  
'prdm1',  
'ptch1',  
'pten',  
'ptpn11',  
'ptprd',  
'ptprt',  
'rab35',  
'rac1',  
'rad21',  
'rad50',  
'rad51b',  
'rad51d',  
'raf1',  
'rara',  
'rasa1',  
'rb1',  
'rbm10',  
'ret',  
'rheb',  
'rhoa',  
'riCTOR',  
'rit1',  
'rnf43',  
'ros1',  
'rras2',  
'runx1',  
'rybp',  
'sdhb',  
'sdhc',  
'setd2',  
'sf3b1',  
'shoc2',  
'smad2',  
'smad3',  
'smad4',  
'smarca4',  
'smarcb1',  
'smo',  
'sox9',  
'spop',  
'src',  
'stag2',  
'stat3',  
'stk11',  
'tcf7l2',  
'tert',  
'tet1',  
'tet2',  
'tgfbr1',  
'tgfbr2',  
'tmprss2',  
'tp53',  
'tp53bp1',  
'tsc1',  
'tsc2',  
'u2af1',  
'vegfa',  
'vhl',  
'whsc1'

```

whsc1',
'whsc111',
'xpol',
'xrcc2',
'yap1']

```

In [43]:

```

# creating a pandas dataframe of the vectorized features
df_gene_train = pd.DataFrame(train_gene_feature_onehotCoding.toarray(), columns=gene_vectorizer.get_feature_names())
df_gene_test = pd.DataFrame(test_gene_feature_onehotCoding.toarray(), columns=gene_vectorizer.get_feature_names())
df_gene_cv = pd.DataFrame(cv_gene_feature_onehotCoding.toarray(), columns=gene_vectorizer.get_feature_names())

```

In [44]:

```

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, 235)

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

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

In [45]:

```

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, 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()

```

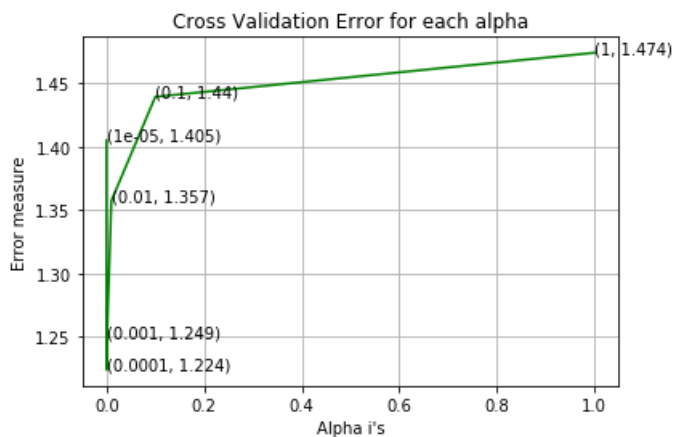
```

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

```

For values of alpha = 1e-05 The log loss is: 1.405159606313378  
 For values of alpha = 0.0001 The log loss is: 1.2240333362141638  
 For values of alpha = 0.001 The log loss is: 1.2494983673910525  
 For values of alpha = 0.01 The log loss is: 1.357474535991347  
 For values of alpha = 0.1 The log loss is: 1.4395265240517152  
 For values of alpha = 1 The log loss is: 1.47433374002722



For values of best alpha = 0.0001 The train log loss is: 1.0295263434870328  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.2240333362141638  
 For values of best alpha = 0.0001 The test log loss is: 1.2024352364023003

**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 [46]:

```

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

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

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

```

Q6. How many data points in Test and CV datasets are covered by the 235 genes in train dataset?  
 Ans

1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 511 out of 532 : 96.05263157894737

### 3.2.2 Univariate Analysis on Variation Feature

**Q7. Variation, What type of feature is it ?**

**Ans.** Variation is a categorical variable

**Q8. How many categories are there?**

In [47]:

```
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 : 1920
Truncating_Mutations      58
Amplification              51
Deletion                  45
Fusions                   25
G12V                      4
Overexpression             3
G12A                      2
R173C                     2
Y64A                      2
F28L                      2
Name: Variation, dtype: int64
```

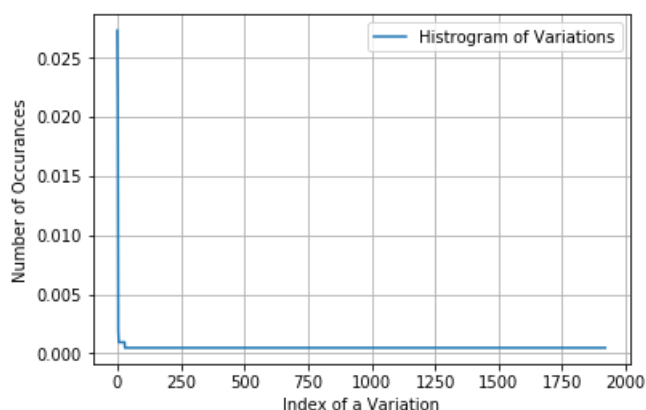
In [48]:

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

Ans: There are 1920 different categories of variations in the train data, and they are distributed as follows

In [49]:

```
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()
```

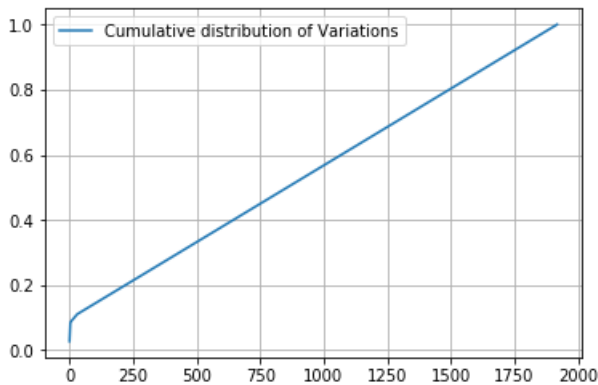


In [50]:

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

```
[0.02730697 0.05131827 0.07250471 ... 0.99905838 0.99952919 1.          ]
```





### Q9. How to featurize this Variation feature ?

**Ans.** There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [51]:

```
# 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 [52]:

```
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: (2124, 9)

In [53]:

```
# 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 [54]:

```
df_var_train = pd.DataFrame(train_variation_feature_onehotCoding.toarray(),
columns=variation_vectorizer.get_feature_names())
df_var_test = pd.DataFrame(test_variation_feature_onehotCoding.toarray(),
columns=variation_vectorizer.get_feature_names())
df_var_cv = pd.DataFrame(cv_variation_feature_onehotCoding.toarray(), columns=variation_vectorizer.get_feature_names())
```

In [55]:

```
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, 1948)

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

Let's build a model just like the earlier!

In [56]:

```
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()

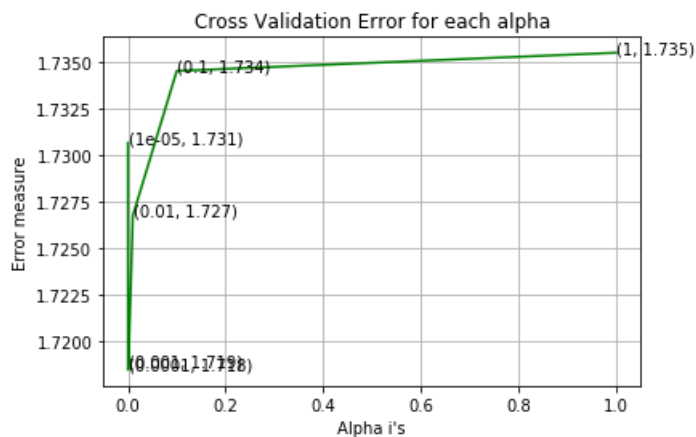
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

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: 1.7306277098519975
For values of alpha = 0.0001 The log loss is: 1.7184363468901507
For values of alpha = 0.001 The log loss is: 1.718632494779622
For values of alpha = 0.01 The log loss is: 1.726716147243816
```

For values of alpha = 0.1 The log loss is: 1.7344923121411628

For values of alpha = 1 The log loss is: 1.735468334063604



For values of best alpha = 0.0001 The train log loss is: 0.7630389018608933

For values of best alpha = 0.0001 The cross validation log loss is: 1.7184363468901507

For values of best alpha = 0.0001 The test log loss is: 1.7083423376941478

**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 [57]:

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

Q12. How many data points are covered by total 1920 genes in test and cross validation data sets?

Ans

1. In test data 61 out of 665 : 9.172932330827068

2. In cross validation data 58 out of 532 : 10.902255639097744

## Univariate Analysis on Word Count Feature

In [61]:

```
alpha = [10 ** x for x in range(-5, 1)]  
  
cv_log_error_array=[]  
for i in alpha:  
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)  
    clf.fit(np.array(train_df["Word_Count"]).reshape(-1,1), y_train)  
  
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")  
    sig_clf.fit(np.array(train_df.Word_Count).reshape(-1,1), y_train)  
    predict_y = sig_clf.predict_proba(np.array(cv_df.Word_Count).reshape(-1,1))  
  
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))  
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas  
ses_, eps=1e-15))  
  
fig, ax = plt.subplots()  
ax.plot(alpha, cv_log_error_array,c='g')  
for i, txt in enumerate(np.round(cv_log_error_array,3)):  
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))  
plt.grid()  
plt.title("Cross Validation Error for each alpha")
```

```

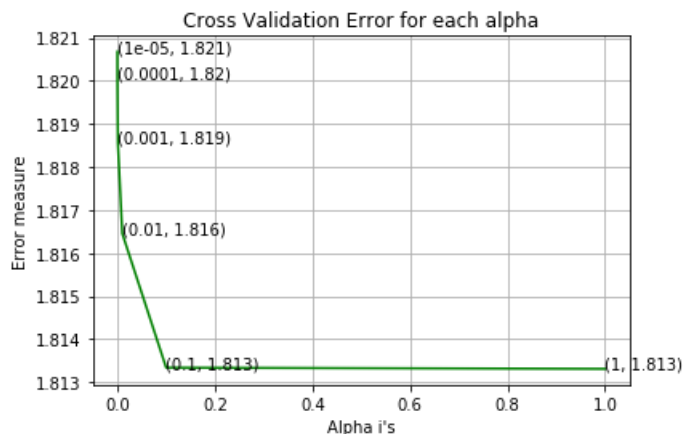
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(np.array(train_df.Word_Count).reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(np.array(train_df.Word_Count).reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(np.array(train_df.Word_Count).reshape(-1,1))
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(np.array(cv_df.Word_Count).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(np.array(test_df.Word_Count).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.8206755107070893  
 For values of alpha = 0.0001 The log loss is: 1.820057656195414  
 For values of alpha = 0.001 The log loss is: 1.8185562570606735  
 For values of alpha = 0.01 The log loss is: 1.816456894439099  
 For values of alpha = 0.1 The log loss is: 1.813341253884523  
 For values of alpha = 1 The log loss is: 1.8133131699767404



For values of best alpha = 1 The train log loss is: 1.8102652901086476  
 For values of best alpha = 1 The cross validation log loss is: 1.8133131699767404  
 For values of best alpha = 1 The test log loss is: 1.8195156975471447

## Univariate analysis on Word Count > 5000 Feature

In [63]:

```

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(np.array(train_df["Word_Count_5000"]).reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(np.array(train_df.Word_Count_5000).reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(np.array(cv_df.Word_Count_5000).reshape(-1,1))

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

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

```

```

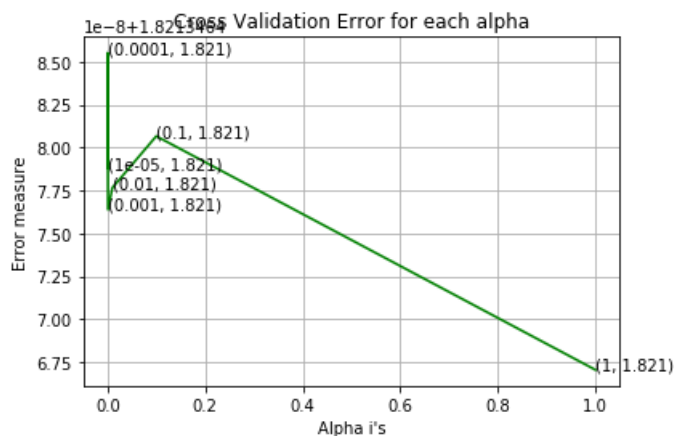
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(np.array(train_df.Word_Count_5000).reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(np.array(train_df.Word_Count_5000).reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(np.array(train_df.Word_Count_5000).reshape(-1,1))
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(np.array(cv_df.Word_Count_5000).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(np.array(test_df.Word_Count_5000).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.8213464787867863  
 For values of alpha = 0.0001 The log loss is: 1.8213464855157826  
 For values of alpha = 0.001 The log loss is: 1.8213464763804603  
 For values of alpha = 0.01 The log loss is: 1.8213464776482495  
 For values of alpha = 0.1 The log loss is: 1.8213464806566992  
 For values of alpha = 1 The log loss is: 1.8213464670427701



For values of best alpha = 1 The train log loss is: 1.8284029227711127  
 For values of best alpha = 1 The cross validation log loss is: 1.8213464670427701  
 For values of best alpha = 1 The test log loss is: 1.8284565597924245

## Univariate Analysis on Gene and Variation Feature

In [64]:

```

genevars = train_df['Gene_Variation'].value_counts()
print('Number of Unique Gene+Variations :', genevars.shape[0])
# the top 10 variations that occurred most
print(genevars.head(10))

```

```

Number of Unique Gene+Variations : 2124
EGFR_EGFRvIV      1
SMAD2_R428A       1
RET_A883F         1
MLH1_R100*        1
KIT_Y578C         1
MTOR_L2427R       1
PTEN_T131S        1
MTOR_P2476L       1
BRCA1_Q12Y        1
PIK3CD_E1021K     1
Name: Gene_Variation, dtype: int64

```

In [65]:

```
# Featurizing the Gene_and_Variation Feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_and_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene_Variation",
train_df))
# test gene feature
test_gene_and_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene_Variation", t
est_df))
# cross validation gene feature
cv_gene_and_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene_Variation", cv_
df))
```

In [66]:

```
# one-hot encoding of gene_and_variation feature.
gene_variation_vectorizer = CountVectorizer()
train_gene_and_variation_feature_onehotCoding = gene_variation_vectorizer.fit_transform(train_df["
Gene_Variation"])
test_gene_and_variation_feature_onehotCoding =
gene_variation_vectorizer.transform(test_df["Gene_Variation"])
cv_gene_and_variation_feature_onehotCoding =
gene_variation_vectorizer.transform(cv_df["Gene_Variation"])
```

In [67]:

```
df_geneandvar_train = pd.DataFrame(train_gene_and_variation_feature_onehotCoding.toarray(),
columns=gene_variation_vectorizer.get_feature_names())
df_geneandvar_test = pd.DataFrame(test_gene_and_variation_feature_onehotCoding.toarray(), columns=
gene_variation_vectorizer.get_feature_names())
df_geneandvar_cv = pd.DataFrame(cv_gene_and_variation_feature_onehotCoding.toarray(),
columns=gene_variation_vectorizer.get_feature_names())
```

In [68]:

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

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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_geneandvar_train, y_train)
    predict_y = sig_clf.predict_proba(df_geneandvar_cv)

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

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

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

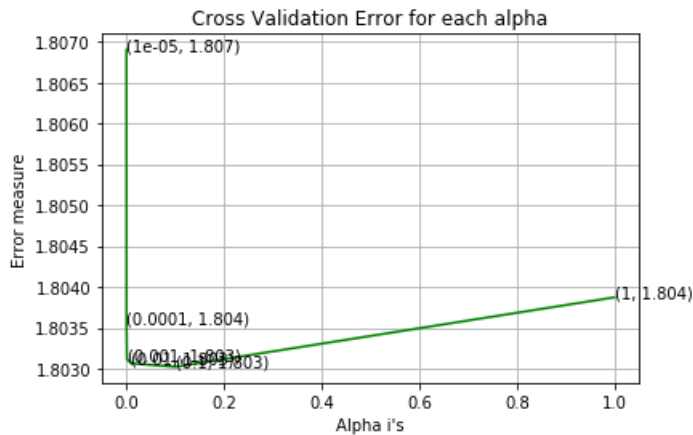
predict_y = sig_clf.predict_proba(df_geneandvar_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
```

```

predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(df_geneandvar_cv)
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(df_geneandvar_test)
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: 1.8069015059874451  
 For values of alpha = 0.0001 The log loss is: 1.803547680634202  
 For values of alpha = 0.001 The log loss is: 1.8031140730089874  
 For values of alpha = 0.01 The log loss is: 1.803064814244995  
 For values of alpha = 0.1 The log loss is: 1.8030216702611788  
 For values of alpha = 1 The log loss is: 1.8038716751978685



For values of best alpha = 0.1 The train log loss is: 0.6431800072832776  
 For values of best alpha = 0.1 The cross validation log loss is: 1.8030216702611788  
 For values of best alpha = 0.1 The test log loss is: 1.7799168034695827

## Univariate Analysis on Character Count Feature

In [70]:

```

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(np.array(train_df["Character_Count"]).reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(np.array(train_df.Character_Count).reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(np.array(cv_df.Character_Count).reshape(-1,1))

    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(np.array(train_df.Character_Count).reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(np.array(train_df.Character_Count).reshape(-1,1), y_train)

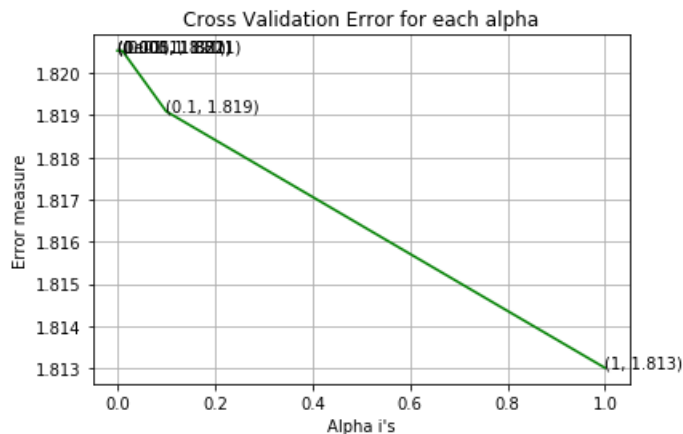
```

```

predict_y = sig_clf.predict_proba(np.array(train_df.Character_Count).reshape(-1,1))
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(np.array(cv_df.Character_Count).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(np.array(test_df.Character_Count).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.8205265318236887  
 For values of alpha = 0.0001 The log loss is: 1.8205265318266872  
 For values of alpha = 0.001 The log loss is: 1.8205265318232127  
 For values of alpha = 0.01 The log loss is: 1.8205265318235444  
 For values of alpha = 0.1 The log loss is: 1.8190905588684776  
 For values of alpha = 1 The log loss is: 1.8130107003447196



For values of best alpha = 1 The train log loss is: 1.8103242797757038  
 For values of best alpha = 1 The cross validation log loss is: 1.8130107003447196  
 For values of best alpha = 1 The test log loss is: 1.819195915055151

## Univariate Analysis on Average Length Feature

In [71]:

```

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(np.array(train_df["Avg_length"]).reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(np.array(train_df.Avg_length).reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(np.array(cv_df.Avg_length).reshape(-1,1))

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

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(np.array(train_df.Avg_length).reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

```



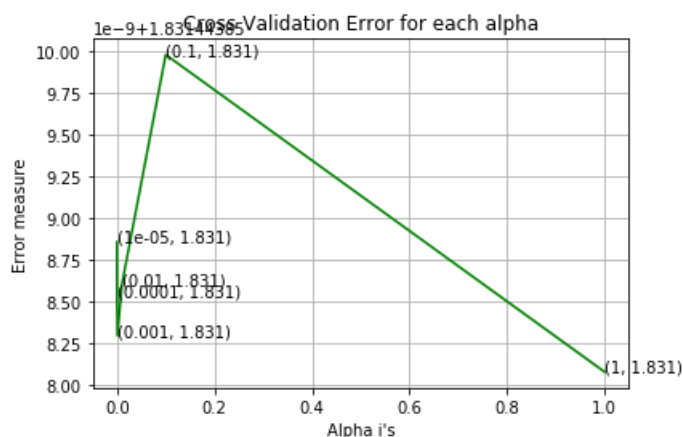
```

sig_clf.fit(np.array(train_df.Avg_length).reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(np.array(train_df.Avg_length).reshape(-1,1))
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(np.array(cv_df.Avg_length).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(np.array(test_df.Avg_length).reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.8314438588597115  
 For values of alpha = 0.0001 The log loss is: 1.8314438585376887  
 For values of alpha = 0.001 The log loss is: 1.8314438582976835  
 For values of alpha = 0.01 The log loss is: 1.8314438586048236  
 For values of alpha = 0.1 The log loss is: 1.8314438599820477  
 For values of alpha = 1 The log loss is: 1.8314438580823476



For values of best alpha = 1 The train log loss is: 1.828236237109059  
 For values of best alpha = 1 The cross validation log loss is: 1.8314438580823476  
 For values of best alpha = 1 The test log loss is: 1.8261140730051668

### 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?

In [72]:

```

# 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 [73]:

```

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 [74]:

```

# building a CountVectorizer with all the words that occurred minimum 10 times in train data with 4
-Gram
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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

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

```

Total number of unique words in train data : 5000

In [75]:

```

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 [76]:

```

#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 [77]:

```

# 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 [78]:

```
# 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 [79]:

```
#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 [80]:

```
# Train a Logistic regression+Calibration model using text features which are 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()

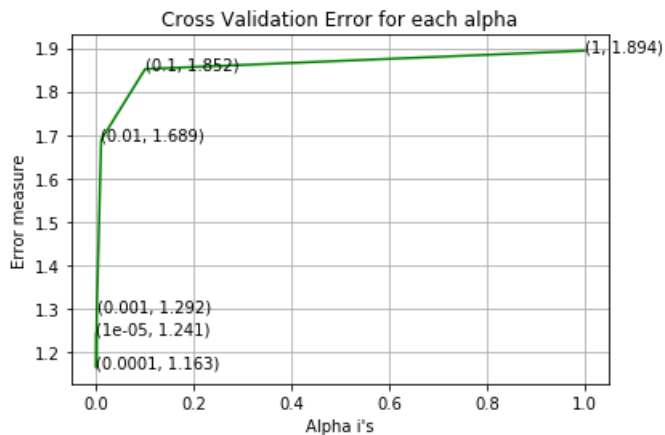
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)
predict_y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

```

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

```

For values of alpha = 1e-05 The log loss is: 1.2413927068627668  
 For values of alpha = 0.0001 The log loss is: 1.162600620019463  
 For values of alpha = 0.001 The log loss is: 1.2916591226794982  
 For values of alpha = 0.01 The log loss is: 1.6885424925838408  
 For values of alpha = 0.1 The log loss is: 1.8517752701201546  
 For values of alpha = 1 The log loss is: 1.8939354875252339



For values of best alpha = 0.0001 The train log loss is: 0.6922988992955096  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.162600620019463  
 For values of best alpha = 0.0001 The test log loss is: 1.1566521172280506

In [81]:

```

df_text_train = pd.DataFrame(train_text_feature_onehotCoding.toarray(), columns=text_vectorizer.ge
t_feature_names())
df_text_test = pd.DataFrame(test_text_feature_onehotCoding.toarray(), columns=text_vectorizer.get_
feature_names())
df_text_cv = pd.DataFrame(cv_text_feature_onehotCoding.toarray(),
columns=text_vectorizer.get_feature_names())

```

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

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

In [82]:

```

def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

```

In [83]:

```

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

```

91.64 % of word of test data appeared in train data  
90.36 % of word of Cross Validation appeared in train data

## 4. Machine Learning Models

In [84]:

```
#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 will 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 [85]:

```
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 [86]:

```
# 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=10, max_features=5000, ngram_range=(1,4))

    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:
            try:
                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))
            except:
```

```

))

    except:
        pass

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

```

## Stacking the features

In [88]:

```

#target variables
train_y = train_df['Class'].values
test_y = test_df['Class'].values
cv_y = cv_df['Class'].values

# concatenating all the vectorized dataframes
df_gene_var_train = pd.concat([df_gene_train, df_var_train], axis=1)
df_gene_var_test = pd.concat([df_gene_test, df_var_test], axis=1)
df_gene_var_cv = pd.concat([df_gene_cv, df_var_cv], axis=1)

df_gene_and_var_train = pd.concat([df_gene_var_train, df_geneandvar_train], axis=1)
df_gene_and_var_test = pd.concat([df_gene_var_test, df_geneandvar_test], axis=1)
df_gene_and_var_cv = pd.concat([df_gene_var_cv, df_geneandvar_cv], axis=1)

df_train = pd.concat([df_gene_and_var_train, df_text_train], axis=1)
df_test = pd.concat([df_gene_and_var_test, df_text_test], axis=1)
df_cv = pd.concat([df_gene_and_var_cv, df_text_cv], axis=1)

# scaling the text_count feature
scaler = MinMaxScaler()
train_df["Word_Count"] = scaler.fit_transform(np.array(train_df["Word_Count"]).reshape(-1,1))
test_df["Word_Count"] = scaler.fit_transform(np.array(test_df["Word_Count"]).reshape(-1,1))
cv_df["Word_Count"] = scaler.fit_transform(np.array(cv_df["Word_Count"]).reshape(-1,1))

train_df["Character_Count"] = scaler.fit_transform(np.array(train_df["Character_Count"]).reshape(-1,1))
test_df["Character_Count"] = scaler.fit_transform(np.array(test_df["Character_Count"]).reshape(-1,1))
cv_df["Character_Count"] = scaler.fit_transform(np.array(cv_df["Character_Count"]).reshape(-1,1))

train_df["Avg_length"] = scaler.fit_transform(np.array(train_df["Avg_length"]).reshape(-1,1))
test_df["Avg_length"] = scaler.fit_transform(np.array(test_df["Avg_length"]).reshape(-1,1))
cv_df["Avg_length"] = scaler.fit_transform(np.array(cv_df["Avg_length"]).reshape(-1,1))

df_train["Gene_Share"] = train_df.Gene_Share.values
df_train["Variation_Share"] = train_df.Variation_Share.values
df_train["Word_Count_5000"] = train_df.Word_Count_5000.values
df_train["Word_Count"] = train_df.Word_Count.values
df_train["Character_Count"] = train_df.Character_Count.values
df_train["Avg_length"] = train_df.Avg_length.values

df_test["Gene_Share"] = test_df.Gene_Share.values
df_test["Variation_Share"] = test_df.Variation_Share.values
df_test["Word_Count_5000"] = test_df.Word_Count_5000.values
df_test["Word_Count"] = test_df.Word_Count.values
df_test["Character_Count"] = test_df.Character_Count.values
df_test["Avg_length"] = test_df.Avg_length.values

df_cv["Gene_Share"] = cv_df.Gene_Share.values
df_cv["Variation_Share"] = cv_df.Variation_Share.values
df_cv["Word_Count_5000"] = cv_df.Word_Count_5000.values
df_cv["Word_Count"] = cv_df.Word_Count.values
df_cv["Character_Count"] = cv_df.Character_Count.values
df_cv["Avg_length"] = cv_df.Avg_length.values

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

```

```

train_geneandvar_responseCoding =
np.hstack((train_gene_var_responseCoding,train_gene_and_variation_feature_responseCoding))
test_geneandvar_responseCoding =
np.hstack((test_gene_var_responseCoding,test_gene_and_variation_feature_responseCoding))
cv_geneandvar_responseCoding =
np.hstack((cv_gene_var_responseCoding,cv_gene_and_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_geneandvar_responseCoding,
train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_geneandvar_responseCoding,
test_text_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_geneandvar_responseCoding, cv_text_feature_responseCoding))

train_geneshare = np.column_stack((train_x_responseCoding, train_df.Gene_Share.values))
test_geneshare = np.column_stack((test_x_responseCoding, test_df.Gene_Share.values))
cv_geneshare = np.column_stack((cv_x_responseCoding, cv_df.Gene_Share.values))

train_varshare = np.column_stack((train_geneshare, train_df.Variation_Share.values))
test_varshare = np.column_stack((test_geneshare, test_df.Variation_Share.values))
cv_varshare = np.column_stack((cv_geneshare, cv_df.Variation_Share.values))

train_text5000 = np.column_stack((train_varshare, train_df.Word_Count_5000.values))
test_text5000 = np.column_stack((test_varshare, test_df.Word_Count_5000.values))
cv_text5000 = np.column_stack((cv_varshare, cv_df.Word_Count_5000.values))

train_x_response = np.column_stack((train_text5000, train_df.Word_Count.values))
test_x_response = np.column_stack((test_text5000, test_df.Word_Count.values))
cv_x_response = np.column_stack((cv_text5000, cv_df.Word_Count.values))

train_x_response = np.column_stack((train_x_response, train_df.Character_Count.values))
test_x_response = np.column_stack((test_x_response, test_df.Character_Count.values))
cv_x_response = np.column_stack((cv_x_response, cv_df.Character_Count.values))

train_x_response = np.column_stack((train_x_response, train_df.Avg_length.values))
test_x_response = np.column_stack((test_x_response, test_df.Avg_length.values))
cv_x_response = np.column_stack((cv_x_response, cv_df.Avg_length.values))

train_x_onehotCoding = df_train
test_x_onehotCoding = df_test
cv_x_onehotCoding = df_cv

train_x_responseCoding = train_x_response
test_x_responseCoding = test_x_response
cv_x_responseCoding = cv_x_response

```

In [89]:

```

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, 9356)
(number of data points * number of features) in test data = (665, 9356)
(number of data points * number of features) in cross validation data = (532, 9356)

```

In [90]:

```

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

```

```

Response encoding features :
(number of data points * number of features) in train data = (2124, 42)
(number of data points * number of features) in test data = (665, 42)

```

(number of data points \* number of features) in cross validation data = (532, 42)

## 4.1. Base Line Model

### 4.1.1. Naive Bayes

#### 4.1.1.1. Hyper parameter tuning

In [91]:

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

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

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

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

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

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



```

sig_clf.fit(train_x_onehotCoding, train_y)

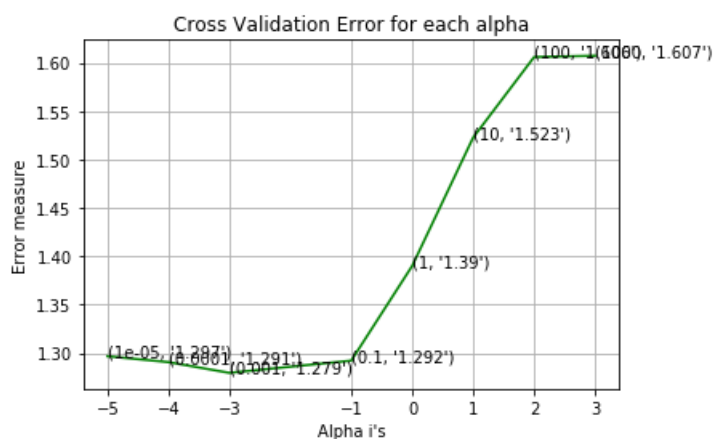
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-05
Log Loss : 1.2966809544298779
for alpha = 0.0001
Log Loss : 1.290571517636302
for alpha = 0.001
Log Loss : 1.2794608010736856
for alpha = 0.1
Log Loss : 1.2920931451092204
for alpha = 1
Log Loss : 1.3900418014389437
for alpha = 10
Log Loss : 1.5226946963896313
for alpha = 100
Log Loss : 1.6059394452144402
for alpha = 1000
Log Loss : 1.607286845403242

```



```

For values of best alpha = 0.001 The train log loss is: 0.6659686415447562
For values of best alpha = 0.001 The cross validation log loss is: 1.2794608010736856
For values of best alpha = 0.001 The test log loss is: 1.289018198008959

```

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

In [92]:

```

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

```

```

# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# -----

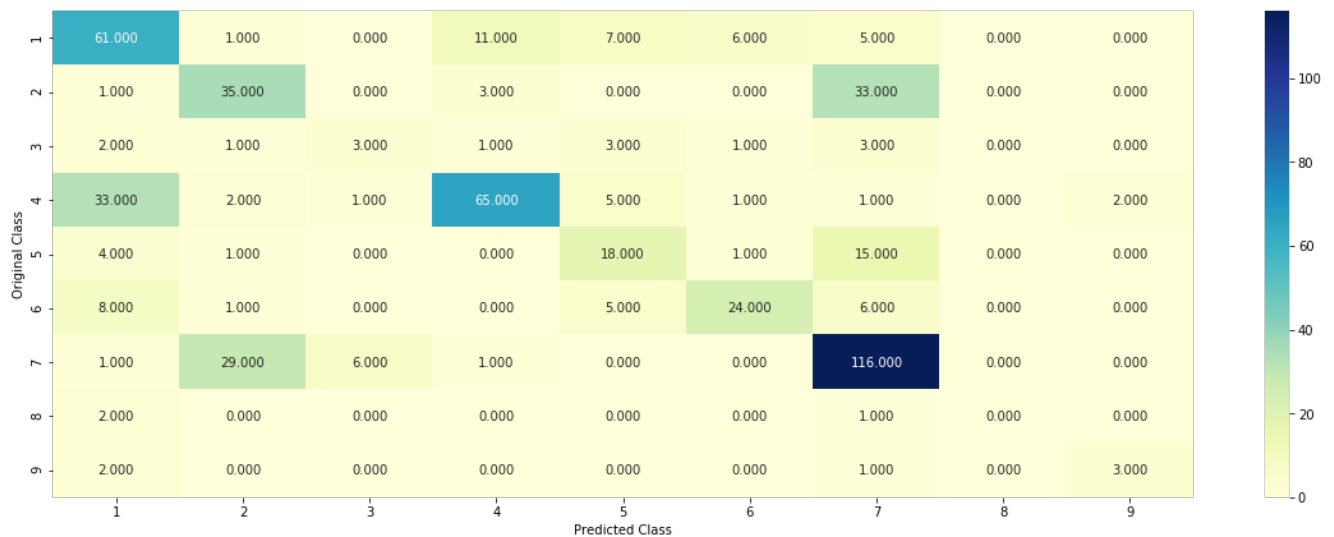
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use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

```

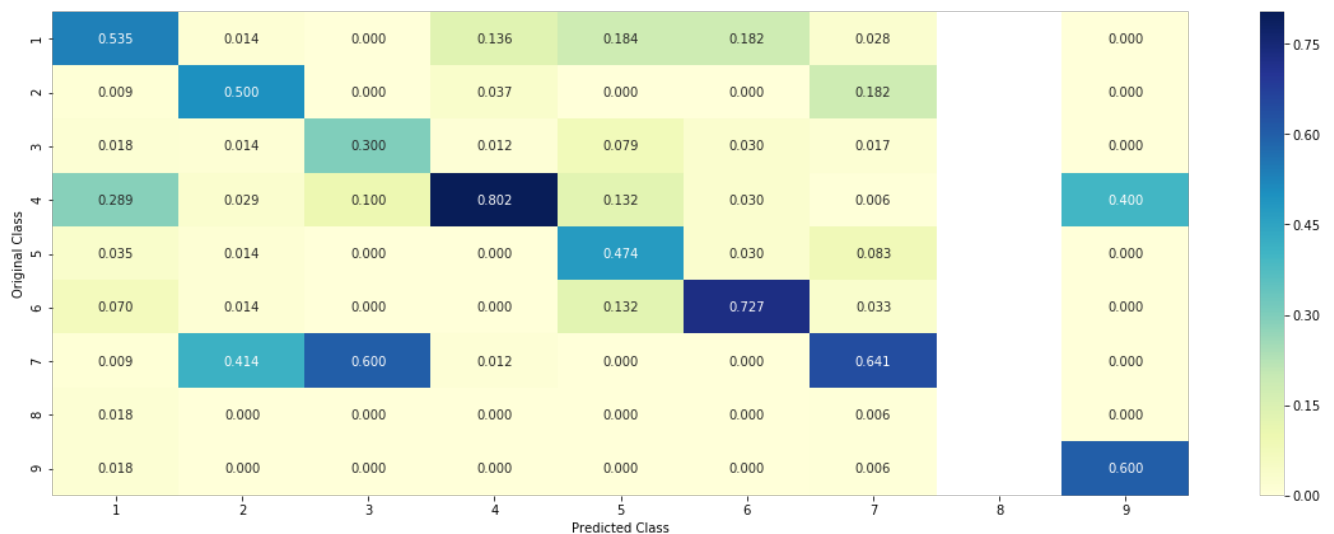
Log Loss : 1.2794608010736856

Number of missclassified point : 0.3890977443609023

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

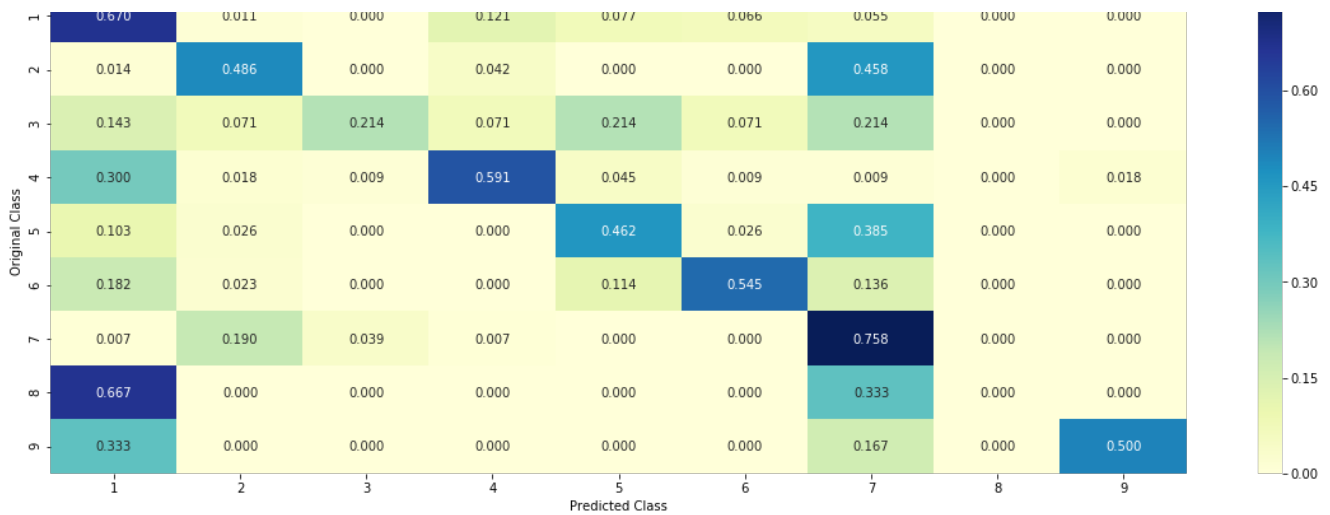


----- Precision matrix (Column Sum=1) -----



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





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

In [96]:

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

Predicted Class : 6

Predicted Class Probabilities: [[0.0612 0.0505 0.0106 0.0791 0.0339 0.6506 0.1071 0.0044 0.0025]]

Actual Class : 6

```
-----
36 Text feature [ring] present in test data point [True]
52 Text feature [noted] present in test data point [True]
58 Text feature [white] present in test data point [True]
78 Text feature [important] present in test data point [True]
79 Text feature [middle] present in test data point [True]
87 Text feature [however] present in test data point [True]
Out of the top 100 features 6 are present in query point
```

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

In [99]:

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

Predicted Class : 1

Predicted Class Probabilities: [[0.5459 0.0608 0.0127 0.1641 0.0406 0.0391 0.1286 0.0053 0.003 ]]

Actual Class : 1

```
-----
16 Text feature [probability] present in test data point [True]
26 Text feature [mean] present in test data point [True]
30 Text feature [well] present in test data point [True]
41 Text feature [isoforms] present in test data point [True]
47 Text feature [three] present in test data point [True]
55 Text feature [indicated] present in test data point [True]
71 Text feature [top] present in test data point [True]
82 Text feature [removal] present in test data point [True]
86 Text feature [invasive] present in test data point [True]
92 Text feature [usually] present in test data point [True]
96 Text feature [variable] present in test data point [True]
Out of the top 100 features 11 are present in query point
```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

In [100]:

```
# 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)

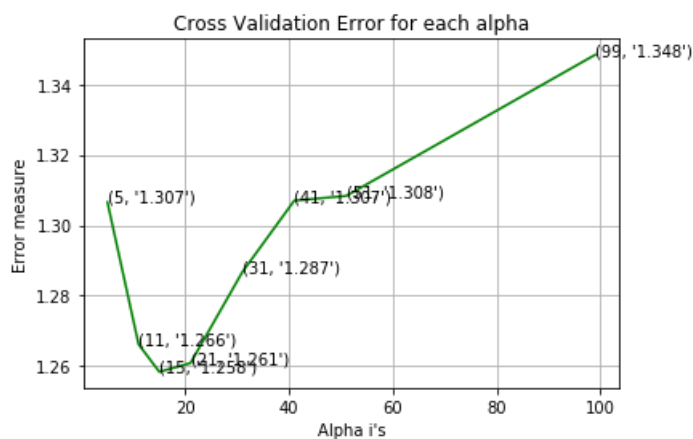
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 5
Log Loss : 1.3065703421717838
for alpha = 11
Log Loss : 1.266197197239213
for alpha = 15
Log Loss : 1.258370620936781
for alpha = 21
Log Loss : 1.2608519051132232
for alpha = 31
Log Loss : 1.2867179342036257
for alpha = 41
Log Loss : 1.3069954506153552
for alpha = 51
Log Loss : 1.3083217444856459
for alpha = 99
Log Loss : 1.3484200553333359

```



```

For values of best alpha = 15 The train log loss is: 0.9684257659571893
For values of best alpha = 15 The cross validation log loss is: 1.258370620936781
For values of best alpha = 15 The test log loss is: 1.2898918857002928

```

## 4.2.2. Testing the model with best hyper paramters

In [101]:

```

# 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

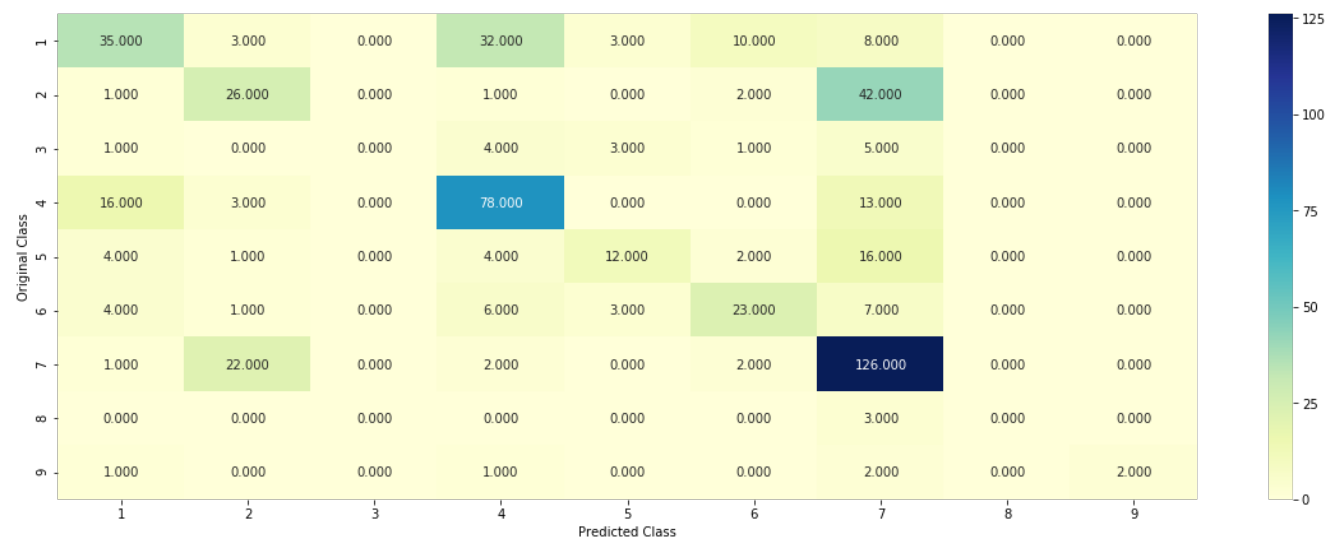
```

```
# predict(X):Return the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

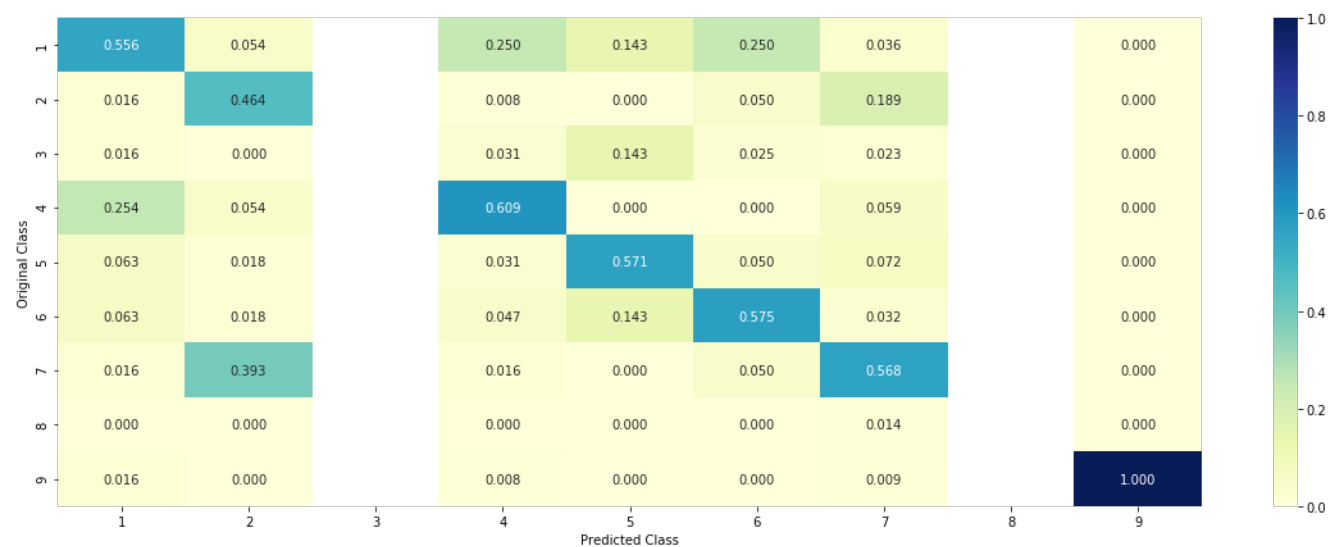
Log loss : 1.258370620936781

Number of mis-classified points : 0.4323308270676692

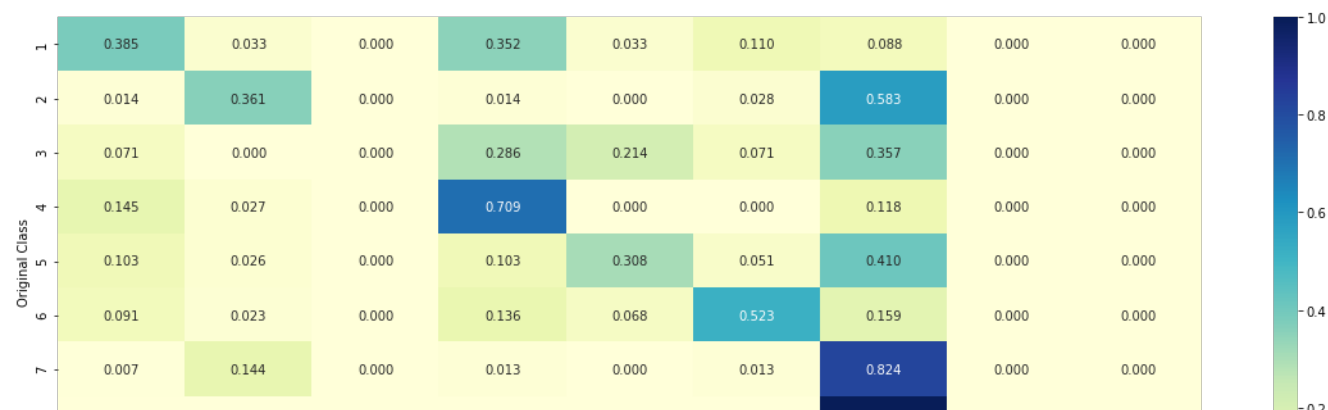
----- Confusion matrix -----

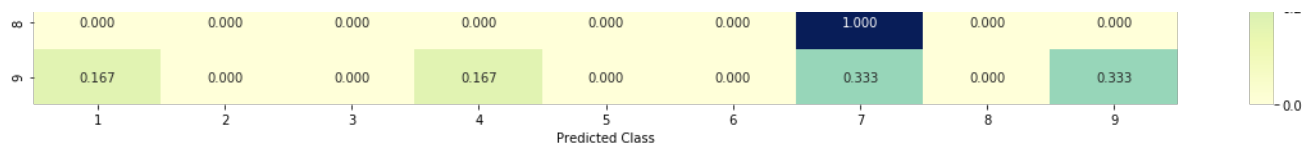


----- Precision matrix (Column Sum=1) -----



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





### 4.2.3. Sample Query point -1

In [102]:

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

Predicted Class : 4

Actual Class : 6

The 15 nearest neighbours of the test points belongs to classes [6 6 6 6 6 6 6 6 6 6 6 6 6 6 6]

Frequency of nearest points : Counter({6: 15})

### 4.2.4. Sample Query Point-2

In [103]:

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

Predicted Class : 1

Actual Class : 1

the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

Frequency of nearest points : Counter({1: 14, 4: 1})

## 4.3. Logistic Regression

### 4.3.1. With Class balancing

#### 4.3.1.1. Hyper paramter tuning

In [104]:

```
# 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.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

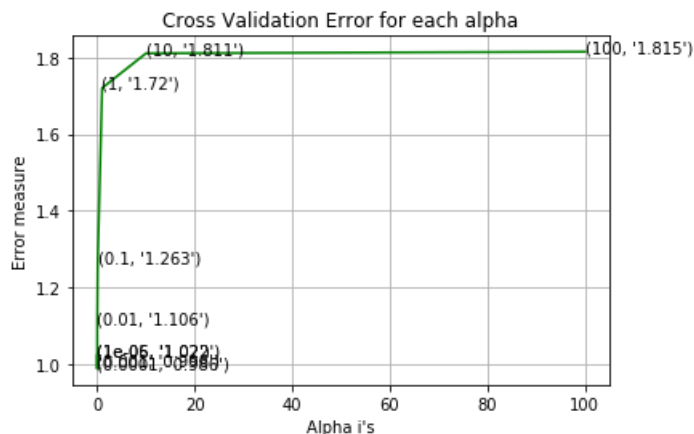
```



```

Log Loss : 1.0216920449958709
for alpha = 1e-05
Log Loss : 1.0200941235945424
for alpha = 0.0001
Log Loss : 0.9863648707366021
for alpha = 0.001
Log Loss : 0.9958232790001812
for alpha = 0.01
Log Loss : 1.1057708341079502
for alpha = 0.1
Log Loss : 1.2628718982201375
for alpha = 1
Log Loss : 1.7201085035459416
for alpha = 10
Log Loss : 1.8107710703723614
for alpha = 100
Log Loss : 1.8150070150202384

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4139835037553935
For values of best alpha = 0.0001 The cross validation log loss is: 0.9863648707366021
For values of best alpha = 0.0001 The test log loss is: 0.9878351128491554

```

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

In [105]:

```

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

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

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

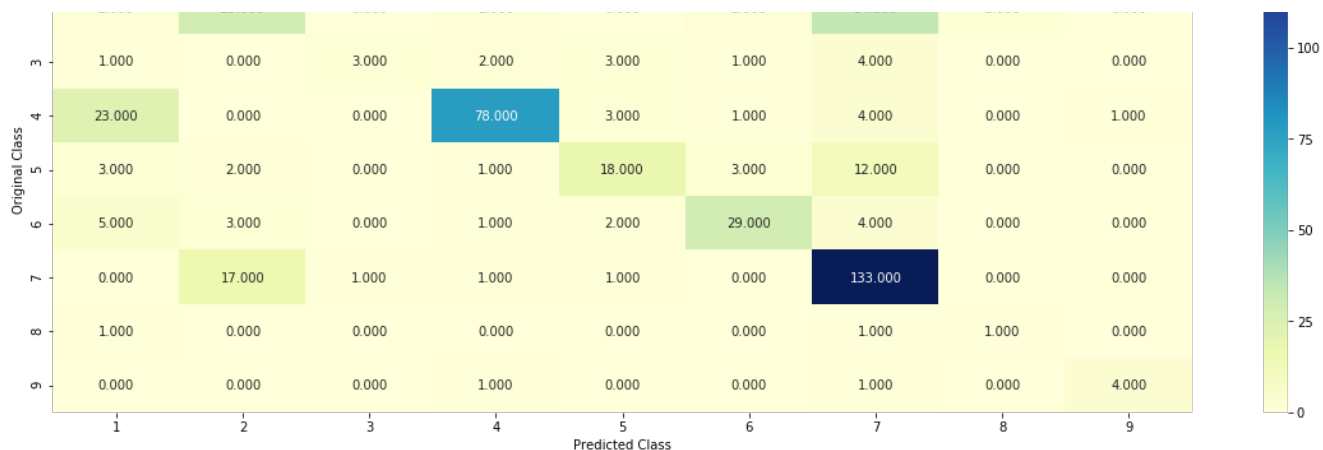
```

```

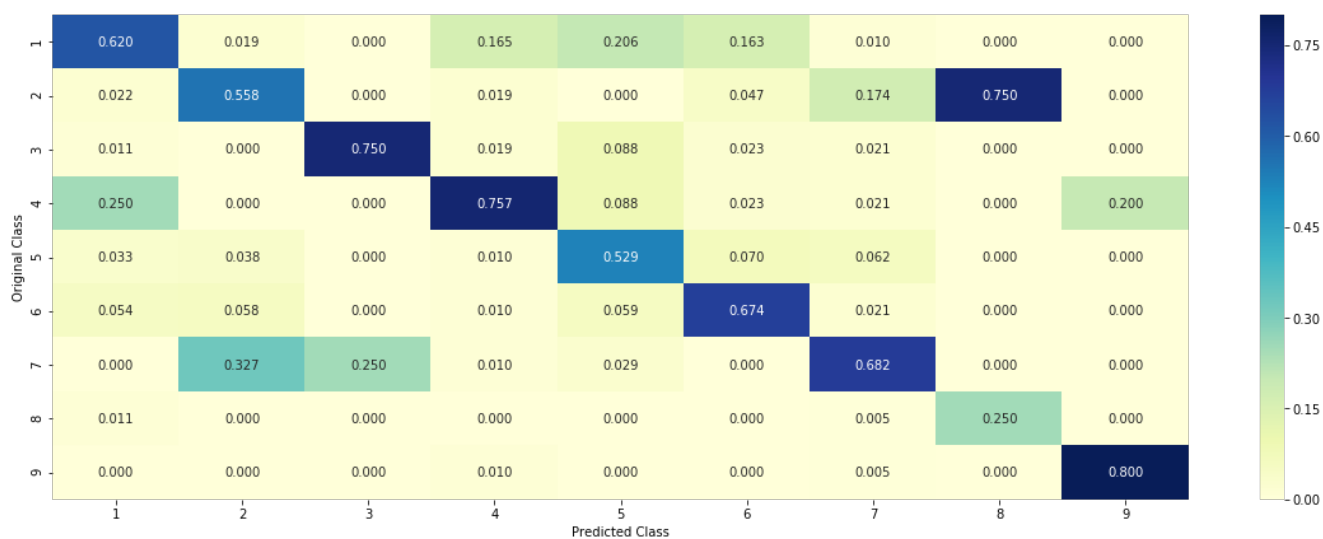
Log loss : 0.9863648707366021
Number of mis-classified points : 0.3383458646616541
----- Confusion matrix -----

```

1	57.000	1.000	0.000	17.000	7.000	7.000	2.000	0.000	0.000
2	2.000	29.000	0.000	2.000	0.000	2.000	34.000	3.000	0.000



Precision matrix (Column Sum=1)



Recall matrix (Row sum=1)



#### 4.3.1.3. Feature Importance

In [106]:

```
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. Correctly Classified point

In [108]:

```

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

```

Predicted Class : 6

Predicted Class Probabilities: [[1.220e-02 2.960e-02 2.400e-03 1.330e-02 6.590e-02 8.744e-01 5.000e-04

1.000e-03 6.000e-04]]

Actual Class : 6

```

-----
1 Text feature [access] present in test data point [True]
5 Text feature [32] present in test data point [True]
20 Text feature [cross] present in test data point [True]
32 Text feature [complex] present in test data point [True]
37 Text feature [71] present in test data point [True]
41 Text feature [components] present in test data point [True]
76 Text feature [carcinoma] present in test data point [True]
80 Text feature [coding] present in test data point [True]
82 Text feature [according] present in test data point [True]
92 Text feature [align] present in test data point [True]
111 Text feature [containing] present in test data point [True]
113 Text feature [category] present in test data point [True]
124 Text feature [4a] present in test data point [True]
163 Text feature [75] present in test data point [True]
182 Text feature [assumed] present in test data point [True]
202 Text feature [a1708e] present in test data point [True]
230 Text feature [receiving] present in test data point [True]
233 Text feature [practice] present in test data point [True]
259 Text feature [affected] present in test data point [True]
273 Text feature [proposed] present in test data point [True]
281 Text feature [two] present in test data point [True]
292 Text feature [origin] present in test data point [True]
298 Text feature [pattern] present in test data point [True]
335 Text feature [subtle] present in test data point [True]
336 Text feature [repair] present in test data point [True]
347 Text feature [subject] present in test data point [True]
357 Text feature [years] present in test data point [True]
368 Text feature [western] present in test data point [True]
369 Text feature [nearly] present in test data point [True]
384 Text feature [grouped] present in test data point [True]

```

```

304 Text feature [groups] present in test data point [True]
386 Text feature [proteins] present in test data point [True]
413 Text feature [unclear] present in test data point [True]
442 Text feature [process] present in test data point [True]
445 Text feature [preliminary] present in test data point [True]
466 Text feature [interestingly] present in test data point [True]
473 Text feature [mim] present in test data point [True]
480 Text feature [well] present in test data point [True]
487 Text feature [subgroups] present in test data point [True]
492 Text feature [level] present in test data point [True]
496 Text feature [posterior] present in test data point [True]
Out of the top 500 features 40 are present in query point

```

#### 4.3.1.3.2. Incorrectly Classified point

In [110]:

```

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

```

Predicted Class : 1

Predicted Class Probabilities: [[0.5708 0.0237 0.0076 0.3638 0.0112 0.0153 0.0046 0.0017 0.0013]]

Actual Class : 1

```

-----
39 Text feature [acting] present in test data point [True]
51 Text feature [62] present in test data point [True]
54 Text feature [encompassing] present in test data point [True]
64 Text feature [diagnosis] present in test data point [True]
67 Text feature [abstract] present in test data point [True]
68 Text feature [developmental] present in test data point [True]
72 Text feature [cancers] present in test data point [True]
116 Text feature [95] present in test data point [True]
117 Text feature [could] present in test data point [True]
121 Text feature [foci] present in test data point [True]
125 Text feature [germany] present in test data point [True]
129 Text feature [death] present in test data point [True]
141 Text feature [exhibit] present in test data point [True]
161 Text feature [dysregulation] present in test data point [True]
187 Text feature [73] present in test data point [True]
192 Text feature [addition] present in test data point [True]
194 Text feature [displays] present in test data point [True]
221 Text feature [established] present in test data point [True]
233 Text feature [mutation] present in test data point [True]
243 Text feature [88] present in test data point [True]
268 Text feature [targets] present in test data point [True]
274 Text feature [peak] present in test data point [True]
300 Text feature [full] present in test data point [True]
304 Text feature [spots] present in test data point [True]
322 Text feature [additional] present in test data point [True]
375 Text feature [case] present in test data point [True]
381 Text feature [side] present in test data point [True]
384 Text feature [abnormal] present in test data point [True]
412 Text feature [figure] present in test data point [True]
423 Text feature [involvement] present in test data point [True]
445 Text feature [relatives] present in test data point [True]
447 Text feature [families] present in test data point [True]
456 Text feature [generally] present in test data point [True]
470 Text feature [proteasomal] present in test data point [True]
474 Text feature [analysis] present in test data point [True]
480 Text feature [subgroup] present in test data point [True]
496 Text feature [pdgfr] present in test data point [True]
Out of the top 500 features 37 are present in query point

```

## 4.3.2. Without Class balancing

### 4.3.2.1. Hyper paramter tuning

In [111]:

```
# 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,
```

```

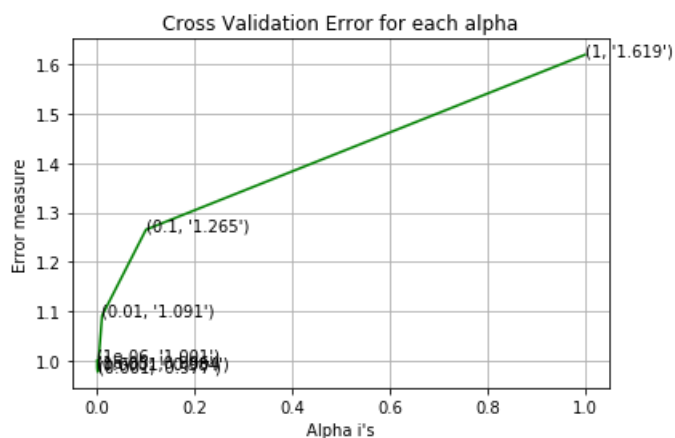
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
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 : 1.00062762982885
for alpha = 1e-05
Log Loss : 0.9863671881605045
for alpha = 0.0001
Log Loss : 0.984345705236273
for alpha = 0.001
Log Loss : 0.9773950399716071
for alpha = 0.01
Log Loss : 1.0912335291607955
for alpha = 0.1
Log Loss : 1.2651404950884308
for alpha = 1
Log Loss : 1.6189381646571483

```



```

For values of best alpha = 0.001 The train log loss is: 0.5221644625750487
For values of best alpha = 0.001 The cross validation log loss is: 0.9773950399716071
For values of best alpha = 0.001 The test log loss is: 0.9974372696535099

```

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

In [112]:

```

# 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:
#-----

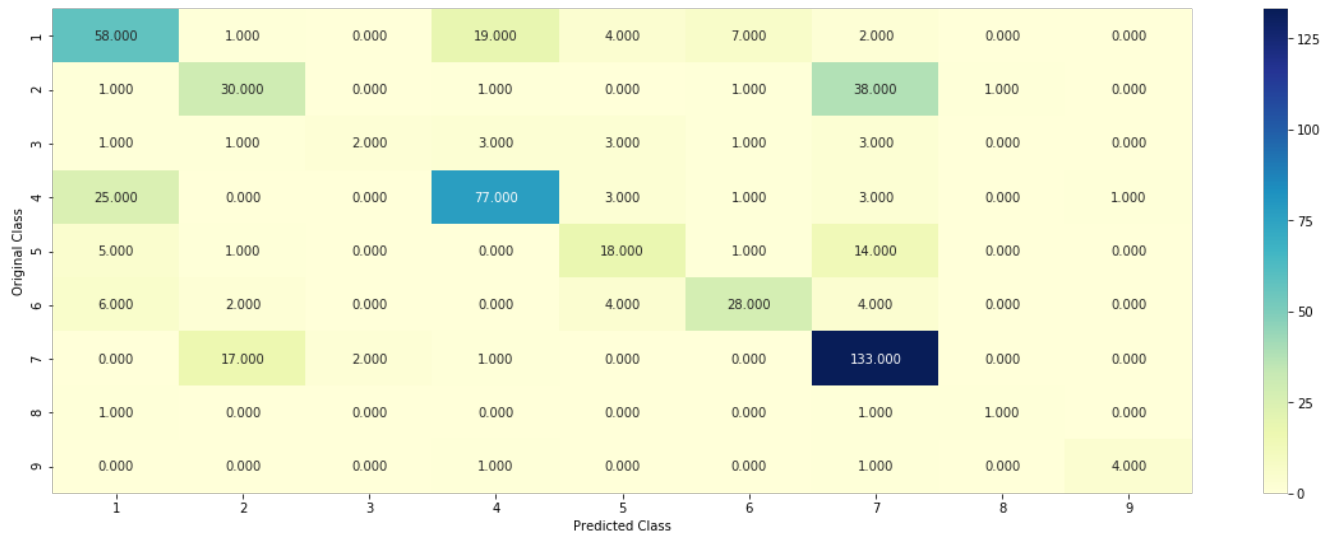
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

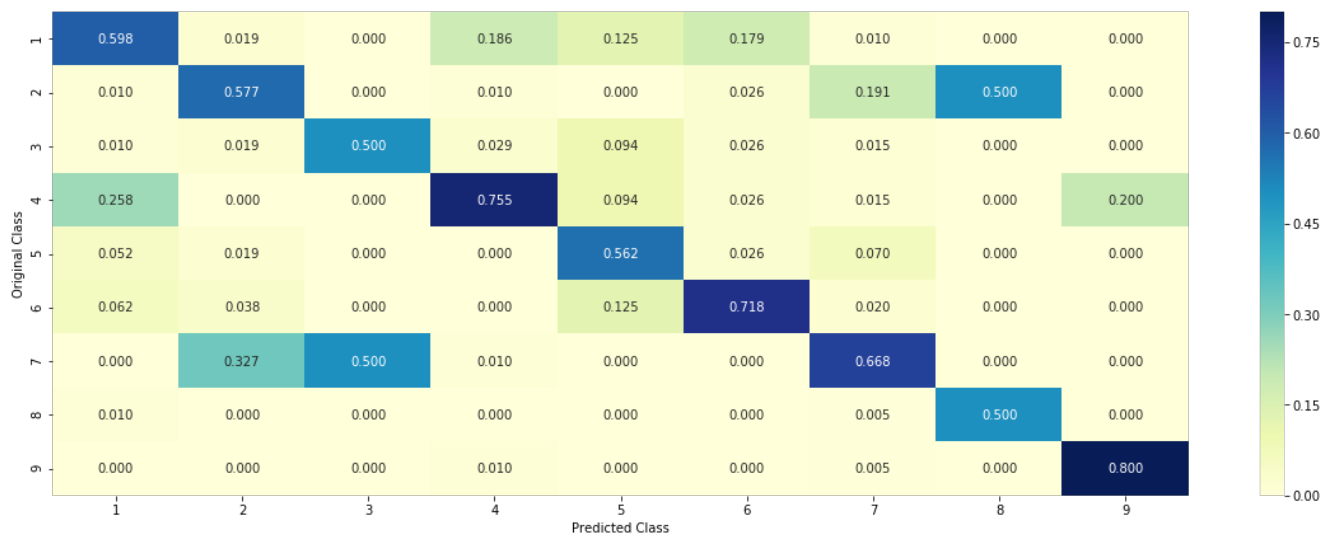
```

Log loss : 0.9773950399716071
Number of mis-classified points : 0.34022556390977443
----- Confusion matrix -----

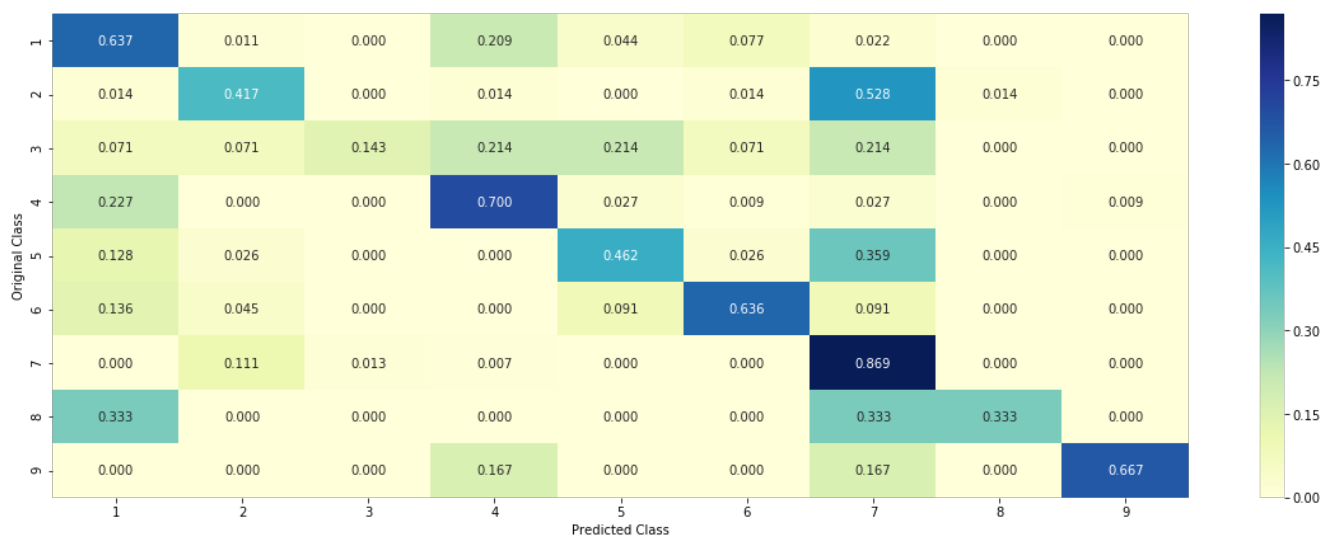
```



Precision matrix (Column Sum=1)



Recall matrix (Row sum=1)



Model	Type of Features Used	Train Loss	Test Loss	CV Loss	% of misclassified points
Naive Bayes	OneHotEncoded	0.6657	1.2794	1.2893	38.9
Decision Tree	OneHotEncoded	0.6668	1.2800	1.2914	41.0

K Nearest Neighbours	Response Coding	0.9803	1.2602	1.2011	44.9
Logistics Regression with Class Balancing	OneHotEncoded	0.5404	0.9929	1.0043	35.15
<b>Logistics Regression without Class Balancing</b>	<b>OneHotEncoded</b>	<b>0.4157</b>	<b>0.9755</b>	<b>0.9938</b>	<b>33.83</b>

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

In [114]:

```

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

```

Predicted Class : 6

Predicted Class Probabilities: [[8.000e-03 5.100e-03 1.700e-03 3.900e-03 1.226e-01 8.571e-01 3.000e-04

1.000e-03 4.000e-04]]

Actual Class : 6

```

-----
14 Text feature [32] present in test data point [True]
20 Text feature [components] present in test data point [True]
29 Text feature [access] present in test data point [True]
56 Text feature [cross] present in test data point [True]
75 Text feature [coding] present in test data point [True]
78 Text feature [4a] present in test data point [True]
97 Text feature [71] present in test data point [True]
98 Text feature [complex] present in test data point [True]
106 Text feature [carcinoma] present in test data point [True]
126 Text feature [assumed] present in test data point [True]
129 Text feature [two] present in test data point [True]
131 Text feature [containing] present in test data point [True]
136 Text feature [category] present in test data point [True]
147 Text feature [proteins] present in test data point [True]
157 Text feature [origin] present in test data point [True]
173 Text feature [subgroups] present in test data point [True]
191 Text feature [75] present in test data point [True]
211 Text feature [according] present in test data point [True]
239 Text feature [brca2] present in test data point [True]
293 Text feature [process] present in test data point [True]
305 Text feature [numbers] present in test data point [True]
307 Text feature [align] present in test data point [True]
324 Text feature [receiving] present in test data point [True]
353 Text feature [repair] present in test data point [True]
354 Text feature [western] present in test data point [True]
362 Text feature [clearly] present in test data point [True]
374 Text feature [subject] present in test data point [True]
407 Text feature [practice] present in test data point [True]
408 Text feature [groups] present in test data point [True]
417 Text feature [noted] present in test data point [True]
418 Text feature [level] present in test data point [True]
421 Text feature [increased] present in test data point [True]
430 Text feature [original] present in test data point [True]
431 Text feature [likelihood] present in test data point [True]
434 Text feature [missense] present in test data point [True]
436 Text feature [white] present in test data point [True]
443 Text feature [table] present in test data point [True]
459 Text feature [ring] present in test data point [True]
468 Text feature [nearly] present in test data point [True]
473 Text feature [reasons] present in test data point [True]
475 Text feature [al708e] present in test data point [True]

```



```

479 Text feature [unclear] present in test data point [True]
480 Text feature [rationale] present in test data point [True]
492 Text feature [proposed] present in test data point [True]
495 Text feature [pattern] present in test data point [True]
Out of the top 500 features 45 are present in query point

```

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

In [115]:

```

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

```

```

Predicted Class : 1
Predicted Class Probabilities: [[0.6012 0.0197 0.0067 0.3347 0.0143 0.0172 0.0026 0.0021 0.0014]]
Actual Class : 1

```

```

-----
25 Text feature [mutation] present in test data point [True]
37 Text feature [encompassing] present in test data point [True]
52 Text feature [developmental] present in test data point [True]
64 Text feature [acting] present in test data point [True]
65 Text feature [targets] present in test data point [True]
68 Text feature [death] present in test data point [True]
95 Text feature [62] present in test data point [True]
96 Text feature [involvement] present in test data point [True]
116 Text feature [cancers] present in test data point [True]
122 Text feature [displays] present in test data point [True]
129 Text feature [peak] present in test data point [True]
170 Text feature [exhibit] present in test data point [True]
173 Text feature [could] present in test data point [True]
261 Text feature [case] present in test data point [True]
264 Text feature [dysregulation] present in test data point [True]
266 Text feature [subgroup] present in test data point [True]
270 Text feature [95] present in test data point [True]
275 Text feature [established] present in test data point [True]
289 Text feature [germany] present in test data point [True]
292 Text feature [diagnosis] present in test data point [True]
294 Text feature [proteasomal] present in test data point [True]
356 Text feature [families] present in test data point [True]
357 Text feature [abstract] present in test data point [True]
391 Text feature [addition] present in test data point [True]
401 Text feature [failed] present in test data point [True]
414 Text feature [analysis] present in test data point [True]
415 Text feature [side] present in test data point [True]
423 Text feature [table] present in test data point [True]
433 Text feature [partial] present in test data point [True]
458 Text feature [pdgff] present in test data point [True]
461 Text feature [spots] present in test data point [True]
464 Text feature [additional] present in test data point [True]
475 Text feature [88] present in test data point [True]
489 Text feature [surgical] present in test data point [True]
493 Text feature [assay] present in test data point [True]
494 Text feature [73] present in test data point [True]
Out of the top 500 features 36 are present in query point

```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper paramter tuning

In [116]:

```
# 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))
```

```

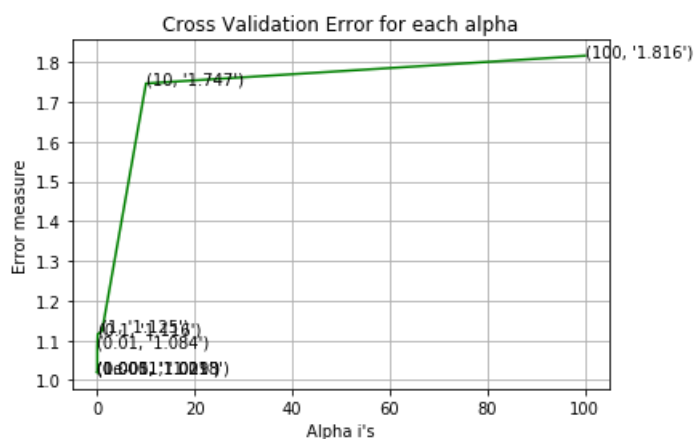
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

```

for C = 1e-05
Log Loss : 1.0193247406349206
for C = 0.0001
Log Loss : 1.018423111620109
for C = 0.001
Log Loss : 1.020664454665481
for C = 0.01
Log Loss : 1.0838840920051762
for C = 0.1
Log Loss : 1.1155063252493709
for C = 1
Log Loss : 1.1245128260791621
for C = 10
Log Loss : 1.7466203362438575
for C = 100
Log Loss : 1.8163121360648775

```



```

For values of best alpha = 0.0001 The train log loss is: 0.47013520012897614
For values of best alpha = 0.0001 The cross validation log loss is: 1.018423111620109
For values of best alpha = 0.0001 The test log loss is: 1.023697031287724

```

## 4.4.2. Testing model with best hyper parameters

In [117]:

```

# 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, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaiaicourse.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)

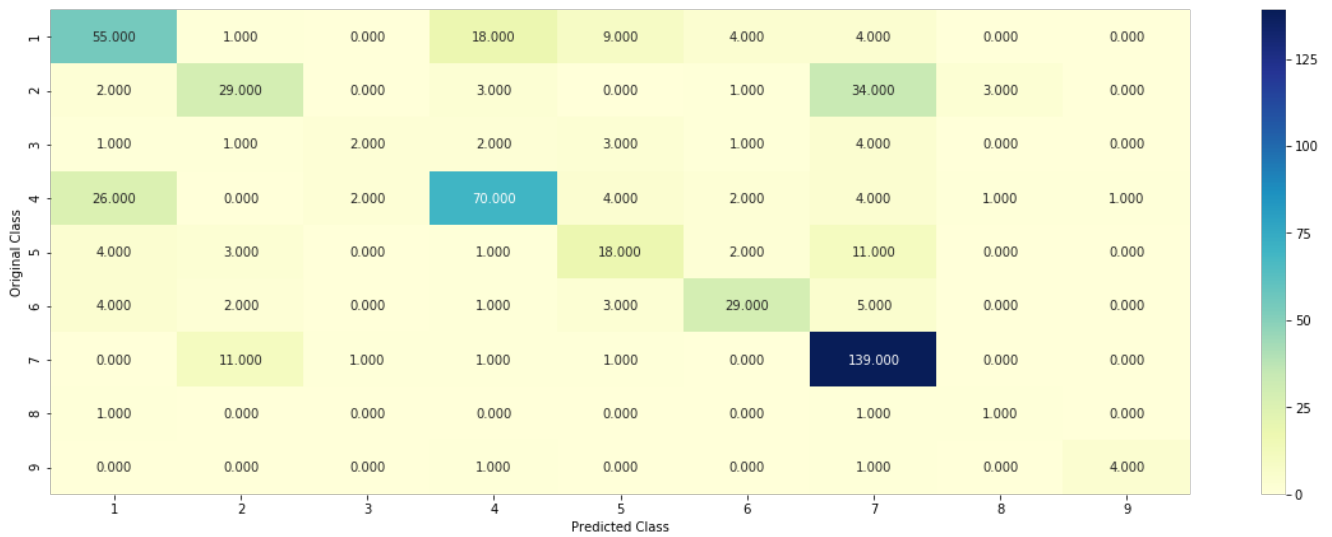
```

```

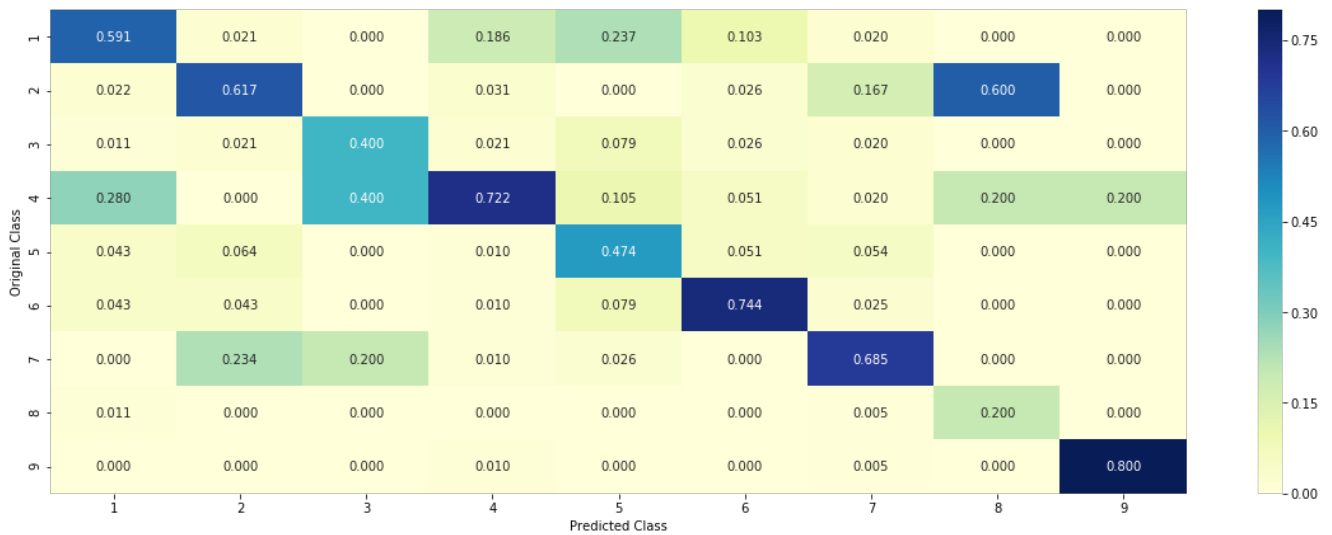
Log loss : 1.018423111620109
Number of mis-classified points : 0.34774436090225563

```

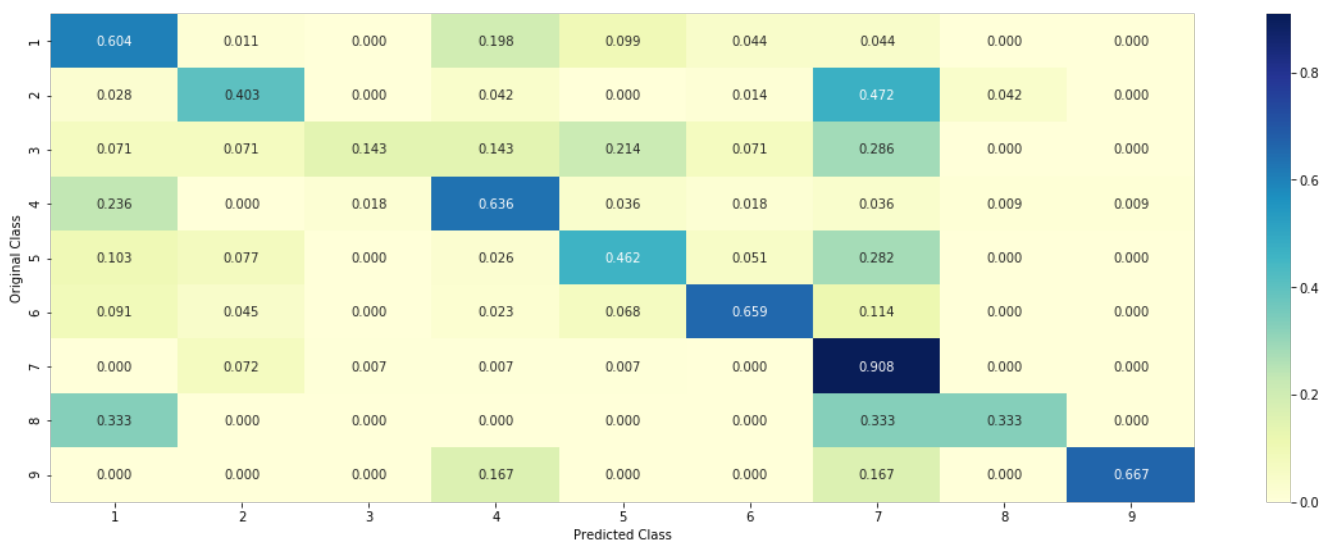
----- Confusion matrix -----



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



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



### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

In [119]:

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

Predicted Class : 6

Predicted Class Probabilities: [[0.0078 0.0131 0.0034 0.0072 0.098 0.8655 0.0026 0.0009 0.0015]]

Actual Class : 6

```
-----
6 Text feature [access] present in test data point [True]
9 Text feature [32] present in test data point [True]
10 Text feature [cross] present in test data point [True]
15 Text feature [71] present in test data point [True]
17 Text feature [complex] present in test data point [True]
39 Text feature [components] present in test data point [True]
54 Text feature [according] present in test data point [True]
56 Text feature [category] present in test data point [True]
64 Text feature [carcinoma] present in test data point [True]
107 Text feature [4a] present in test data point [True]
114 Text feature [align] present in test data point [True]
118 Text feature [containing] present in test data point [True]
151 Text feature [75] present in test data point [True]
173 Text feature [coding] present in test data point [True]
195 Text feature [practice] present in test data point [True]
215 Text feature [pattern] present in test data point [True]
217 Text feature [two] present in test data point [True]
223 Text feature [origin] present in test data point [True]
224 Text feature [repair] present in test data point [True]
249 Text feature [proteins] present in test data point [True]
278 Text feature [subject] present in test data point [True]
297 Text feature [al] present in test data point [True]
304 Text feature [affected] present in test data point [True]
318 Text feature [affect] present in test data point [True]
357 Text feature [brca2] present in test data point [True]
363 Text feature [assumed] present in test data point [True]
381 Text feature [a1708e] present in test data point [True]
382 Text feature [clinically] present in test data point [True]
384 Text feature [clearly] present in test data point [True]
391 Text feature [algorithm] present in test data point [True]
411 Text feature [nearly] present in test data point [True]
426 Text feature [years] present in test data point [True]
428 Text feature [proposed] present in test data point [True]
437 Text feature [level] present in test data point [True]
445 Text feature [process] present in test data point [True]
449 Text feature [unclear] present in test data point [True]
460 Text feature [groups] present in test data point [True]
480 Text feature [receiving] present in test data point [True]
488 Text feature [includes] 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 [120]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(np.array(test_x_onehotCoding.iloc[test_point_index, :]).reshape(1, -1))
```

```

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

```

Predicted Class : 1

Predicted Class Probabilities: [[0.6579 0.0487 0.0081 0.2298 0.0248 0.0173 0.0093 0.0018 0.0022]]

Actual Class : 1

```

-----
15 Text feature [acting] present in test data point [True]
36 Text feature [germany] present in test data point [True]
38 Text feature [diagnosis] present in test data point [True]
50 Text feature [cancers] present in test data point [True]
54 Text feature [62] present in test data point [True]
57 Text feature [abnormal] present in test data point [True]
65 Text feature [dysregulation] present in test data point [True]
69 Text feature [developmental] present in test data point [True]
80 Text feature [could] present in test data point [True]
99 Text feature [abstract] present in test data point [True]
167 Text feature [case] present in test data point [True]
183 Text feature [addition] present in test data point [True]
196 Text feature [exhibit] present in test data point [True]
209 Text feature [foci] present in test data point [True]
231 Text feature [88] present in test data point [True]
238 Text feature [95] present in test data point [True]
266 Text feature [four] present in test data point [True]
287 Text feature [73] present in test data point [True]
289 Text feature [analysis] present in test data point [True]
293 Text feature [established] present in test data point [True]
313 Text feature [death] present in test data point [True]
322 Text feature [encompassing] present in test data point [True]
325 Text feature [failed] present in test data point [True]
331 Text feature [figure] present in test data point [True]
339 Text feature [displays] present in test data point [True]
375 Text feature [peak] present in test data point [True]
399 Text feature [targets] present in test data point [True]
401 Text feature [side] present in test data point [True]
408 Text feature [substitutions] present in test data point [True]
409 Text feature [mutation] present in test data point [True]
418 Text feature [pancreas] present in test data point [True]
427 Text feature [subgroup] present in test data point [True]
440 Text feature [surface] present in test data point [True]
447 Text feature [structure] present in test data point [True]
462 Text feature [spots] present in test data point [True]
466 Text feature [testing] present in test data point [True]
496 Text feature [surgery] present in test data point [True]
497 Text feature [involvement] present in test data point [True]
499 Text feature [proteasomal] present in test data point [True]
Out of the top 500 features 39 are present in query point

```

## 4.5 Random Forest Classifier

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

In [121]:

```

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

```

```

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

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

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

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

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

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

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

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

```

for n estimators = 100 and max depth = 5

```

Log Loss : 1.1982675640377527
for n_estimators = 100 and max depth = 10
Log Loss : 1.171152154040783
for n_estimators = 200 and max depth = 5
Log Loss : 1.1941005390919497
for n_estimators = 200 and max depth = 10
Log Loss : 1.152809332917723
for n_estimators = 500 and max depth = 5
Log Loss : 1.184684587960283
for n_estimators = 500 and max depth = 10
Log Loss : 1.150720100620576
for n_estimators = 1000 and max depth = 5
Log Loss : 1.1776128791553742
for n_estimators = 1000 and max depth = 10
Log Loss : 1.147642562668301
for n_estimators = 2000 and max depth = 5
Log Loss : 1.1763758735068064
for n_estimators = 2000 and max depth = 10
Log Loss : 1.14535628318609
For values of best estimator = 2000 The train log loss is: 0.5792981642276354
For values of best estimator = 2000 The cross validation log loss is: 1.14535628318609
For values of best estimator = 2000 The test log loss is: 1.1417129604631588

```

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

In [123]:

```

# -----
# 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-fores
t-and-their-construction-2/
# -----

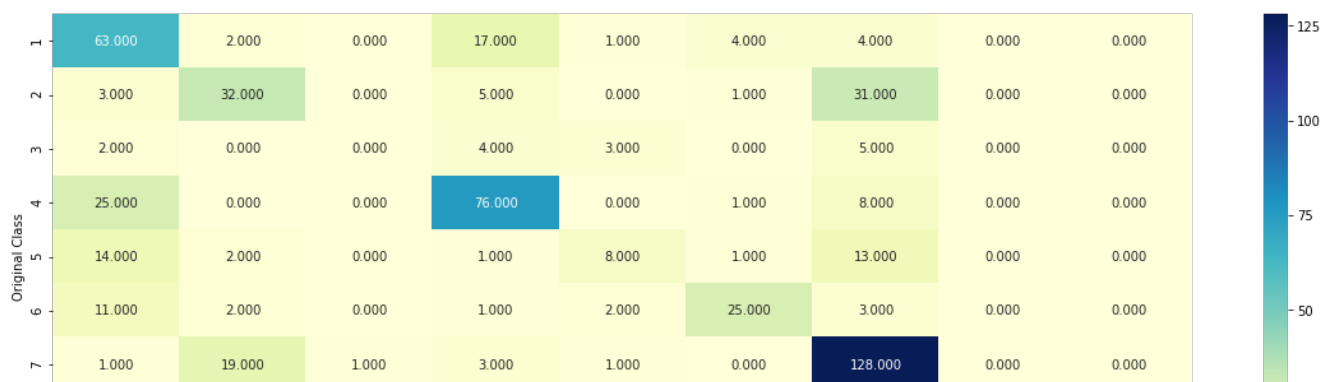
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)

```

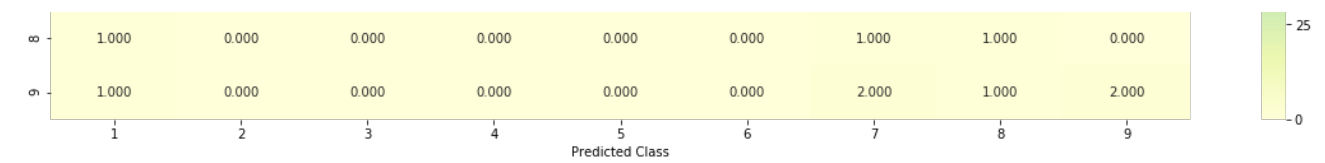
Log loss : 1.14535628318609

Number of mis-classified points : 0.37030075187969924

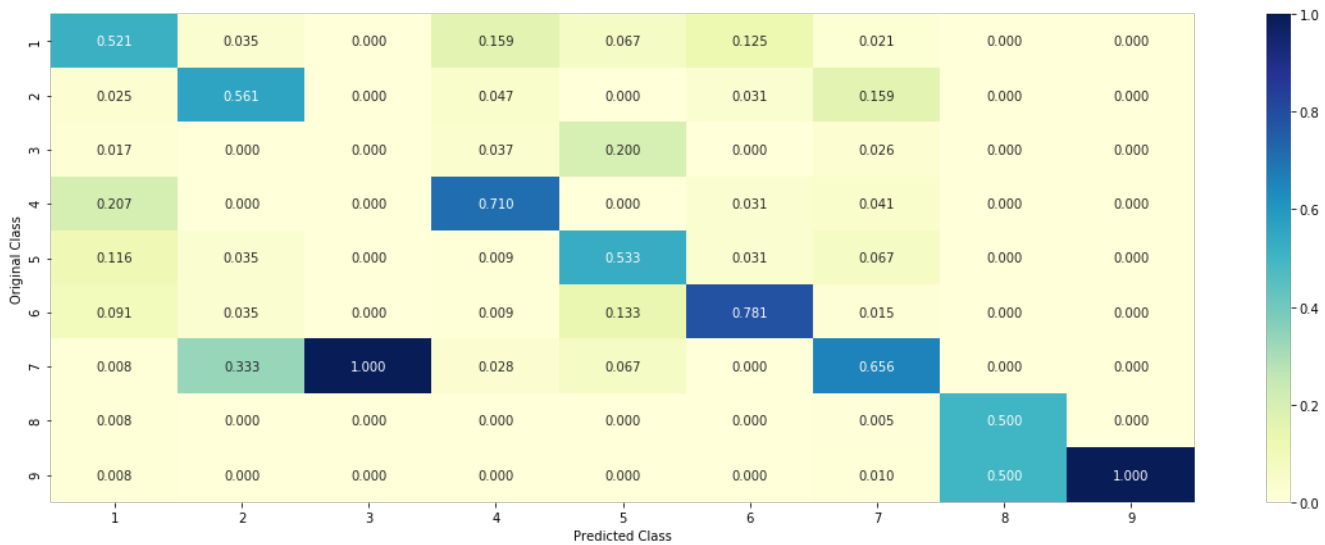
----- Confusion matrix -----



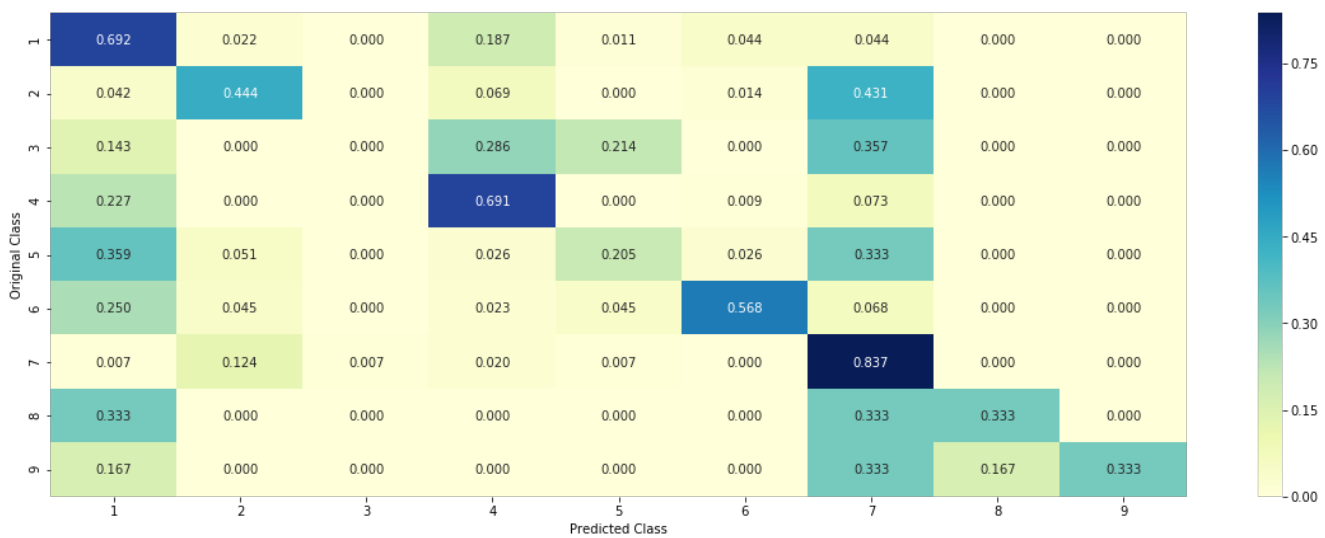




----- Precision matrix (Column Sum=1) -----



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



## 4.5.3. Feature Importance

### 4.5.3.1. Correctly Classified point

In [124]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(np.array(test_x_onehotCoding.iloc[test_point_index, :]).reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:")
```

```

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

```

Predicted Class : 6

Predicted Class Probabilities: [[0.0309 0.0174 0.0114 0.0207 0.2025 0.6912 0.0188 0.0027 0.0046]]

Actual Class : 6

```

-----
17 Text feature [well] present in test data point [True]
21 Text feature [statistical] present in test data point [True]
44 Text feature [tolerated] present in test data point [True]
65 Text feature [similar] present in test data point [True]
83 Text feature [respectively] present in test data point [True]
91 Text feature [probably] present in test data point [True]
94 Text feature [many] present in test data point [True]
Out of the top 100 features 7 are present in query point

```

#### 4.5.3.2. Inorrectly Classified point

In [125]:

```

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

```

Predicted Class : 1

Predicted Class Probabilities: [[0.3475 0.144 0.0213 0.2497 0.0621 0.0523 0.1058 0.0072 0.0101]]

Actuall Class : 1

```

-----
17 Text feature [well] present in test data point [True]
32 Text feature [mutants] present in test data point [True]
51 Text feature [prostate] present in test data point [True]
65 Text feature [similar] present in test data point [True]
72 Text feature [retinoblastoma] present in test data point [True]
79 Text feature [retention] present in test data point [True]
81 Text feature [rearrangements] present in test data point [True]
83 Text feature [respectively] present in test data point [True]
86 Text feature [interval] present in test data point [True]
94 Text feature [many] present in test data point [True]
Out of the top 100 features 10 are present in query point

```

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

In [126]:

```

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

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

```

```

# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

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

# -----
# video link: https://www.appliedaicomse.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 paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-----
# video link:
#-----

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

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_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 : 2.873614404746155
for n_estimators = 10 and max depth = 3

```

```

Log Loss : 2.5923034467532156
for n_estimators = 10 and max depth = 5
Log Loss : 2.4729494608092337
for n_estimators = 10 and max depth = 10
Log Loss : 1.9652847475884017
for n_estimators = 50 and max depth = 2
Log Loss : 2.8007903491445747
for n_estimators = 50 and max depth = 3
Log Loss : 2.2488055734519614
for n_estimators = 50 and max depth = 5
Log Loss : 2.2837510556614977
for n_estimators = 50 and max depth = 10
Log Loss : 2.3114213703879596
for n_estimators = 100 and max depth = 2
Log Loss : 2.509516171484144
for n_estimators = 100 and max depth = 3
Log Loss : 2.230080833740313
for n_estimators = 100 and max depth = 5
Log Loss : 2.120866868149124
for n_estimators = 100 and max depth = 10
Log Loss : 2.307978186874626
for n_estimators = 200 and max depth = 2
Log Loss : 2.5125874132267585
for n_estimators = 200 and max depth = 3
Log Loss : 2.341959802075814
for n_estimators = 200 and max depth = 5
Log Loss : 2.121295475628152
for n_estimators = 200 and max depth = 10
Log Loss : 2.289056027933227
for n_estimators = 500 and max depth = 2
Log Loss : 2.32175230548983
for n_estimators = 500 and max depth = 3
Log Loss : 2.22643003573414
for n_estimators = 500 and max depth = 5
Log Loss : 2.086138779673616
for n_estimators = 500 and max depth = 10
Log Loss : 2.2923714340256724
for n_estimators = 1000 and max depth = 2
Log Loss : 2.278823268952547
for n_estimators = 1000 and max depth = 3
Log Loss : 2.2052055654069487
for n_estimators = 1000 and max depth = 5
Log Loss : 2.075171561361834
for n_estimators = 1000 and max depth = 10
Log Loss : 2.2867325289584004
For values of best alpha = 10 The train log loss is: 0.01522883046488177
For values of best alpha = 10 The cross validation log loss is: 1.9652847475884017
For values of best alpha = 10 The test log loss is: 1.9728161717937343

```

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

In [127]:

```

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

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

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

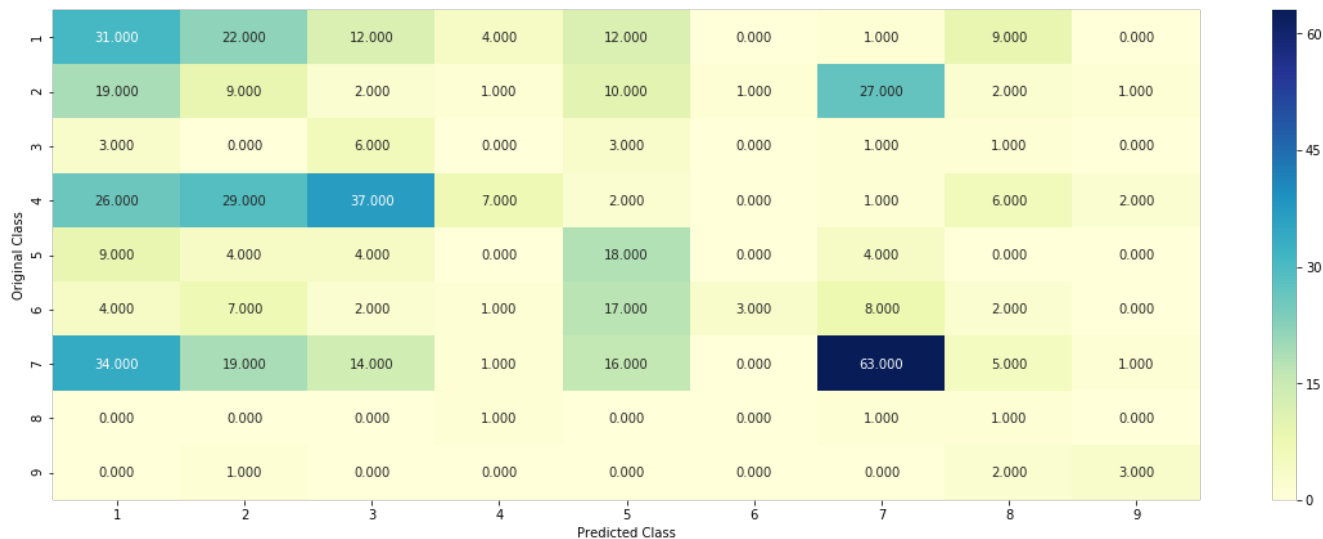
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/

```

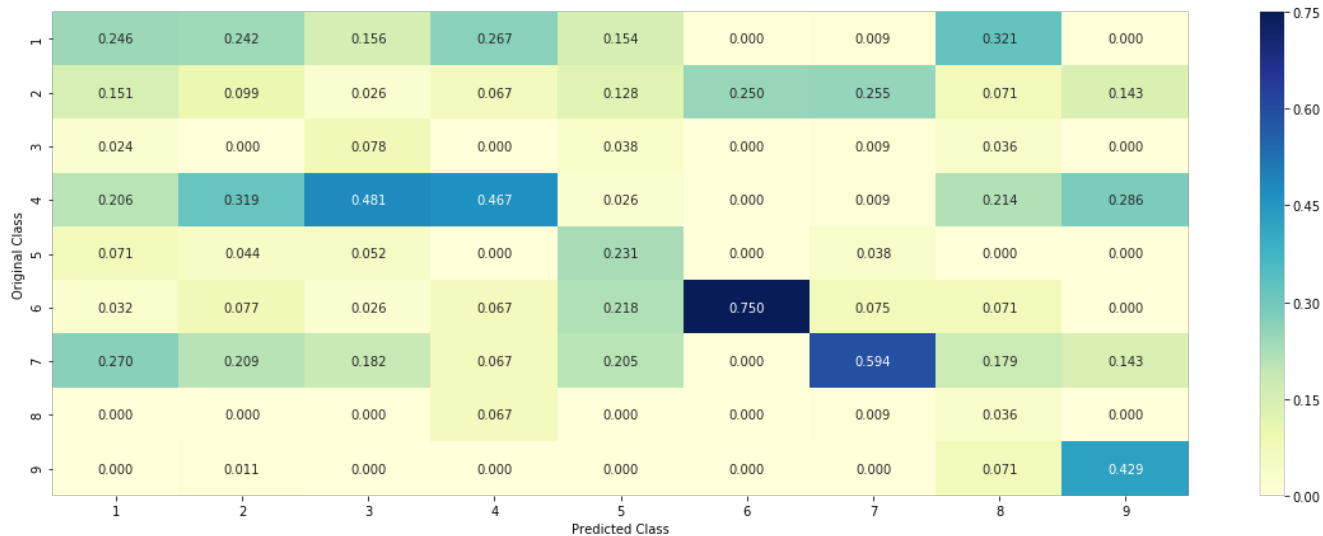
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)],
n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_state=42)
predict and plot confusion matrix(train x responseCoding, train y,cv x responseCoding,cv y, clf)
```

Number of mis-classified points : 0.7349624060150376

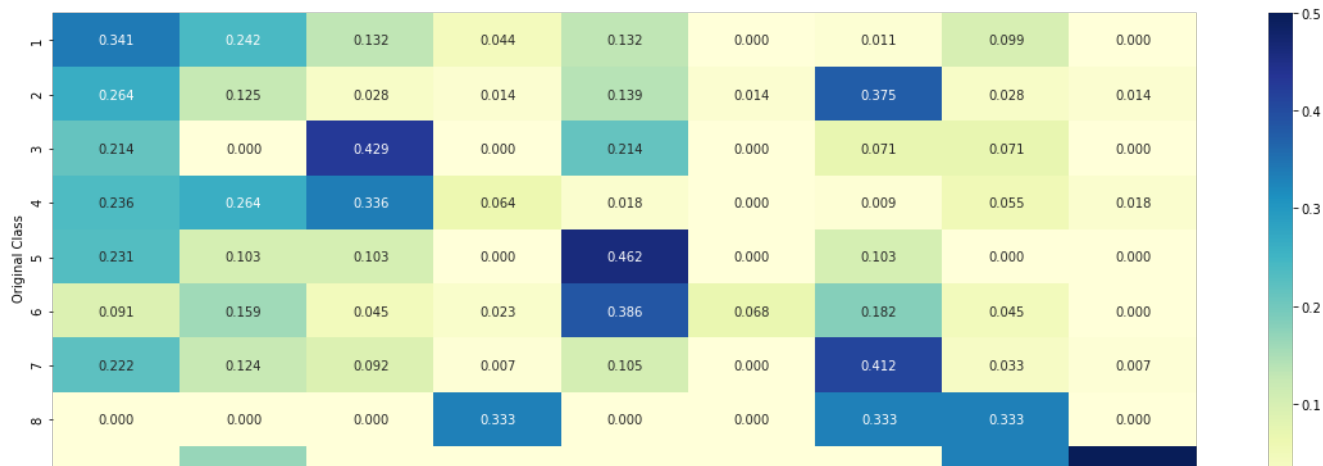
```
----- Confusion matrix -----
```

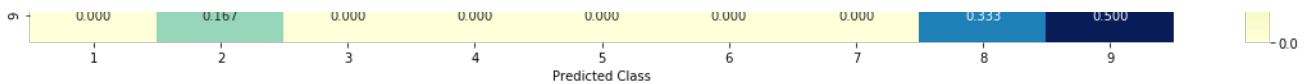


```
----- Precision matrix (Columm Sum=1) -----
```



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





## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point

In [101]:

```
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(np.array(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(np.array(test_x_responseCoding[test_point_index, :]).reshape(1,-1)),
4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 5
Predicted Class Probabilities: [[ 0.046  0.094  0.2307  0.0546  0.2904  0.0744  0.0482  0.0686  0.
.093 ]]
Actual Class : 6
```

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

#### 4.5.5.2. Incorrectly Classified point

In [102]:

```
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 : 1
Predicted Class Probabilities: [[ 0.6838  0.0116  0.0103  0.2706  0.0076  0.0056  0.0042  0.0034
 0.003 ]]
```

```
Actual Class : 1
```

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

Gene is important feature  
Text is important feature  
Text is important feature

## 4.7 Stack the models

### 4.7.1 testing with hyper parameter tuning

In [128]:

```
# 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/
#-----

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

```

```

Logistic Regression : Log Loss: 1.00
Support vector machines : Log Loss: 1.12
Naive Bayes : Log Loss: 1.28

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.176
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.015
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.454
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.119
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.348
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.934

```

## 4.7.2 testing the model with the best hyper parameters

In [129]:

```

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

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

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

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

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

```

```

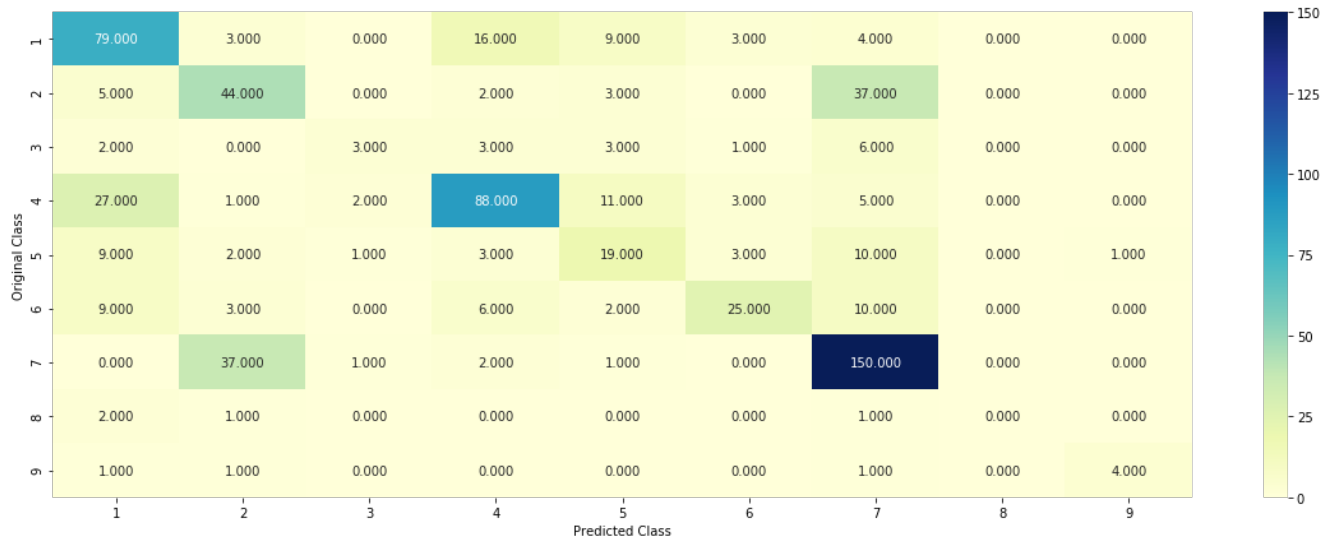
Log loss (train) on the stacking classifier : 0.5270143360902533
Log loss (CV) on the stacking classifier : 1.1191933625319337

```

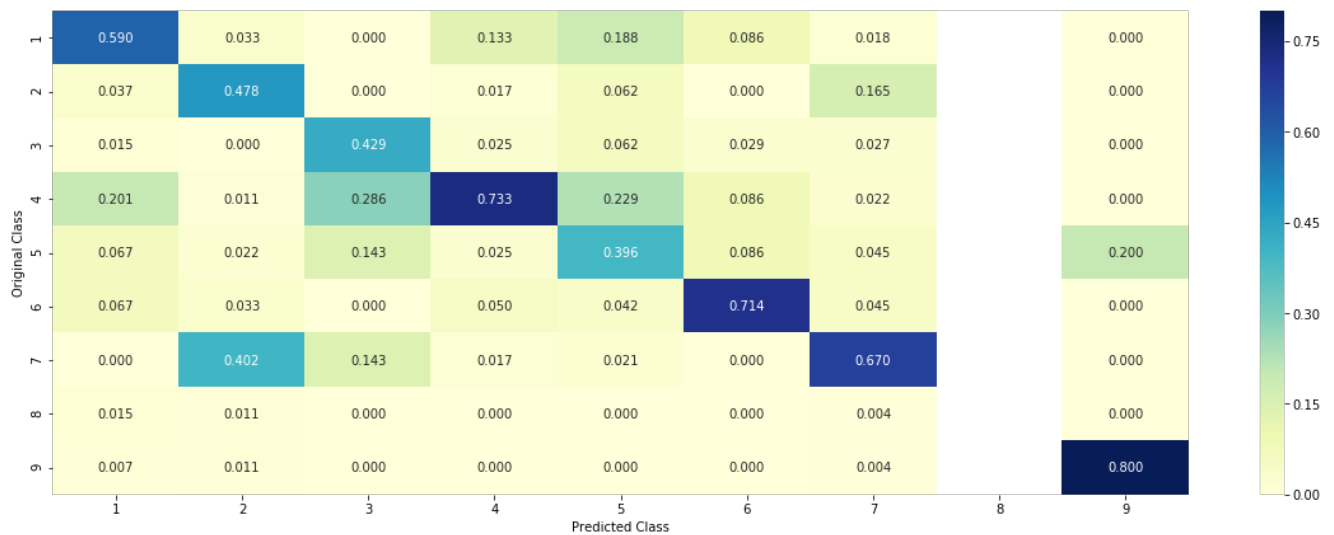
Log loss (test) on the stacking classifier : 1.1397037310843494

Number of missclassified point : 0.3804511278195489

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



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



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



### 4.7.3 Maximum Voting classifier

In [130]:

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

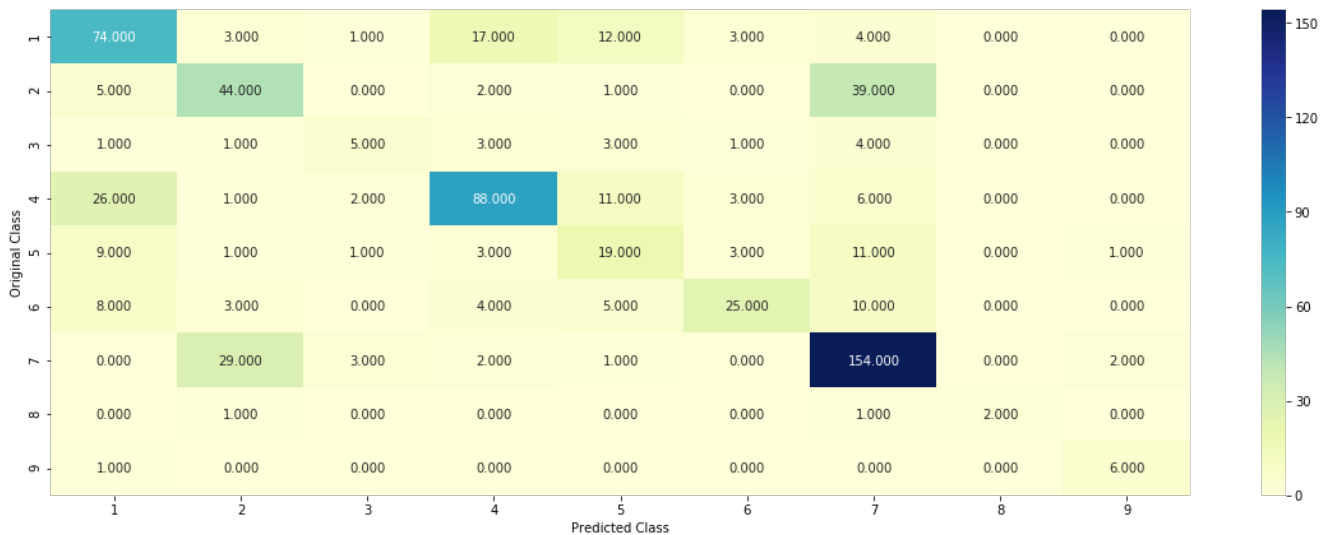
```
Log loss (train) on the VotingClassifier : 0.6459383722232275
```

Log loss (CV) on the VotingClassifier : 1.0821671744903376

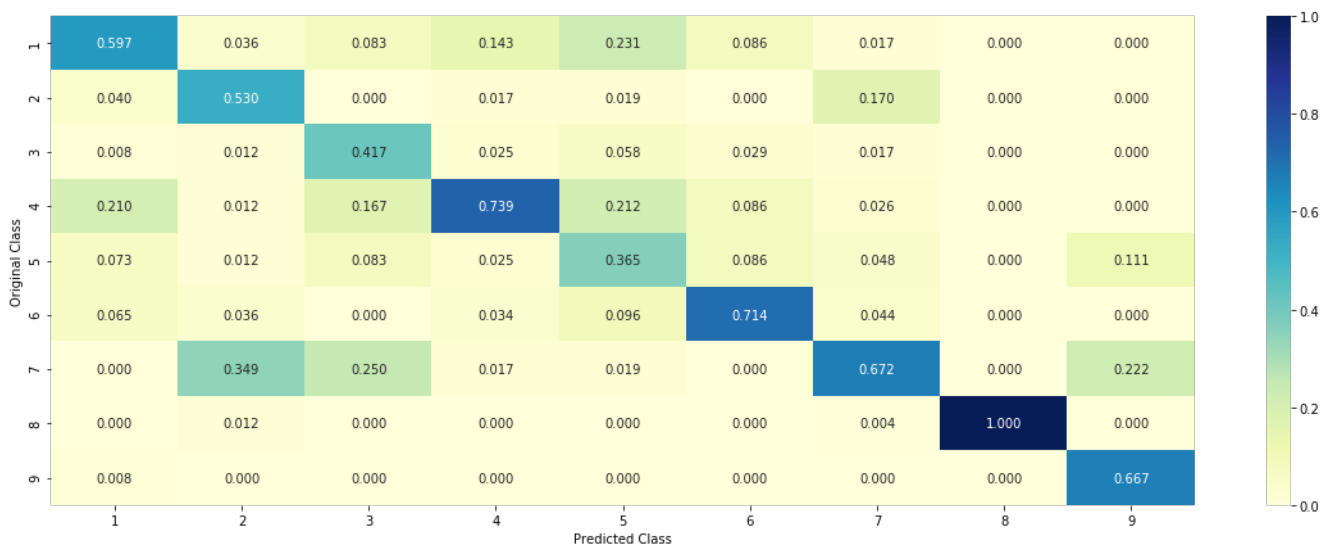
```
Log loss (test) on the VotingClassifier : 1.0839799765935323
```

Number of missclassified point : 0.37293233082706767

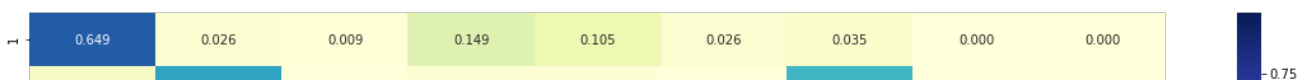
```
----- Confusion matrix -----
```

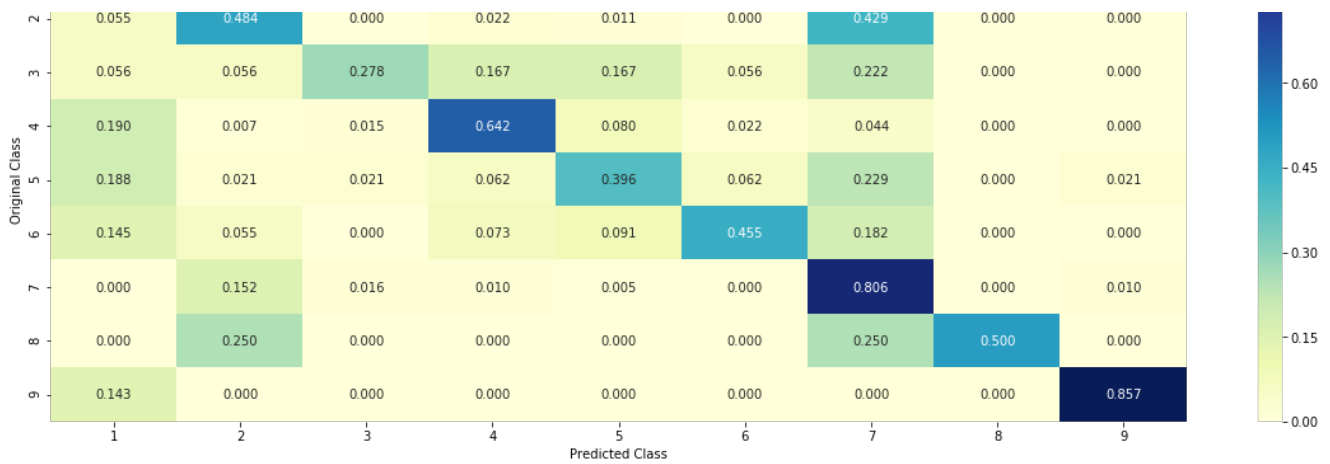


```
----- Precision matrix (Columm Sum=1) -----
```



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





## Summary & Conclusion

In [131]:

```
from IPython.display import Image
img = 'C:/Users/Ghost/Downloads/AAIC Assignment/Feat CS.png'
Image(img, width=50000, height=50000)
```

Out[131]:

With New Features in Dataset and Tfidf Performance				
Classifier	Train Loss	CV Loss	Test Loss	% Miss Class
Naïve Bayes	0.665	1.28	1.29	38.9 %
KNN	0.968	1.25	1.29	43.2 %
Logistic Reg. (Balanced)	0.413	0.986	0.987	33.83 %
Logistic Reg. (W/O Balanced)	0.522	0.977	0.997	34.02 %
Linear SVM	0.47	1.01	1.02	34.77 %
Random Forest (One Hot Encoding)	0.58	1.145	1.141	37.03 %
Random Forest (Response Coding)	0.015	1.96	1.97	73.49 %
Stacking	0.527	1.119	1.13	38.04 %
Max Voting	0.645	1.082	1.083	37.2 %

- 1] Used 7 New Features Like Gene\_Variation, Gene\_Share, Variation\_Share, Word Count, More Than 5000 Words, Character Length, Average Length.
- 2] Logistics Regression with Balanced Dataset obtained the best result using OneHotEncoded Features, Minimum Log Loss that is less than 1 and have 33.83 % of Miss classified point.
- 3] Logistics Regression without Balanced Dataset obtained the Second best result using OneHotEncoded Features, With Log Loss that is less than 1 for train, test, CV and have 34.02 % of Miss classified point.
- 4] Using new features in Dataset definitely have incresed the performance of all the model compare to Bow & Tfidf model performance.
- 5] Random Forest have had bad performance in all the models with Response coding features and still after using new features havent changed anything and have 73.49 % of miss classified point.