

# 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 [1]:

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

from mlxtend.classifier import StackingClassifier

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

Using TensorFlow backend.

```
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or 'ltype' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
    _np_qint8 = np.dtype [("qint8", np.int8, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or 'ltype' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
    _np_quint8 = np.dtype [("quint8", np.uint8, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or 'ltype' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
    _np_qint16 = np.dtype [("qint16", np.int16, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or 'ltype' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
    _np_quint16 = np.dtype [("quint16", np.uint16, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or 'ltype' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
```

```
(1,)) / '(1,)type'.
_np_qint32 = np.dtype [("qint32", np.int32, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or 'ltype' as
a synonym of type is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
_np_resource = np.dtype [("resource", np.ubyte, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or 'lt
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
_np_qint8 = np.dtype [("qint8", np.int8, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or 'lt
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
_np_quint8 = np.dtype [("quint8", np.uint8, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or 'lt
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
_np_qint16 = np.dtype [("qint16", np.int16, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or 'lt
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
_np_quint16 = np.dtype [("quint16", np.uint16, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:545: FutureWarning: Passing (type, 1) or 'lt
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
_np_qint32 = np.dtype [("qint32", np.int32, 1)])
c:\users\acer\appdata\local\programs\python\python37\lib\site-
packages\tensorboard\compat\tensorflow_stub\dtypes.py:550: FutureWarning: Passing (type, 1) or 'lt
ype' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
_np_resource = np.dtype [("resource", np.ubyte, 1)])
```

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

```
Number of data points : 3321
Number of features : 4
Features : ['ID' 'Gene' 'Variation' 'Class']
```

Out[2]:

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

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

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

### 3.1.2. Reading Text Data

In [3]:

```
# note the 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[3]:

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

### 3.1.3. Preprocessing of text

In [4]:

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

def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\\s+', ' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

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

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

In [5]:

```
#text processing stage.
start_time = time.clock()
for index, row in 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 : 35.21483539999997 seconds

```

In [6]:

```

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

```

Out[6]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

In [7]:

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

Out[7]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

In [8]:

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

In [9]:

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

Out[9]:

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

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

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

In [10]:

```

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

```

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

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

In [11]:

```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

### 3.1.4.2. Distribution of y\_i's in Train, Test and Cross Validation datasets

In [12]:

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

my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
train_class_distribution.plot(kind='bar', color = my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

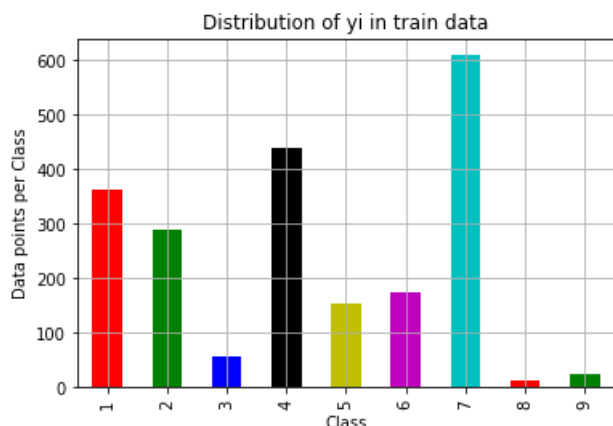
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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

print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color = my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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

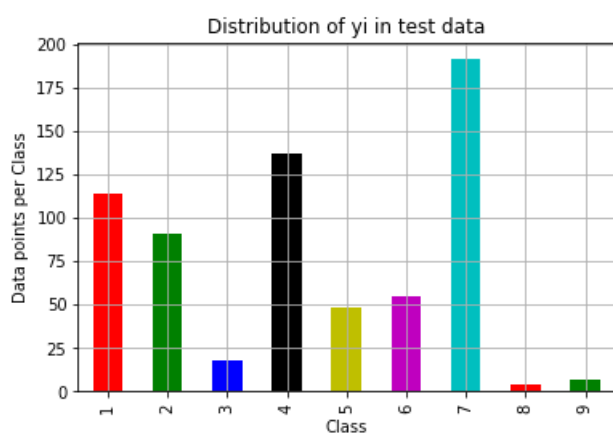
print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color = my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()
```

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



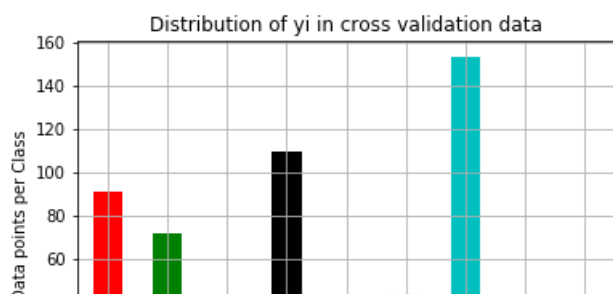
Number of data points in class 7 : 609 ( 28.672 %)  
 Number of data points in class 4 : 439 ( 20.669 %)  
 Number of data points in class 1 : 363 ( 17.09 %)  
 Number of data points in class 2 : 289 ( 13.606 %)  
 Number of data points in class 6 : 176 ( 8.286 %)  
 Number of data points in class 5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

---

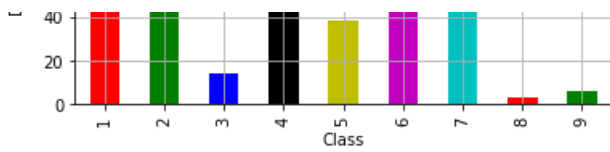


Number of data points in class 7 : 191 ( 28.722 %)  
 Number of data points in class 4 : 137 ( 20.602 %)  
 Number of data points in class 1 : 114 ( 17.143 %)  
 Number of data points in class 2 : 91 ( 13.684 %)  
 Number of data points in class 6 : 55 ( 8.271 %)  
 Number of data points in class 5 : 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 [13]:

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

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

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

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                               [3/7, 4/7]]
    # sum of row elements = 1

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

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

    print("-"*20, "Precision matrix (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 [14]:

```

# 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))

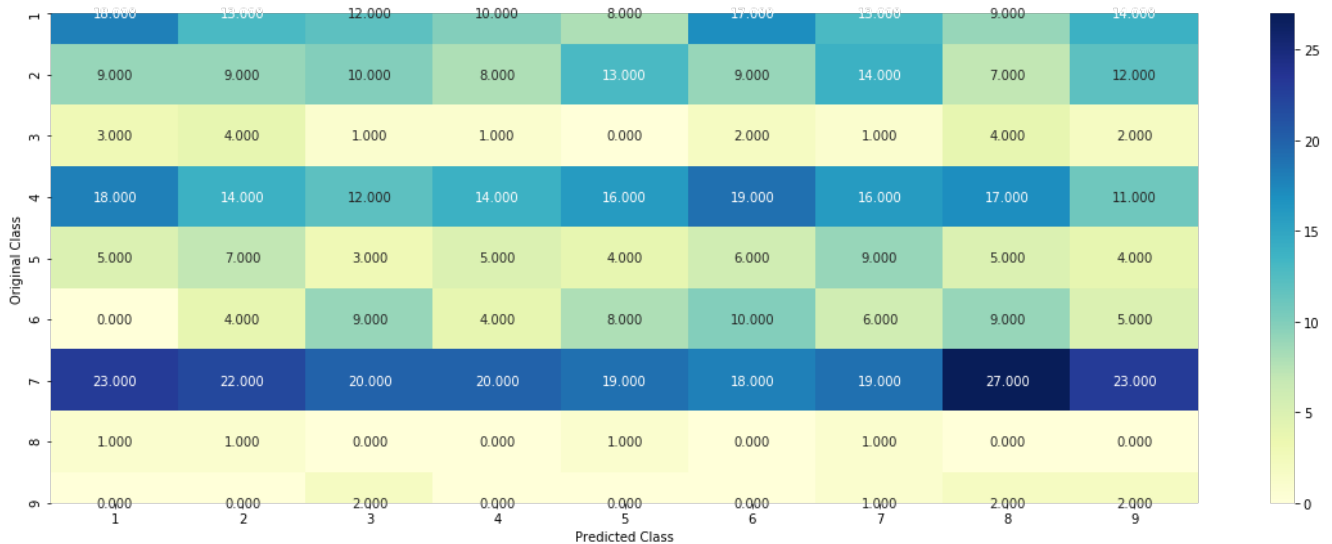
predicted_y = np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

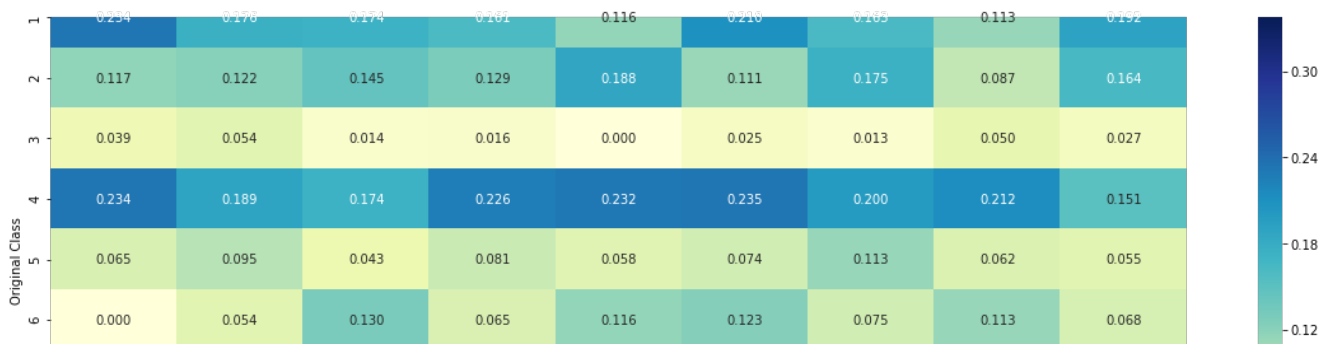
Log loss on Cross Validation Data using Random Model 2.522234920576902

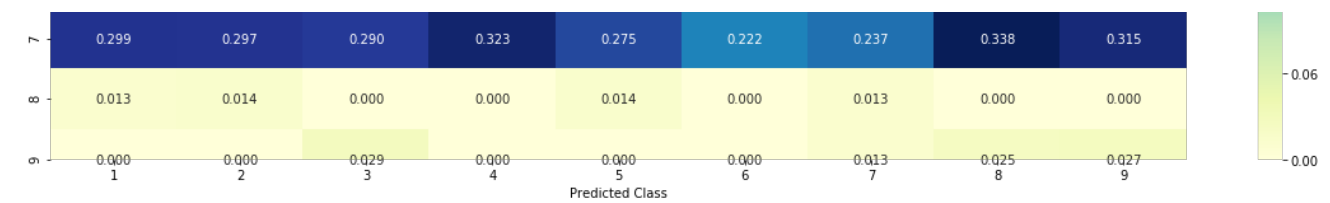
Log loss on Test Data using Random Model 2.452790339094687

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

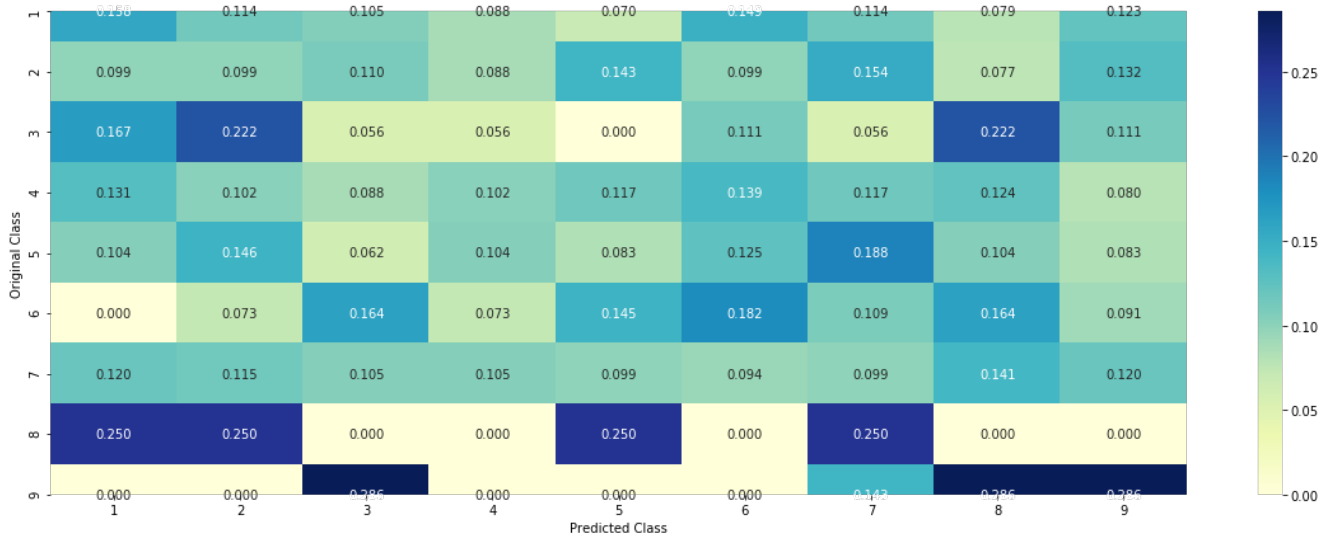


----- Precision matrix (Column Sum=1) -----





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



### 3.3 Univariate Analysis

In [15]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# -----
# Consider all unique values and the number of 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 time that particular feature occurred in whole data
for i, denominator in value_count.items():
    # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
    # vec is 9 dimensional vector
    vec = []
    for k in range(1,10):
        # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
        #
        # ID      Gene      Variation  Class
        # 2470    2470    BRCA1          S1715C      1
        # 2486    2486    BRCA1          S1841R      1
        # 2614    2614    BRCA1          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, 0.068181818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.13939393939393934, 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, 0.09333333333333338, 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([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#
    return gv_fea

```

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

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

### 3.2.1 Univariate Analysis on Gene Feature

**Q1.** Gene, What type of feature it is ?

**Ans.** Gene is a categorical variable

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

In [16]:

```

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

```

Number of Unique Genes : 233

BRCA1	162
TP53	113
EGFR	95
BRCA2	77
PTEN	74
BRAF	62
KIT	57
ALK	46
ERBB2	45
PDGFRA	40

Name: Gene, dtype: int64

In [17]:

```

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

```

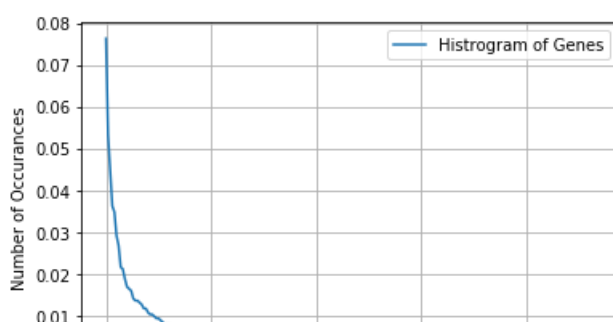
Ans: There are 233 different categories of genes in the train data, and they are distributed as follows

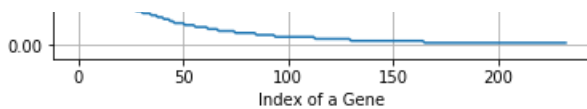
In [18]:

```

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()

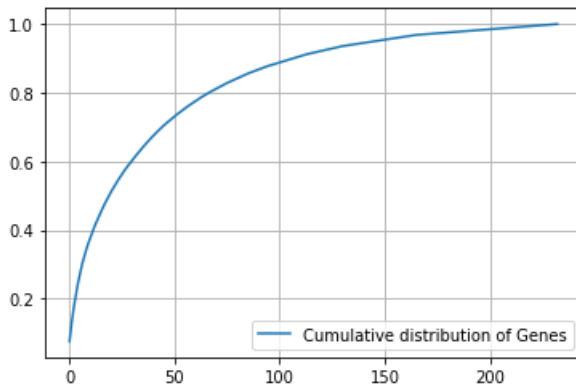
```





In [19]:

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



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

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

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

1. One hot Encoding
2. Response coding

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

In [20]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [21]:

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

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

In [22]:

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

In [23]:

```
In [23]:
```

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

```
Out[23]:
```

```
554      SMAD3
1379     FGFR1
2634     BRCA1
2199      PTEN
2386     PTPN11
Name: Gene, dtype: object
```

```
In [24]:
```

```
#gene_vectorizer.get_feature_names()
```

```
In [25]:
```

```
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, 233)
```

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

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

```
In [26]:
```

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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))

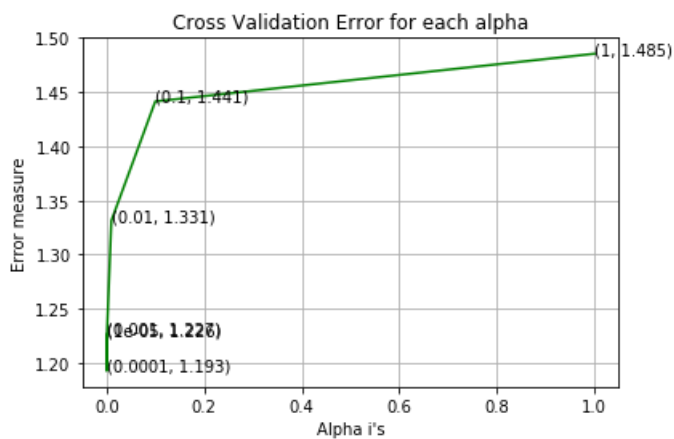
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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: 1.225819165308005
For values of alpha = 0.0001 The log loss is: 1.1926607677293912
For values of alpha = 0.001 The log loss is: 1.22705812621942
For values of alpha = 0.01 The log loss is: 1.33132593738324
For values of alpha = 0.1 The log loss is: 1.4414353277697223
For values of alpha = 1 The log loss is: 1.4854596901941093
```



```
For values of best alpha = 0.0001 The train log loss is: 1.0073996150469482
For values of best alpha = 0.0001 The cross validation log loss is: 1.1926607677293912
For values of best alpha = 0.0001 The test log loss is: 1.19273673001153
```

**Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?**

**Ans.** Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

In [27]:

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

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

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

Q6. How many data points in Test and CV datasets are covered by the 233 genes in train dataset?  
Ans

1. In test data 646 out of 665 : 97.14285714285714
2. In cross validation data 507 out of 532 : 95.30075187969925

### 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 [28]:

```
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 : 1932
Truncating_Mutations      59
Deletion                   47
Amplification              41
Fusions                    23
Overexpression             4
E17K                       3
T167A                      2
EWSR1-ETV1_Fusion          2
ETV6-NTRK3_Fusion          2
Q61R                       2
Name: Variation, dtype: int64
```

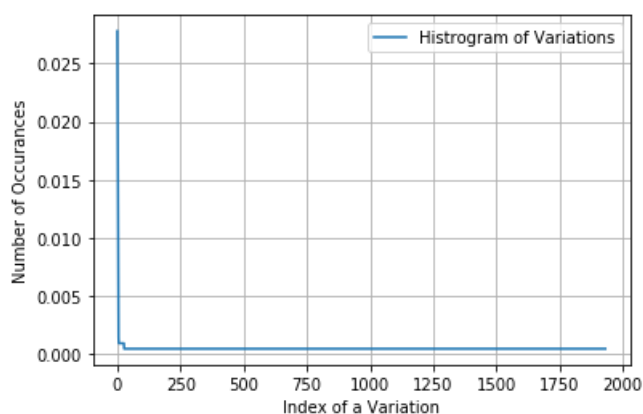
In [29]:

```
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 1932 different categories of variations in the train data, and they are distributed as follows

In [30]:

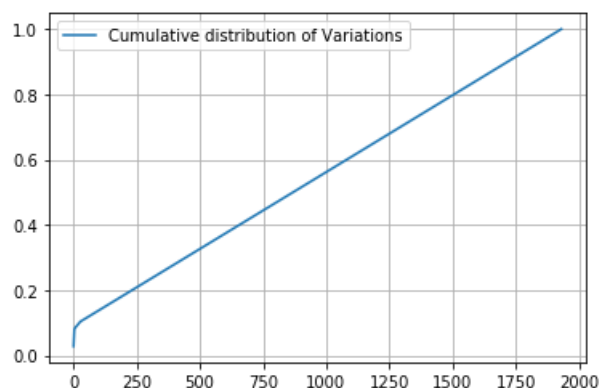
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:

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

```
[0.02777778 0.04990584 0.06920904 ... 0.99905838 0.99952919 1. ]
```



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

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

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

1. One hot Encoding
2. Response coding

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

In [32]:

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

In [33]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding 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 [34]:

```
# 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 [35]:

```
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, 1968)

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

Let's build a model just like the earlier!

In [36]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_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)

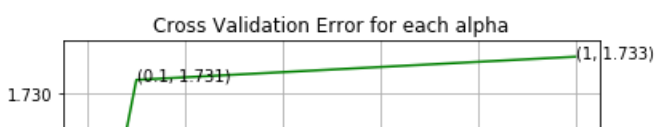
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

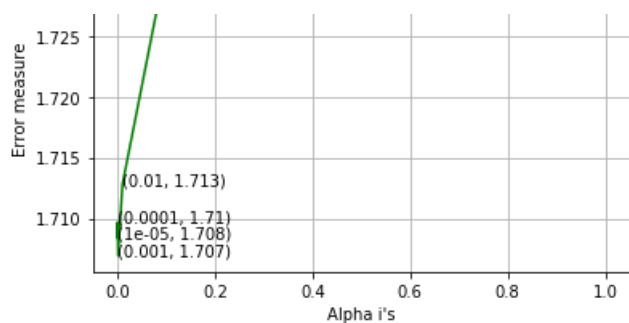
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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: 1.7084153880367634
For values of alpha = 0.0001 The log loss is: 1.7096813529061323
For values of alpha = 0.001 The log loss is: 1.706916046574698
For values of alpha = 0.01 The log loss is: 1.7127993878381382
For values of alpha = 0.1 The log loss is: 1.7310779980893052
For values of alpha = 1 The log loss is: 1.7329792327347957
```





For values of best alpha = 0.001 The train log loss is: 1.1009209292740243  
 For values of best alpha = 0.001 The cross validation log loss is: 1.706916046574698  
 For values of best alpha = 0.001 The test log loss is: 1.7053192376839361

**Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?**

**Ans.** Not sure! But lets be very sure using the below analysis.

In [37]:

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

Q12. How many data points are covered by total 1932 genes in test and cross validation data sets?

Ans

1. In test data 74 out of 665 : 11.12781954887218
2. In cross validation data 54 out of 532 : 10.150375939849624

### 3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicting  $y_i$ ?
5. Is the text feature stable across train, test and CV datasets?

In [38]:

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

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

In [39]:

```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
```

```

        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding

```

In [40]:

```

# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,5), max_features=4000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it 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 : 4000

In [41]:

```

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 [42]:

```

#response coding of text features
train_text_feature_responseCoding = get_text_responsecoding(train_df)
test_text_feature_responseCoding = get_text_responsecoding(test_df)
cv_text_feature_responseCoding = get_text_responsecoding(cv_df)

```

In [43]:

```

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

```

In [44]:

```

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

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

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

```

In [45]:

```

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

```

In [46]:

```

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

```

In [47]:

```

# 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)

```

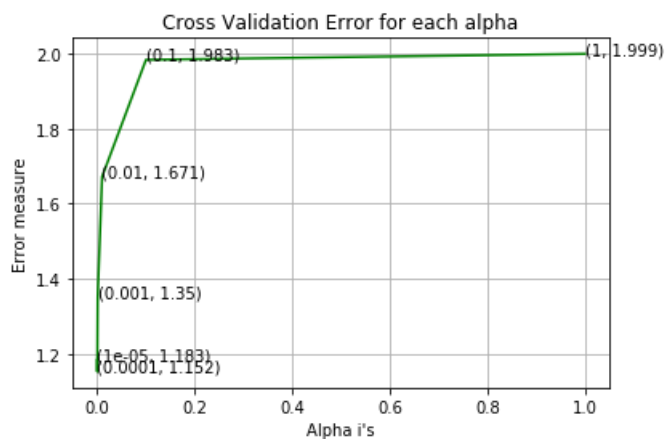
```

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)
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: 1.183426155940956  
 For values of alpha = 0.0001 The log loss is: 1.1520539482797045  
 For values of alpha = 0.001 The log loss is: 1.3495803100341182  
 For values of alpha = 0.01 The log loss is: 1.6713823083594324  
 For values of alpha = 0.1 The log loss is: 1.9829847172629858  
 For values of alpha = 1 The log loss is: 1.9988524154517249



For values of best alpha = 0.0001 The train log loss is: 0.6660818738167186  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.1520539482797045  
 For values of best alpha = 0.0001 The test log loss is: 1.088854237394518

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

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

In [48]:

```

def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=10, ngram_range=(1,5), max_features=3000)
    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 [49]:

```

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")

```

98.7 % of word of test data appeared in train data  
 95.5 % of word of Cross Validation appeared in train data

## 4. Machine Learning Models

In [50]:

```
#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 [51]:

```
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 [52]:

```
# 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, ngram_range=(1,5), max_features=3000)

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

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

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

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



## Stacking the three types of features

In [53]:

```
# merging gene, variance and text features

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

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

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

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

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

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

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

In [54]:

```
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, 6201)
(number of data points * number of features) in test data = (665, 6201)
(number of data points * number of features) in cross validation data = (532, 6201)
```

In [55]:

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

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

## 4.1. Base Line Model

### 4.1.1. Naive Bayes

#### 4.1.1.1. Hyper parameter tuning

In [112]:

```
# 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 probabilities we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

```

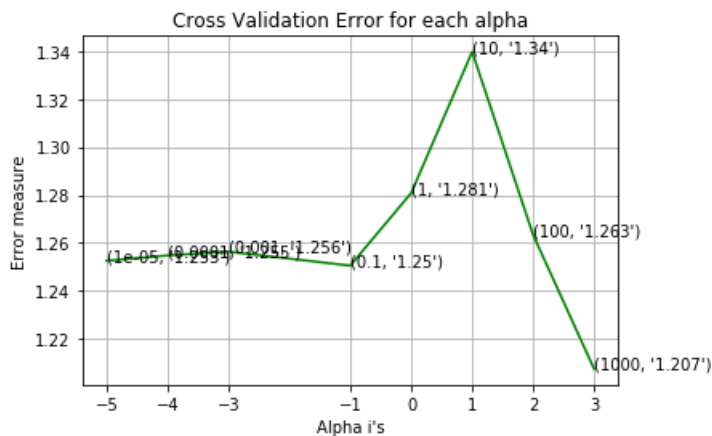
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_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 : 1.2525070426162148
for alpha = 0.0001
Log Loss : 1.2547155986627674
for alpha = 0.001
Log Loss : 1.2563224200135392
for alpha = 0.1
Log Loss : 1.2504806987067105
for alpha = 1
Log Loss : 1.2808019309035574
for alpha = 10
Log Loss : 1.3397981540878574
for alpha = 100
Log Loss : 1.2633371391431119
for alpha = 1000
Log Loss : 1.2073752432083387

```



```

For values of best alpha = 1000 The train log loss is: 0.9156058233155762
For values of best alpha = 1000 The cross validation log loss is: 1.2073752432083387
For values of best alpha = 1000 The test log loss is: 1.2070256615007953

```

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

In [60]:

```

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

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

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

```

```

# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# -----

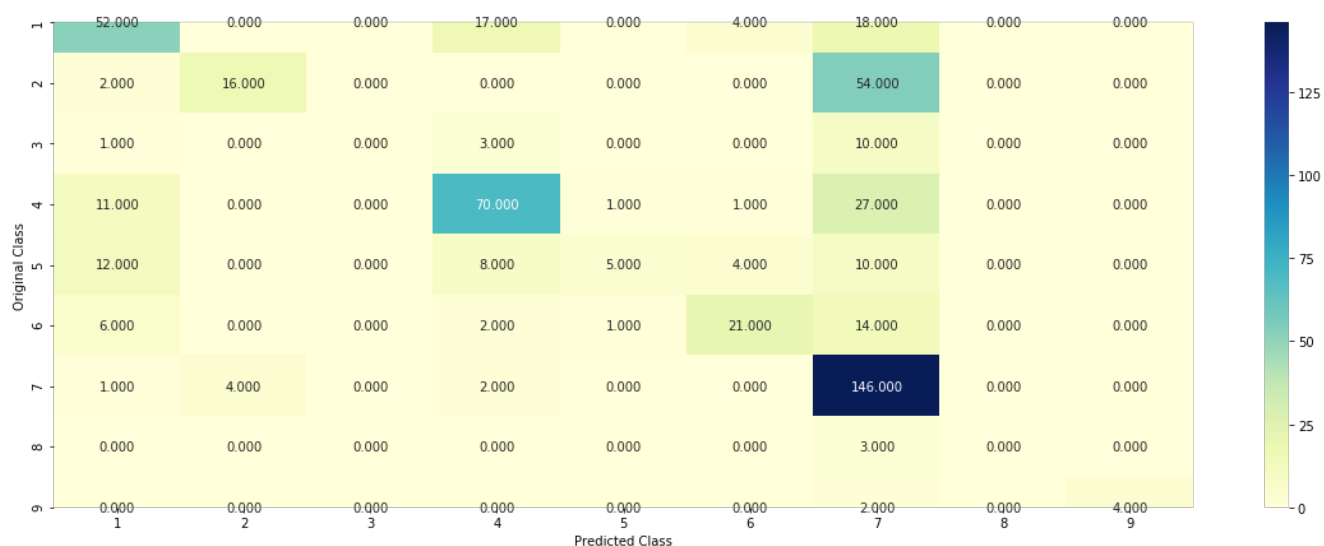
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilities we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

```

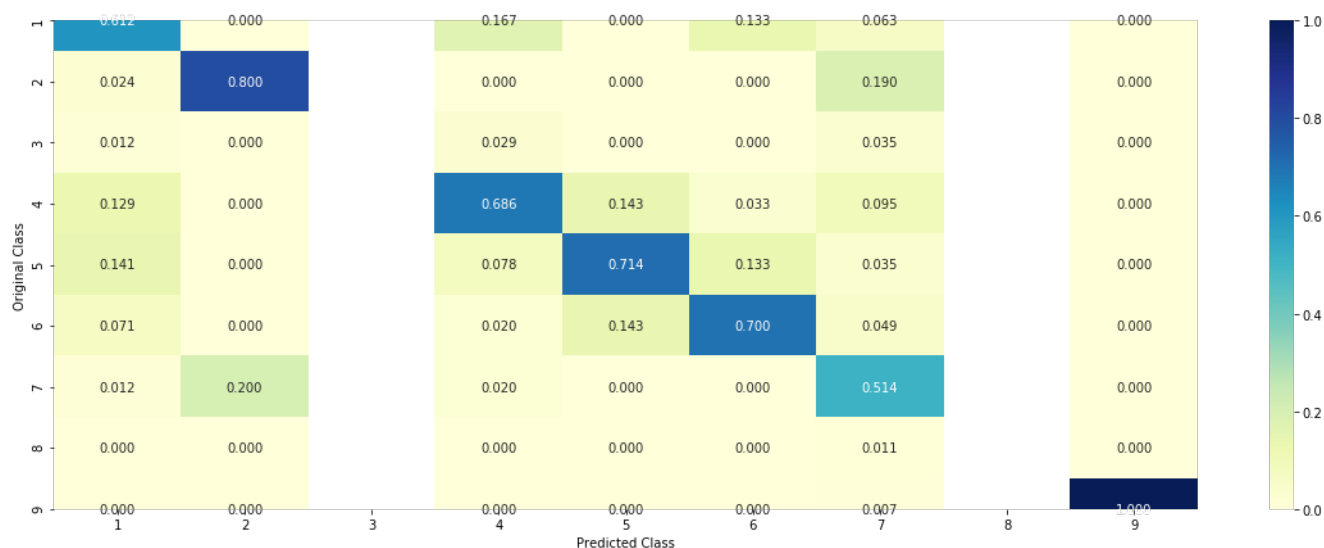
Log Loss : 1.2073752432083387

Number of missclassified point : 0.40977443609022557

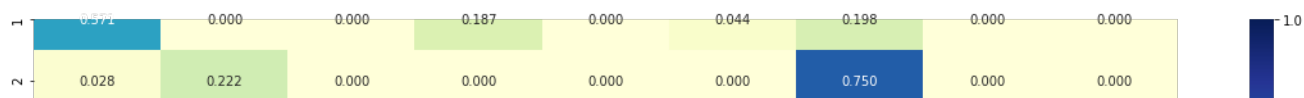
----- Confusion matrix -----

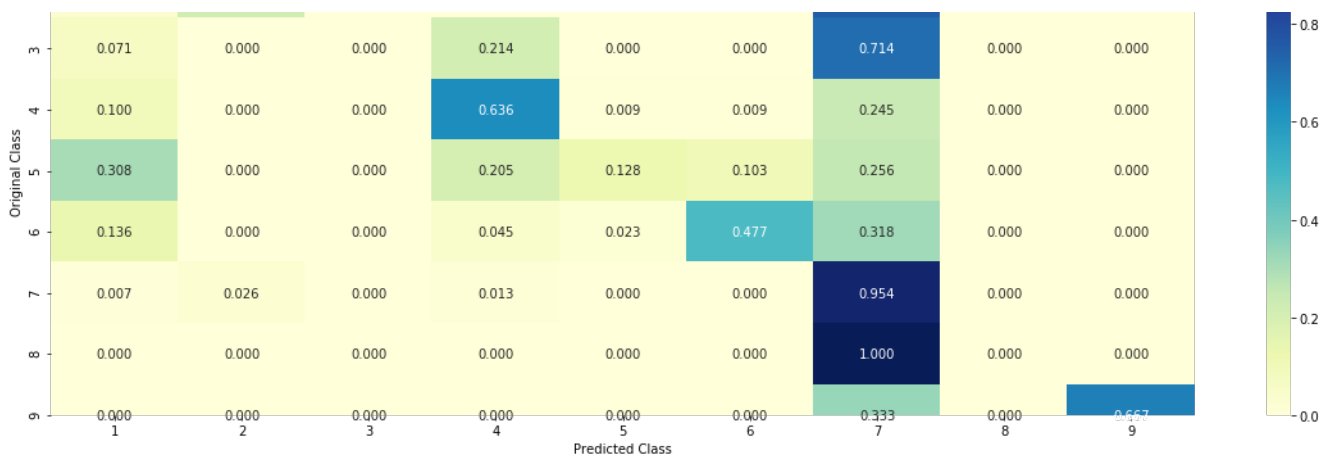


----- Precision matrix (Column Sum=1) -----



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





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

In [64]:

```
test_point_index = 7
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

Predicted Class : 7

Predicted Class Probabilities: [[4.900e-03 4.675e-01 2.700e-03 5.800e-03 1.040e-02 7.000e-03 5.000e-01

1.400e-03 2.000e-04]]

Actual Class : 7

```
-----
14 Text feature [cells] present in test data point [True]
16 Text feature [activated] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [cell] present in test data point [True]
19 Text feature [activation] present in test data point [True]
20 Text feature [contrast] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
22 Text feature [factor] present in test data point [True]
23 Text feature [presence] present in test data point [True]
24 Text feature [phosphorylation] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [shown] present in test data point [True]
27 Text feature [growth] present in test data point [True]
28 Text feature [inhibitor] present in test data point [True]
29 Text feature [also] present in test data point [True]
30 Text feature [recently] present in test data point [True]
31 Text feature [however] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [signaling] present in test data point [True]
38 Text feature [increased] present in test data point [True]
39 Text feature [suggest] present in test data point [True]
40 Text feature [3b] present in test data point [True]
41 Text feature [similar] present in test data point [True]
42 Text feature [independent] present in test data point [True]
43 Text feature [addition] present in test data point [True]
44 Text feature [previously] present in test data point [True]
45 Text feature [1a] present in test data point [True]
46 Text feature [potential] present in test data point [True]
47 Text feature [compared] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [treated] present in test data point [True]
50 Text feature [figure] present in test data point [True]
51 Text feature [interestingly] present in test data point [True]
52 Text feature [mechanism] present in test data point [True]
```

```

52 Text feature [mechanism] present in test data point [True]
53 Text feature [antibodies] present in test data point [True]
54 Text feature [showed] present in test data point [True]
55 Text feature [constitutive] present in test data point [True]
56 Text feature [various] present in test data point [True]
58 Text feature [found] present in test data point [True]
59 Text feature [tyrosine] present in test data point [True]
61 Text feature [demonstrated] present in test data point [True]
62 Text feature [described] present in test data point [True]
66 Text feature [using] present in test data point [True]
68 Text feature [consistent] present in test data point [True]
69 Text feature [total] present in test data point [True]
70 Text feature [observed] present in test data point [True]
71 Text feature [followed] present in test data point [True]
72 Text feature [3a] present in test data point [True]
73 Text feature [may] present in test data point [True]
74 Text feature [sensitive] present in test data point [True]
75 Text feature [treatment] present in test data point [True]
76 Text feature [mutations] present in test data point [True]
77 Text feature [without] present in test data point [True]
78 Text feature [fig] present in test data point [True]
79 Text feature [higher] present in test data point [True]
80 Text feature [obtained] present in test data point [True]
81 Text feature [proliferation] present in test data point [True]
82 Text feature [anti] present in test data point [True]
83 Text feature [absence] present in test data point [True]
84 Text feature [mutant] present in test data point [True]
85 Text feature [pathways] present in test data point [True]
86 Text feature [respectively] present in test data point [True]
87 Text feature [including] present in test data point [True]
88 Text feature [reported] present in test data point [True]
89 Text feature [approximately] present in test data point [True]
90 Text feature [activating] present in test data point [True]
91 Text feature [expression] present in test data point [True]
92 Text feature [induced] present in test data point [True]
93 Text feature [increase] present in test data point [True]
94 Text feature [two] present in test data point [True]
95 Text feature [identified] present in test data point [True]
97 Text feature [furthermore] present in test data point [True]
98 Text feature [inhibition] present in test data point [True]
99 Text feature [domain] present in test data point [True]
Out of the top 100 features 74 are present in query point

```

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

In [63]:

```

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

```

```

Predicted Class : 2
Predicted Class Probabilities: [[5.900e-03 6.065e-01 3.500e-03 8.500e-03 1.230e-02 7.900e-03 3.533
e-01
1.700e-03 3.000e-04]]
Actual Class : 7
-----
10 Text feature [treatment] present in test data point [True]
13 Text feature [patients] present in test data point [True]
14 Text feature [clinical] present in test data point [True]
15 Text feature [response] present in test data point [True]
16 Text feature [therapy] present in test data point [True]
17 Text feature [molecular] present in test data point [True]
18 Text feature [time] present in test data point [True]
23 Text feature [advanced] present in test data point [True]
24 Text feature [studv] present in test data point [True]

```

```

25 Text feature [patient] present in test data point [True]
26 Text feature [first] present in test data point [True]
27 Text feature [11] present in test data point [True]
28 Text feature [months] present in test data point [True]
29 Text feature [10] present in test data point [True]
30 Text feature [mutation] present in test data point [True]
31 Text feature [confirmed] present in test data point [True]
32 Text feature [achieved] present in test data point [True]
33 Text feature [case] present in test data point [True]
34 Text feature [treated] present in test data point [True]
35 Text feature [therapeutic] present in test data point [True]
36 Text feature [however] present in test data point [True]
37 Text feature [reported] present in test data point [True]
38 Text feature [13] present in test data point [True]
39 Text feature [including] present in test data point [True]
40 Text feature [different] present in test data point [True]
41 Text feature [17] present in test data point [True]
42 Text feature [primary] present in test data point [True]
43 Text feature [mutations] present in test data point [True]
44 Text feature [15] present in test data point [True]
45 Text feature [chronic] present in test data point [True]
46 Text feature [another] present in test data point [True]
47 Text feature [kinase] present in test data point [True]
48 Text feature [also] present in test data point [True]
49 Text feature [progression] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [recently] present in test data point [True]
52 Text feature [median] present in test data point [True]
53 Text feature [small] present in test data point [True]
54 Text feature [longer] present in test data point [True]
55 Text feature [respectively] present in test data point [True]
56 Text feature [19] present in test data point [True]
58 Text feature [higher] present in test data point [True]
59 Text feature [tissue] present in test data point [True]
60 Text feature [18] present in test data point [True]
61 Text feature [initial] present in test data point [True]
63 Text feature [using] present in test data point [True]
64 Text feature [12] present in test data point [True]
65 Text feature [inhibitor] present in test data point [True]
67 Text feature [analysis] present in test data point [True]
68 Text feature [still] present in test data point [True]
69 Text feature [complete] present in test data point [True]
70 Text feature [common] present in test data point [True]
71 Text feature [development] present in test data point [True]
72 Text feature [performed] present in test data point [True]
73 Text feature [27] present in test data point [True]
74 Text feature [identified] present in test data point [True]
75 Text feature [studies] present in test data point [True]
76 Text feature [new] present in test data point [True]
77 Text feature [due] present in test data point [True]
78 Text feature [pcr] present in test data point [True]
79 Text feature [one] present in test data point [True]
80 Text feature [overall] present in test data point [True]
81 Text feature [sequencing] present in test data point [True]
82 Text feature [disease] present in test data point [True]
84 Text feature [best] present in test data point [True]
85 Text feature [observed] present in test data point [True]
86 Text feature [present] present in test data point [True]
87 Text feature [gene] present in test data point [True]
88 Text feature [found] present in test data point [True]
89 Text feature [cases] present in test data point [True]
90 Text feature [second] present in test data point [True]
91 Text feature [table] present in test data point [True]
92 Text feature [positive] present in test data point [True]
93 Text feature [previously] present in test data point [True]
94 Text feature [20] present in test data point [True]
95 Text feature [subsequently] present in test data point [True]
96 Text feature [approved] present in test data point [True]
97 Text feature [presence] present in test data point [True]
98 Text feature [report] present in test data point [True]
99 Text feature [specific] present in test data point [True]
Out of the top 100 features 80 are present in query point

```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

In [65]:

```
# 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 probabilitites 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
```



```

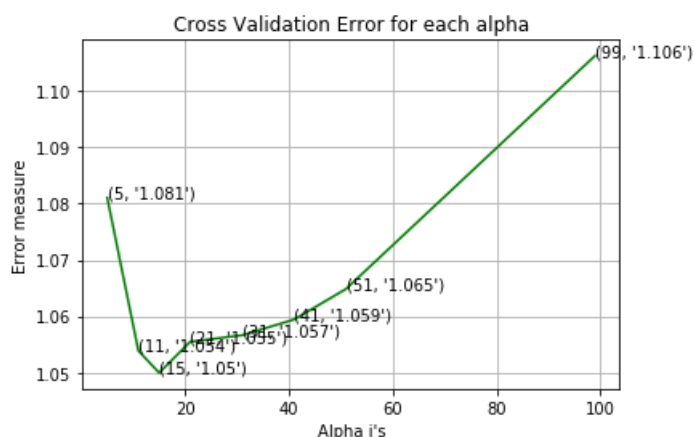
predict_y, labels=clf.classes_, eps=1e-15))

```

```

for alpha = 5
Log Loss : 1.0810425418038823
for alpha = 11
Log Loss : 1.0538949651367326
for alpha = 15
Log Loss : 1.0499630865513279
for alpha = 21
Log Loss : 1.0554891451572397
for alpha = 31
Log Loss : 1.0566357184807638
for alpha = 41
Log Loss : 1.0594280920852348
for alpha = 51
Log Loss : 1.0648424915270216
for alpha = 99
Log Loss : 1.1062304448086284

```



For values of best alpha = 15 The train log loss is: 0.7121247745006319  
 For values of best alpha = 15 The cross validation log loss is: 1.0499630865513279  
 For values of best alpha = 15 The test log loss is: 1.0540129658574022

## 4.2.2. Testing the model with best hyper paramters

In [66]:

```

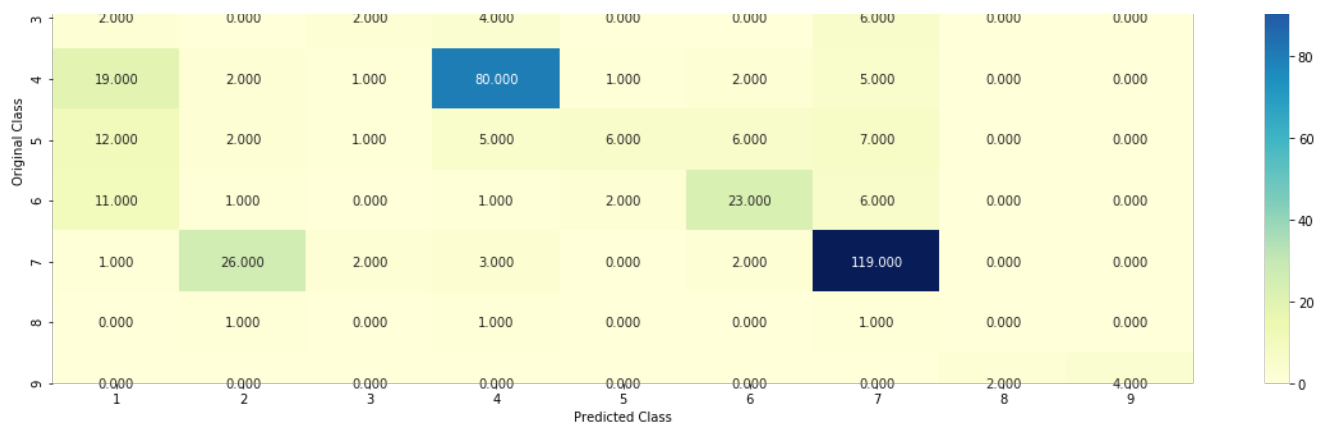
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None,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

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

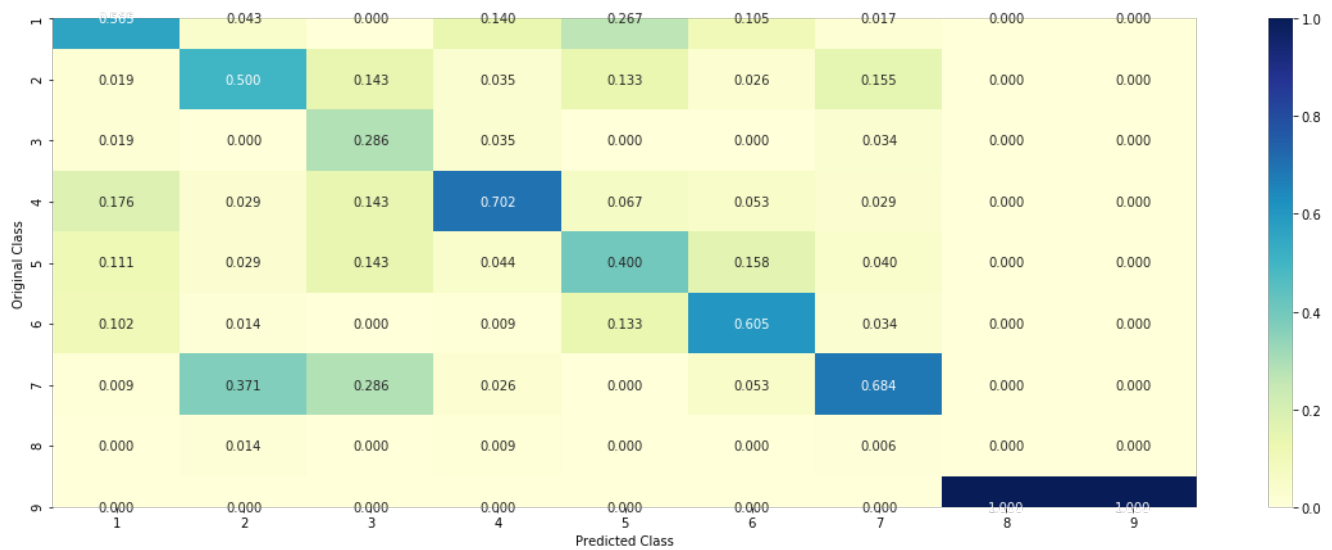
```

Log loss : 1.0499630865513279  
 Number of mis-classified points : 0.37969924812030076  
 ----- Confusion matrix -----

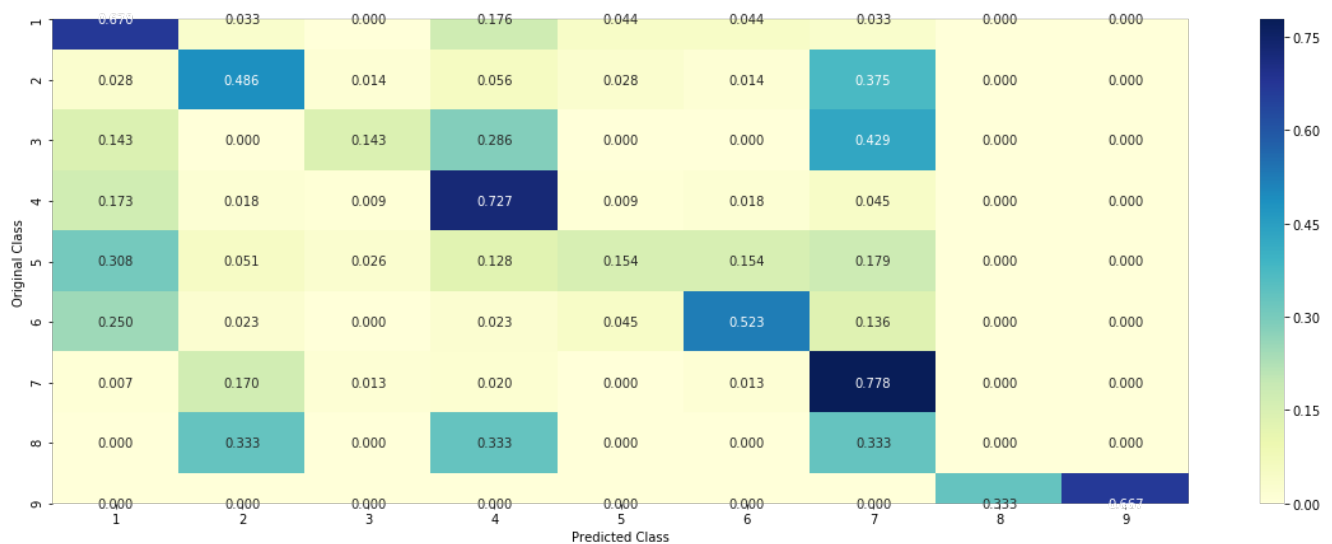
1	01.000	3.000	0.000	16.000	4.000	4.000	3.000	0.000	0.000
2	2.000	35.000	1.000	4.000	2.000	1.000	27.000	0.000	0.000



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



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



#### 4.2.3.Sample Query point -1

In [67]:

```
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

```

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 ",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 : 1  
 The 15 nearest neighbours of the test points belongs to classes [1 4 1 1 7 4 3 1 4 4 4 4 5 1 4]  
 Frequency of nearest points : Counter({4: 7, 1: 5, 7: 1, 3: 1, 5: 1})

## 4.2.4. Sample Query Point-2

In [68]:

```

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 : 2  
 Actual Class : 7  
 the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [2 2 2 7 2 2 7 7 2 7 2 2 2 2 2]  
 Frequency of nearest points : Counter({2: 11, 7: 4})

## 4.3. Logistic Regression

### 4.3.1. With Class balancing

#### 4.3.1.1. Hyper parameter tuning

In [56]:

```

# 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-in-tuition-1/
#-----

```

```

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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)

    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilties 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)

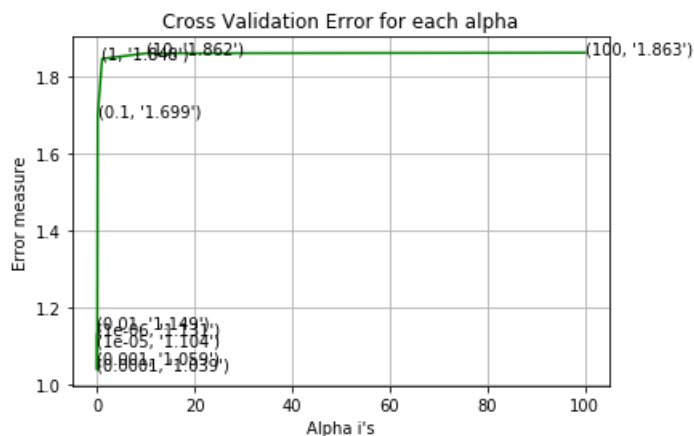
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "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 : 1.1307822833517576
for alpha = 1e-05
Log Loss : 1.1040512549078758
for alpha = 0.0001
Log Loss : 1.0394699812212884
for alpha = 0.001
Log Loss : 1.0587926545565625
for alpha = 0.01
Log Loss : 1.1492265128468704
for alpha = 0.1
Log Loss : 1.6990536554383329
for alpha = 1
Log Loss : 1.8476109655516277
for alpha = 10
Log Loss : 1.8616631699901065
for alpha = 100
Log Loss : 1.8631641107890782

```



For values of best alpha = 0.0001 The train log loss is: 0.4247406330571905  
 For values of best alpha = 0.0001 The cross validation log loss is: 1.0394699812212884  
 For values of best alpha = 0.0001 The test log loss is: 0.9904334370219134

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

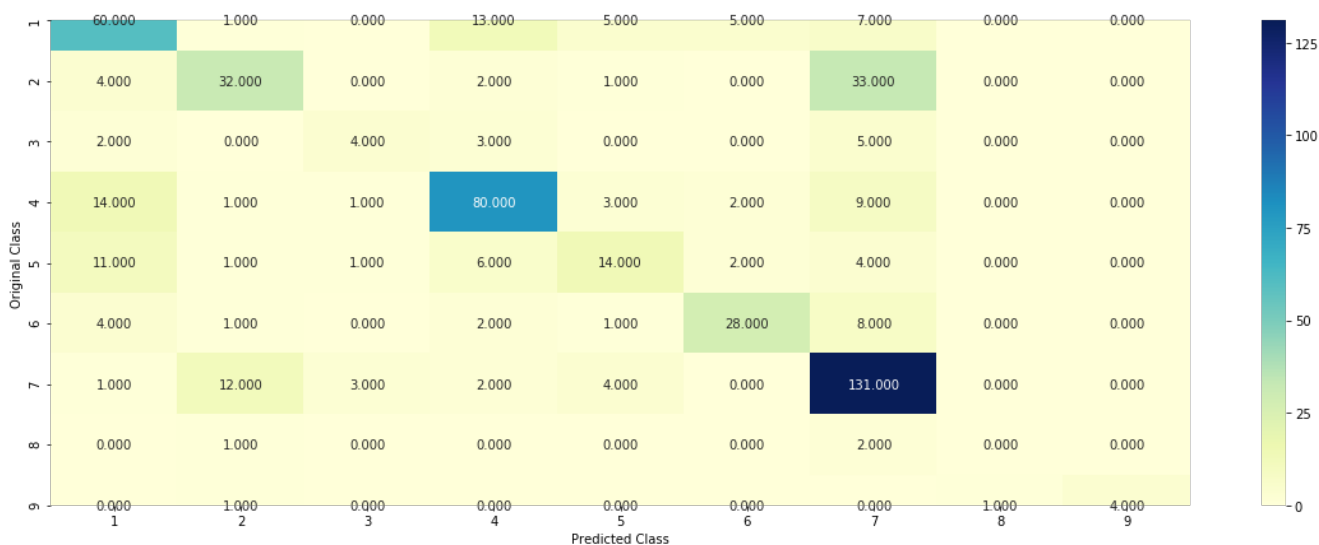
In [70]:

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

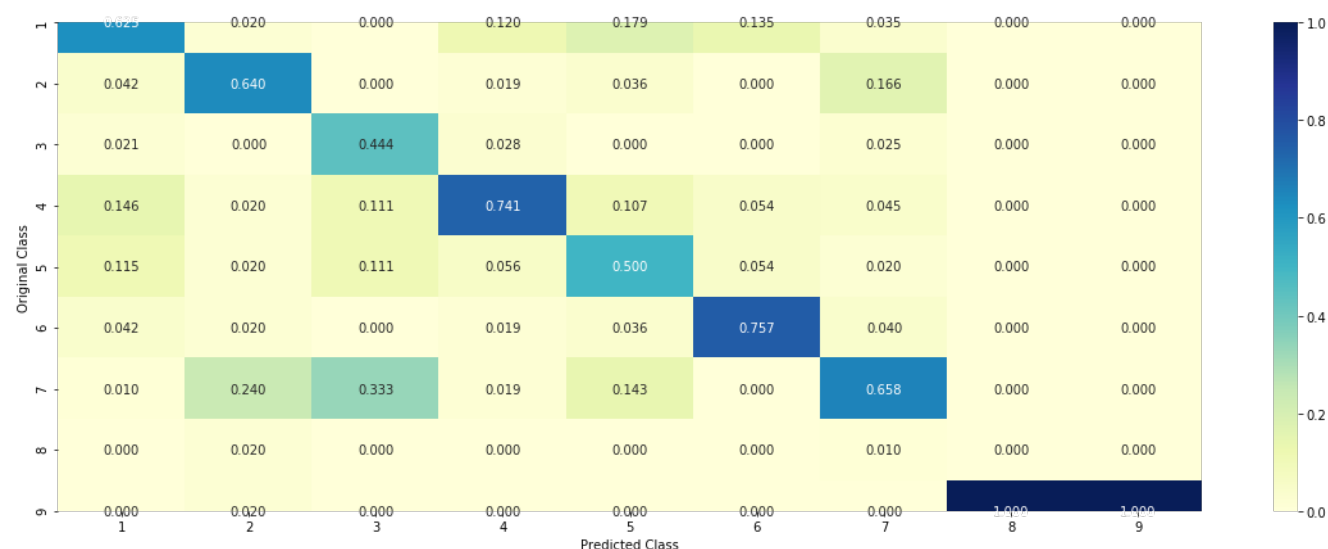
# some of methods
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

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

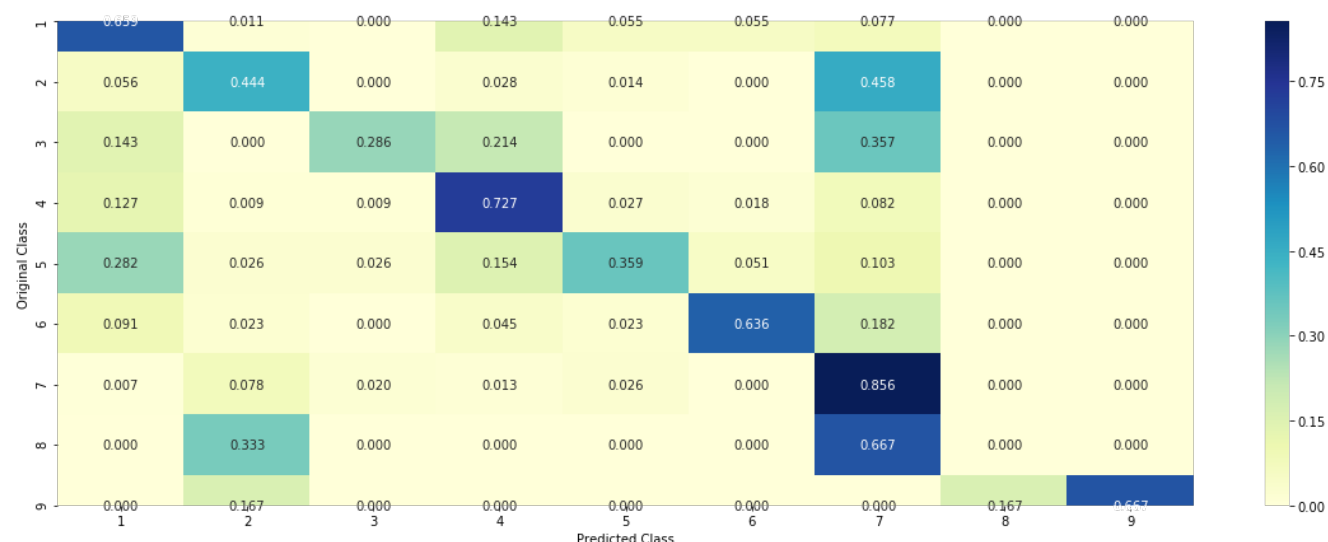
Log loss : 1.0474849293245414  
 Number of mis-classified points : 0.33646616541353386  
 ----- Confusion matrix -----



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



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



#### 4.3.1.3. Feature Importance

In [71]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i < 18:
            tabulte_list.append([incresingorder_ind, "Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind, train_text_features[i], yes_no])
            incresingorder_ind += 1
    print(word_present, "most 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 [72]:

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

Predicted Class : 4

Predicted Class Probabilities: [[0.1305 0.084 0.0189 0.5243 0.0418 0.0263 0.1609 0.0054 0.0078]]

Actual Class : 1

-----

47 Text feature [jsi] present in test data point [True]  
210 Text feature [epitect] present in test data point [True]  
211 Text feature [suppressor] present in test data point [True]  
220 Text feature [pstat3] present in test data point [True]  
433 Text feature [pten] present in test data point [True]  
448 Text feature [localization] present in test data point [True]  
482 Text feature [laryngeal] present in test data point [True]  
Out of the top 500 features 7 are present in query point

#### 4.3.1.3.2. Incorrectly Classified point

In [73]:

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

Predicted Class : 2

Predicted Class Probabilities: [[0.0021 0.9452 0.0019 0.0023 0.004 0.0011 0.0313 0.0107 0.0014]]

Actual Class : 7

-----

70 Text feature [5v] present in test data point [True]  
142 Text feature [chronic] present in test data point [True]  
222 Text feature [md] present in test data point [True]  
252 Text feature [achieved] present in test data point [True]  
316 Text feature [advanced] present in test data point [True]  
394 Text feature [surgical] present in test data point [True]  
Out of the top 500 features 6 are present in query point

### 4.3.2. Without Class balancing

#### 4.3.2.1. Hyper parameter tuning

In [184]:

```

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=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
7 1 0.6713886625002474

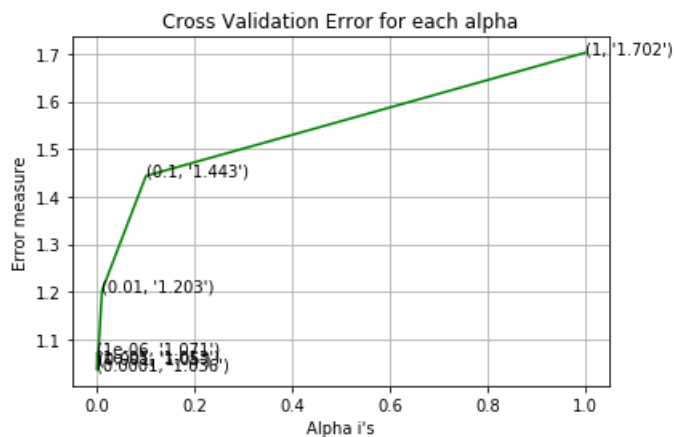
```



```

Log Loss : 1.0713096635283474
for alpha = 1e-05
Log Loss : 1.0530466405148637
for alpha = 0.0001
Log Loss : 1.0363876416667275
for alpha = 0.001
Log Loss : 1.0504687264184296
for alpha = 0.01
Log Loss : 1.2033375584492598
for alpha = 0.1
Log Loss : 1.4434452153965283
for alpha = 1
Log Loss : 1.701971492906462

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4151502510610657
For values of best alpha = 0.0001 The cross validation log loss is: 1.0363876416667275
For values of best alpha = 0.0001 The test log loss is: 1.035816594976169

```

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

In [75]:

```

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

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

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

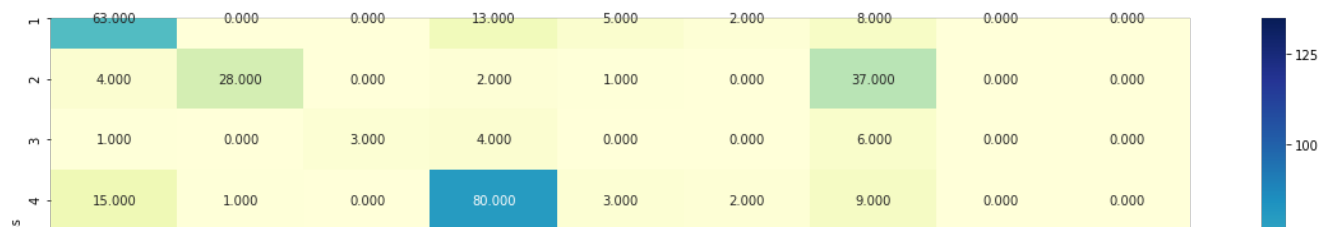
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

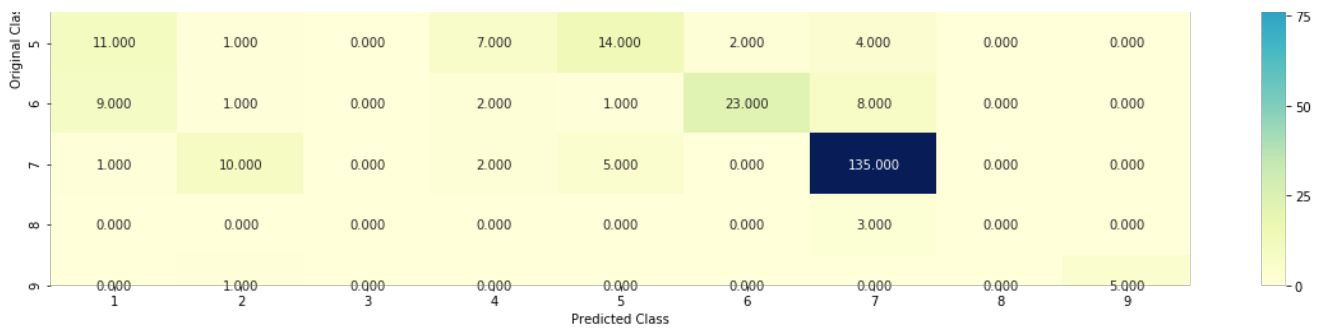
```

```

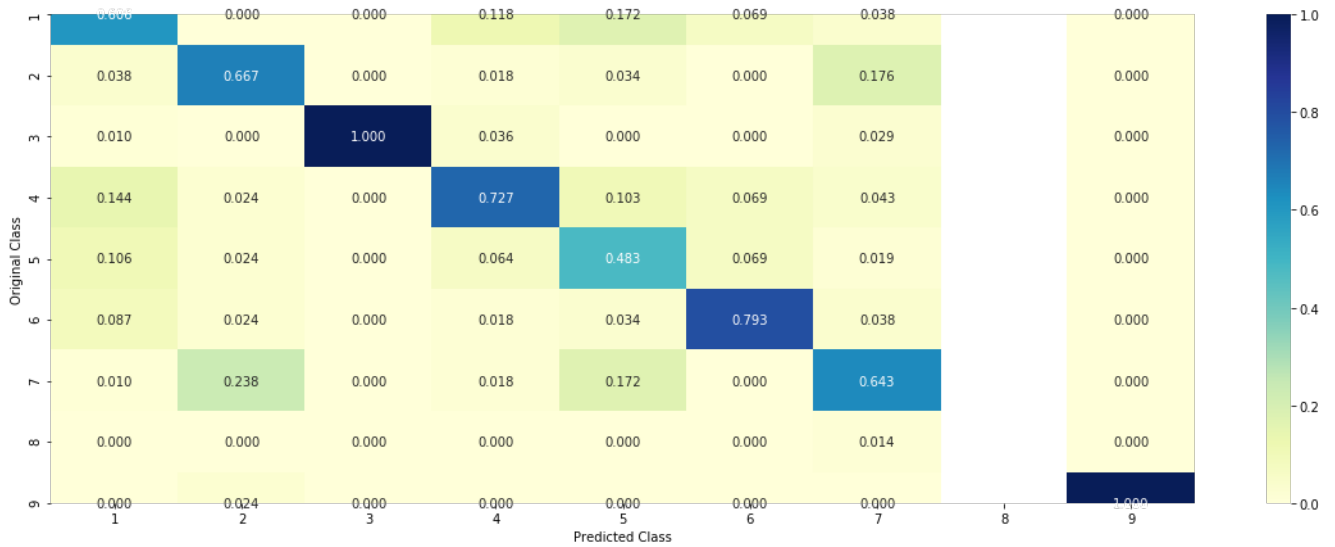
Log loss : 1.0395923451750153
Number of mis-classified points : 0.34022556390977443
----- Confusion matrix -----

```

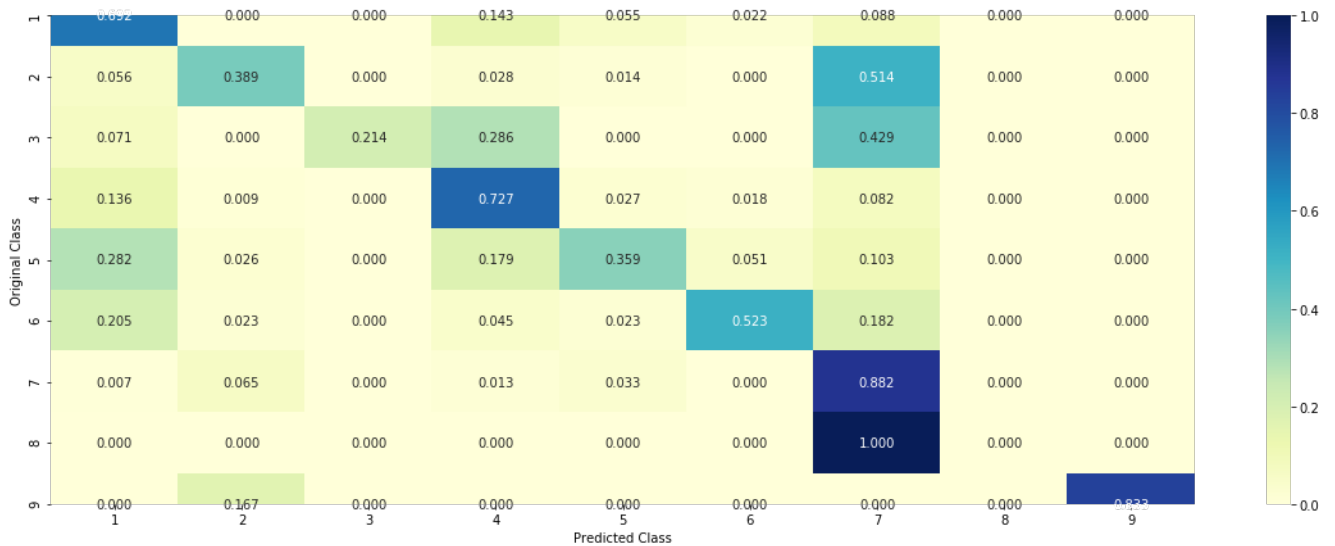




----- Precision matrix (Column Sum=1) -----



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



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

In [77]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 7
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
      np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 4))
print("Actual Class :", test_y[test_point_index])
```

```

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

```

```

Predicted Class : 7
Predicted Class Probabilities: [[4.800e-03 4.676e-01 2.900e-03 4.000e-03 6.300e-03 2.600e-03 5.091
e-01
2.600e-03 2.000e-04]]
Actual Class : 7
-----
63 Text feature [activated] present in test data point [True]
73 Text feature [constitutive] present in test data point [True]
78 Text feature [technology] present in test data point [True]
86 Text feature [mitogen] present in test data point [True]
104 Text feature [ligand] present in test data point [True]
148 Text feature [oncogene] present in test data point [True]
150 Text feature [activation] present in test data point [True]
156 Text feature [tyr204] present in test data point [True]
162 Text feature [mapk] present in test data point [True]
189 Text feature [thr202] present in test data point [True]
201 Text feature [gist882] present in test data point [True]
209 Text feature [expressing] present in test data point [True]
221 Text feature [phosphorylation] present in test data point [True]
239 Text feature [kinase] present in test data point [True]
249 Text feature [phospho] present in test data point [True]
254 Text feature [downstream] present in test data point [True]
310 Text feature [transformed] present in test data point [True]
333 Text feature [f3] present in test data point [True]
336 Text feature [ba] present in test data point [True]
344 Text feature [subcutaneous] present in test data point [True]
361 Text feature [erk] present in test data point [True]
379 Text feature [factor] present in test data point [True]
419 Text feature [activating] present in test data point [True]
422 Text feature [inhibitor] present in test data point [True]
426 Text feature [through] present in test data point [True]
427 Text feature [3b] present in test data point [True]
459 Text feature [physician] present in test data point [True]
462 Text feature [oncogenic] present in test data point [True]
Out of the top 500 features 28 are present in query point

```

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

In [78]:

```

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

```

```

Predicted Class : 2
Predicted Class Probabilities: [[2.500e-03 9.451e-01 1.300e-03 2.900e-03 3.300e-03 1.000e-03 4.150
e-02
2.300e-03 1.000e-04]]
Actual Class : 7
-----
72 Text feature [5v] present in test data point [True]
149 Text feature [chronic] present in test data point [True]
223 Text feature [md] present in test data point [True]
274 Text feature [achieved] present in test data point [True]
305 Text feature [advanced] present in test data point [True]
390 Text feature [surgical] present in test data point [True]
Out of the top 500 features 6 are present in query point

```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper paramter tuning

In [79]:

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

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, 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 paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-----
# video link:
#-----

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

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

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

```

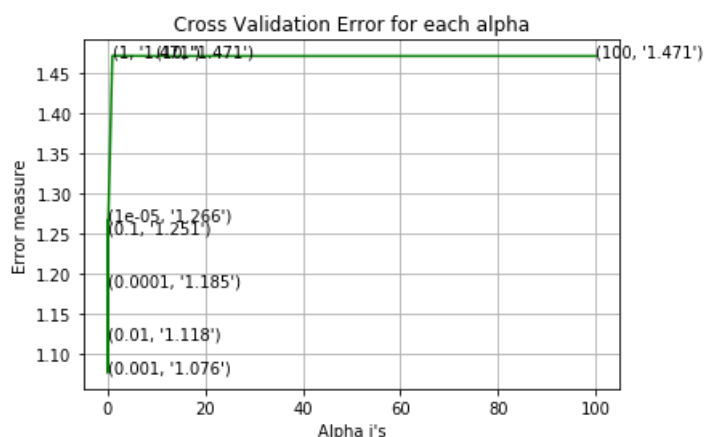
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for C = 1e-05
Log Loss : 1.2660427971526638
for C = 0.0001
Log Loss : 1.1848593192448167
for C = 0.001
Log Loss : 1.0760880535058295
for C = 0.01
Log Loss : 1.1176983682908839
for C = 0.1
Log Loss : 1.2512802604291182
for C = 1
Log Loss : 1.4714035245025547
for C = 10
Log Loss : 1.4710540879250138
for C = 100
Log Loss : 1.4710536630803406

```



```

For values of best alpha = 0.001 The train log loss is: 0.5888493871411611
For values of best alpha = 0.001 The cross validation log loss is: 1.0760880535058295
For values of best alpha = 0.001 The test log loss is: 1.1191067824915681

```

## 4.4.2. Testing model with best hyper parameters

In [80]:

```

# 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.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')

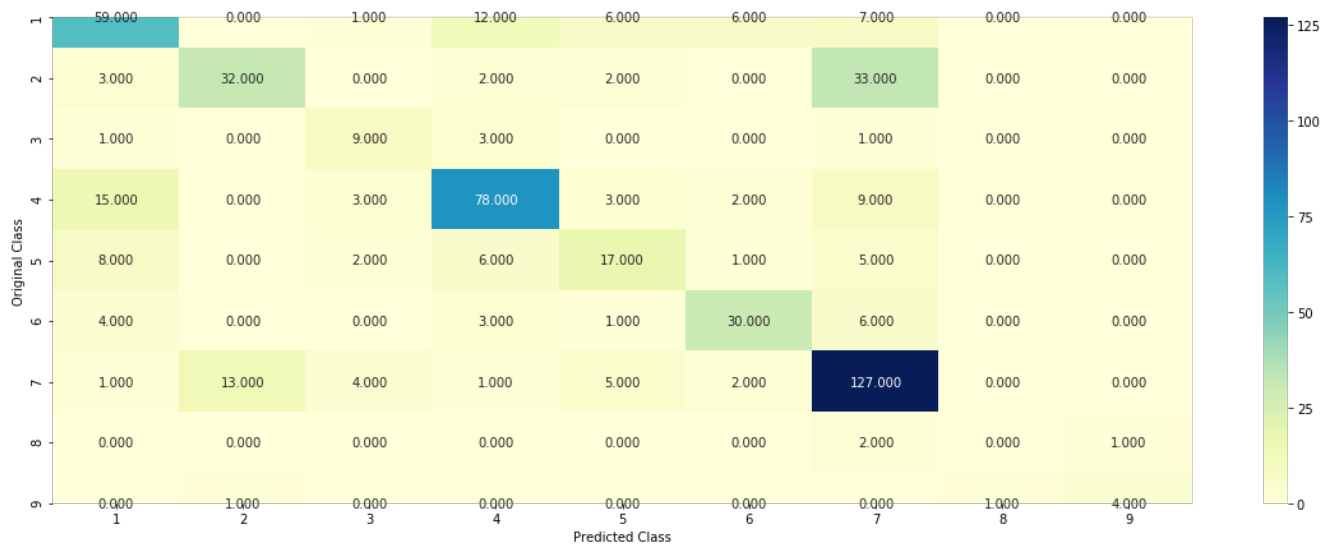
```

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

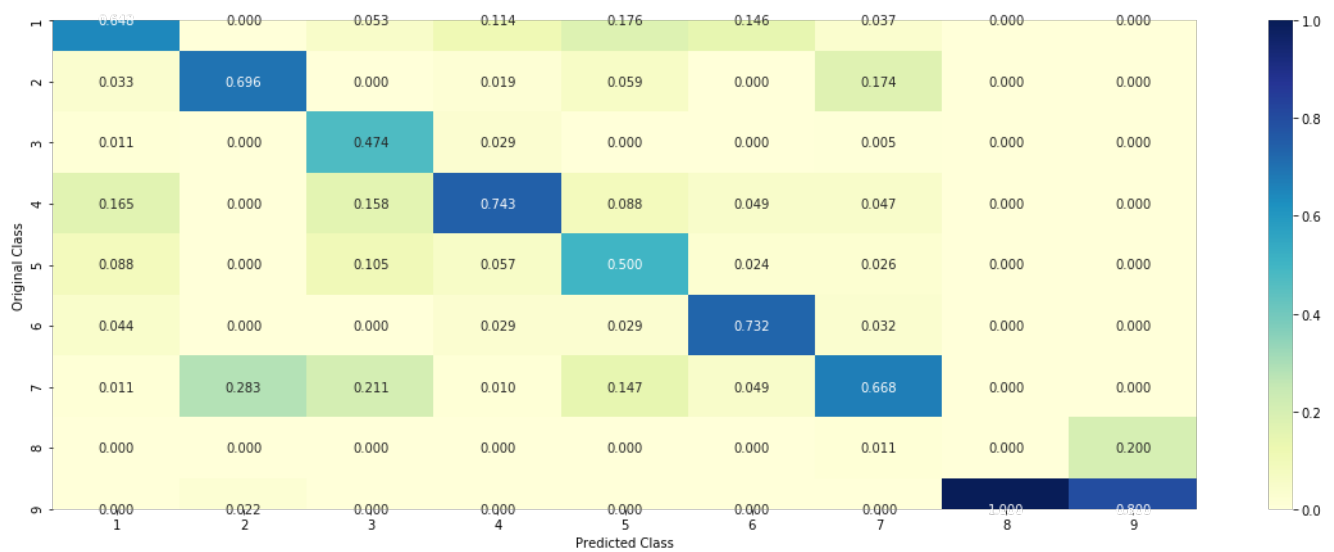
Log loss : 1.0760880535058295

Number of mis-classified points : 0.3308270676691729

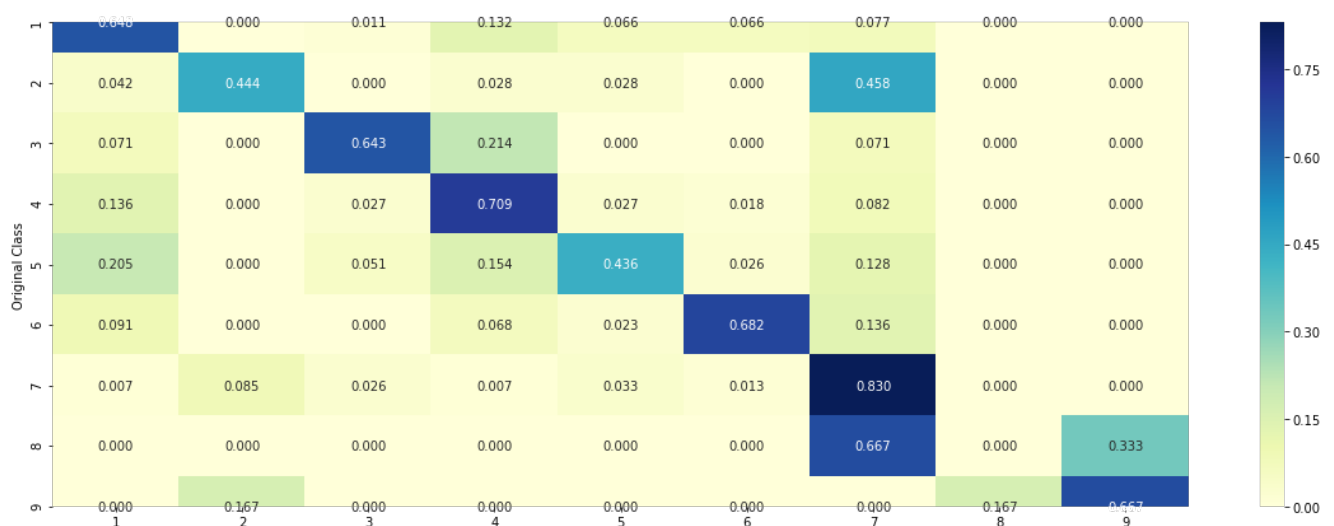
----- Confusion matrix -----



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



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



### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

In [83]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5581 0.078  0.0176 0.1449 0.0395 0.0221 0.1277 0.0052 0.0069]]
Actual Class : 1
-----
285 Text feature [mediates] present in test data point [True]
300 Text feature [silencing] present in test data point [True]
326 Text feature [fl] present in test data point [True]
477 Text feature [archaeobacterial] present in test data point [True]
Out of the top 500 features 4 are present in query point
```

#### 4.3.3.2. For Incorrectly classified point

In [84]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0193 0.9028 0.0102 0.0111 0.0204 0.0045 0.0176 0.0078 0.0062]]
Actual Class : 7
-----
69 Text feature [5v] present in test data point [True]
255 Text feature [posthybridization] present in test data point [True]
256 Text feature [50jc] present in test data point [True]
331 Text feature [md] present in test data point [True]
408 Text feature [chronic] present in test data point [True]
426 Text feature [chromosome] present in test data point [True]
494 Text feature [011] present in test data point [True]
Out of the top 500 features 7 are present in query point
```

## 4.5 Random Forest Classifier

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

In [85]:

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

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

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default 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 : 1.2587131541833148
for n_estimators = 100 and max depth = 10
Log Loss : 1.173486906768551
for n_estimators = 200 and max depth = 5
Log Loss : 1.2494572090806122
for n_estimators = 200 and max depth = 10
Log Loss : 1.1676038193003722
for n_estimators = 500 and max depth = 5
Log Loss : 1.2353535786200665
for n_estimators = 500 and max depth = 10
Log Loss : 1.15672825264805
for n_estimators = 1000 and max depth = 5
Log Loss : 1.2302635570380878
for n_estimators = 1000 and max depth = 10
Log Loss : 1.1588388695627898
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2319177328283872
for n_estimators = 2000 and max depth = 10
Log Loss : 1.157972908849874
For values of best estimator = 500 The train log loss is: 0.6533924017715651
For values of best estimator = 500 The cross validation log loss is: 1.15672825264805
For values of best estimator = 500 The test log loss is: 1.16571188945288

```

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

In [86]:

```

# -----
# 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/
# -----

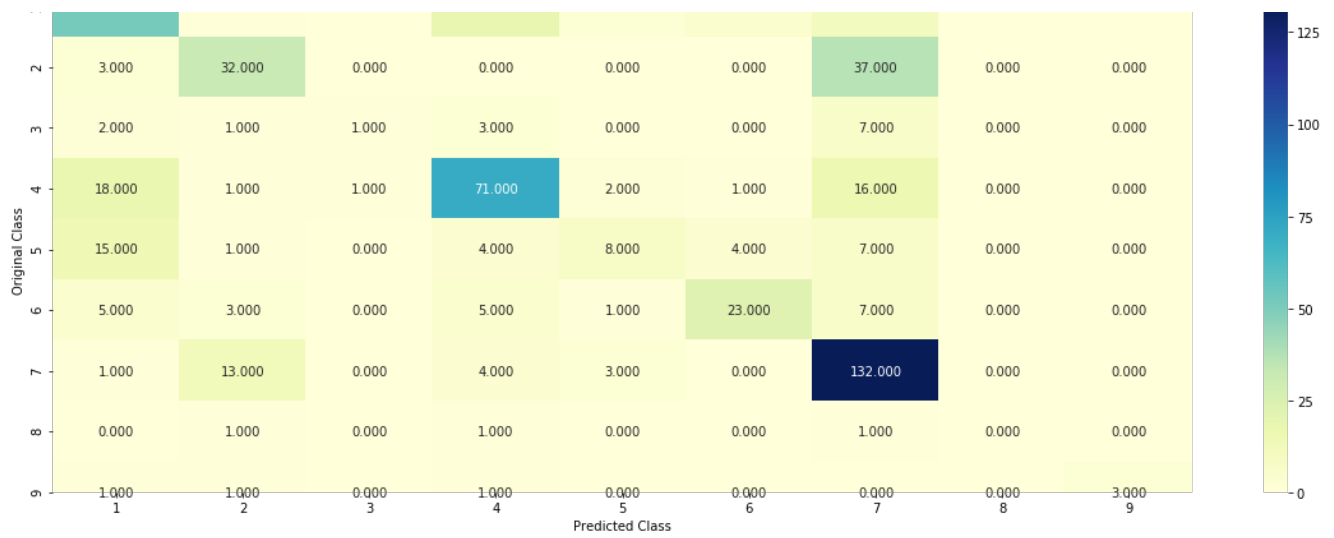
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_
depth[int(best_alpha*2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

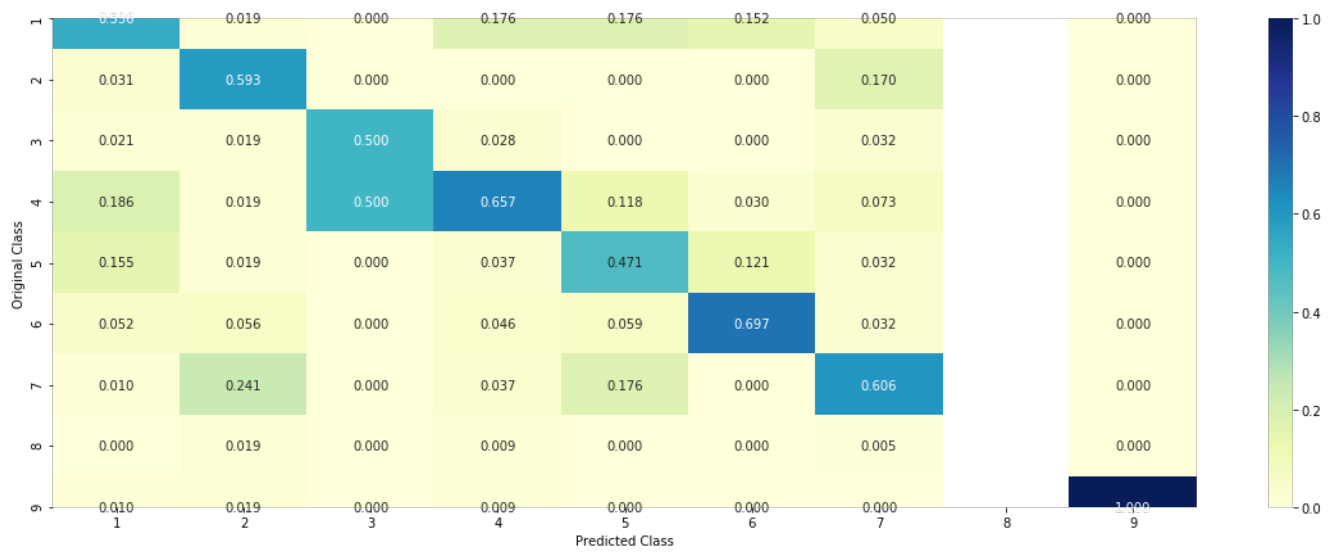
```

Log loss : 1.15672825264805
Number of mis-classified points : 0.39473684210526316
----- Confusion matrix -----

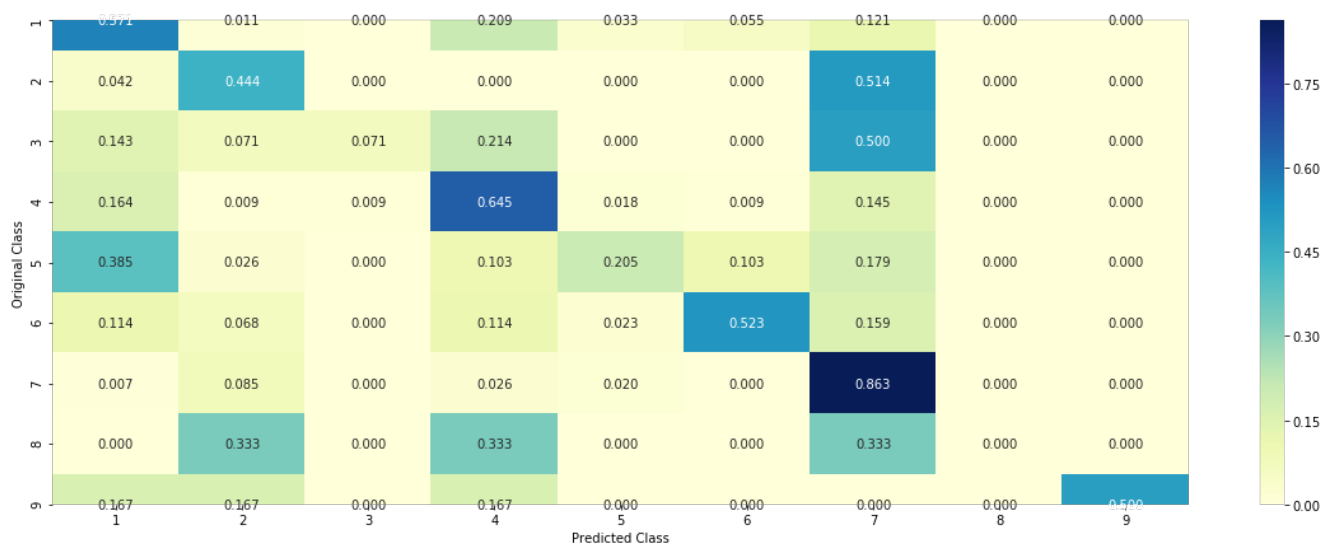
```



Precision matrix (Column Sum=1)



Recall matrix (Row sum=1)



## 4.5.3. Feature Importance

### 4.5.3.1. Correctly Classified point

In [88]:

```
# 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 = 9
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 4

Predicted Class Probabilities: [[0.1611 0.0766 0.023 0.5091 0.0556 0.0339 0.1285 0.0059 0.0064]]

Actual Class : 4

```
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activated] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
5 Text feature [activation] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [growth] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
14 Text feature [signaling] present in test data point [True]
15 Text feature [missense] present in test data point [True]
17 Text feature [therapy] present in test data point [True]
18 Text feature [downstream] present in test data point [True]
22 Text feature [oncogenic] present in test data point [True]
23 Text feature [suppressor] present in test data point [True]
24 Text feature [treated] present in test data point [True]
25 Text feature [yeast] present in test data point [True]
26 Text feature [inhibition] present in test data point [True]
27 Text feature [loss] present in test data point [True]
28 Text feature [activate] present in test data point [True]
31 Text feature [resistance] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
37 Text feature [response] present in test data point [True]
39 Text feature [patients] present in test data point [True]
40 Text feature [serum] present in test data point [True]
43 Text feature [cells] present in test data point [True]
45 Text feature [oncogene] present in test data point [True]
47 Text feature [inhibited] present in test data point [True]
50 Text feature [survival] present in test data point [True]
51 Text feature [protein] present in test data point [True]
52 Text feature [factor] present in test data point [True]
53 Text feature [amplification] present in test data point [True]
54 Text feature [drug] present in test data point [True]
56 Text feature [cell] present in test data point [True]
60 Text feature [expressing] present in test data point [True]
63 Text feature [clinical] present in test data point [True]
69 Text feature [functional] present in test data point [True]
78 Text feature [phosphorylated] present in test data point [True]
83 Text feature [activity] present in test data point [True]
85 Text feature [stability] present in test data point [True]
87 Text feature [terminal] present in test data point [True]
88 Text feature [functions] present in test data point [True]
90 Text feature [predicted] present in test data point [True]
91 Text feature [active] present in test data point [True]
94 Text feature [potential] present in test data point [True]
99 Text feature [resistant] present in test data point [True]
Out of the top 100 features 46 are present in query point
```

#### 4.5.3.2. Incorrectly Classified point

In [89]:

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

Predicted Class : 2

Predicted Class Probabilities: [[0.019 0.6554 0.0153 0.0155 0.0334 0.0233 0.23 0.0046 0.0034]]

Actual Class : 7

```
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activated] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
5 Text feature [activation] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [growth] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
11 Text feature [inhibitor] present in test data point [True]
12 Text feature [receptor] present in test data point [True]
13 Text feature [trials] present in test data point [True]
16 Text feature [constitutive] present in test data point [True]
17 Text feature [therapy] present in test data point [True]
18 Text feature [downstream] present in test data point [True]
22 Text feature [oncogenic] present in test data point [True]
24 Text feature [treated] present in test data point [True]
26 Text feature [inhibition] present in test data point [True]
27 Text feature [loss] present in test data point [True]
30 Text feature [ligand] present in test data point [True]
31 Text feature [resistance] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
33 Text feature [kinases] present in test data point [True]
36 Text feature [constitutively] present in test data point [True]
37 Text feature [response] present in test data point [True]
38 Text feature [proliferation] present in test data point [True]
39 Text feature [patients] present in test data point [True]
40 Text feature [serum] present in test data point [True]
42 Text feature [months] present in test data point [True]
43 Text feature [cells] present in test data point [True]
45 Text feature [oncogene] present in test data point [True]
47 Text feature [inhibited] present in test data point [True]
50 Text feature [survival] present in test data point [True]
51 Text feature [protein] present in test data point [True]
52 Text feature [factor] present in test data point [True]
53 Text feature [amplification] present in test data point [True]
54 Text feature [drug] present in test data point [True]
55 Text feature [ic50] present in test data point [True]
56 Text feature [cell] present in test data point [True]
59 Text feature [phospho] present in test data point [True]
60 Text feature [expressing] present in test data point [True]
62 Text feature [pathogenic] present in test data point [True]
63 Text feature [clinical] present in test data point [True]
68 Text feature [imatinib] present in test data point [True]
73 Text feature [dose] present in test data point [True]
76 Text feature [efficacy] present in test data point [True]
77 Text feature [trial] present in test data point [True]
78 Text feature [phosphorylated] present in test data point [True]
79 Text feature [sensitive] present in test data point [True]
82 Text feature [sensitivity] present in test data point [True]
83 Text feature [activity] present in test data point [True]
84 Text feature [chronic] present in test data point [True]
87 Text feature [terminal] present in test data point [True]
89 Text feature [autophosphorylation] present in test data point [True]
90 Text feature [predicted] present in test data point [True]
91 Text feature [active] present in test data point [True]
92 Text feature [phosphatase] present in test data point [True]
```

```

93 Text feature [phosphotyrosine] present in test data point [True]
94 Text feature [potential] present in test data point [True]
97 Text feature [abl] present in test data point [True]
98 Text feature [advanced] present in test data point [True]
99 Text feature [resistant] present in test data point [True]
Out of the top 100 features 63 are present in query point

```

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

In [90]:

```

# -----
# 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/
# -----

# 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 value")

```

```

plt.xlabel("Alpha 1/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))

```

```

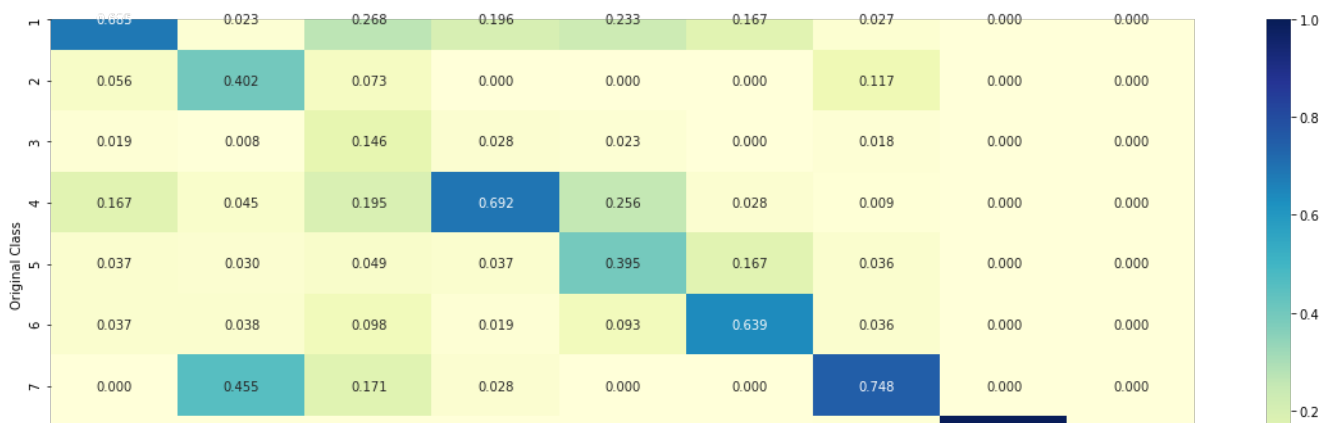
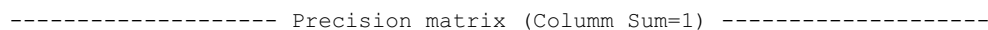
for n_estimators = 10 and max depth = 2
Log Loss : 2.1981960693115625
for n_estimators = 10 and max depth = 3
Log Loss : 1.7126433295939254
for n_estimators = 10 and max depth = 5
Log Loss : 1.442216859968172
for n_estimators = 10 and max depth = 10
Log Loss : 1.7560484632916606
for n_estimators = 50 and max depth = 2
Log Loss : 1.6998878650537383
for n_estimators = 50 and max depth = 3
Log Loss : 1.5029015803404657
for n_estimators = 50 and max depth = 5
Log Loss : 1.288073990122659
for n_estimators = 50 and max depth = 10
Log Loss : 1.6618200328045722
for n_estimators = 100 and max depth = 2
Log Loss : 1.6430354254810784
for n_estimators = 100 and max depth = 3
Log Loss : 1.5569433353161275
for n_estimators = 100 and max depth = 5
Log Loss : 1.3373561493436124
for n_estimators = 100 and max depth = 10
Log Loss : 1.7902647644792653
for n_estimators = 200 and max depth = 2
Log Loss : 1.6460768789032745
for n_estimators = 200 and max depth = 3
Log Loss : 1.520884557523119
for n_estimators = 200 and max depth = 5
Log Loss : 1.3631379421142817
for n_estimators = 200 and max depth = 10
Log Loss : 1.7083999260980136
for n_estimators = 500 and max depth = 2
Log Loss : 1.7516254688012196
for n_estimators = 500 and max depth = 3
Log Loss : 1.6016149986496127
for n_estimators = 500 and max depth = 5
Log Loss : 1.4262842103381799
for n_estimators = 500 and max depth = 10
Log Loss : 1.7648610444148642
for n_estimators = 1000 and max depth = 2
Log Loss : 1.7183705177984065
for n_estimators = 1000 and max depth = 3
Log Loss : 1.593912478154091
for n_estimators = 1000 and max depth = 5
Log Loss : 1.4383766844186008
for n_estimators = 1000 and max depth = 10
Log Loss : 1.7492944999986313
For values of best alpha = 50 The train log loss is: 0.055716510433751615
For values of best alpha = 50 The cross validation log loss is: 1.288073990122659
For values of best alpha = 50 The test log loss is: 1.3002581248226222

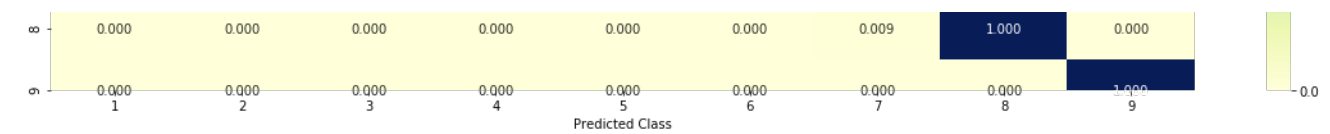
```

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

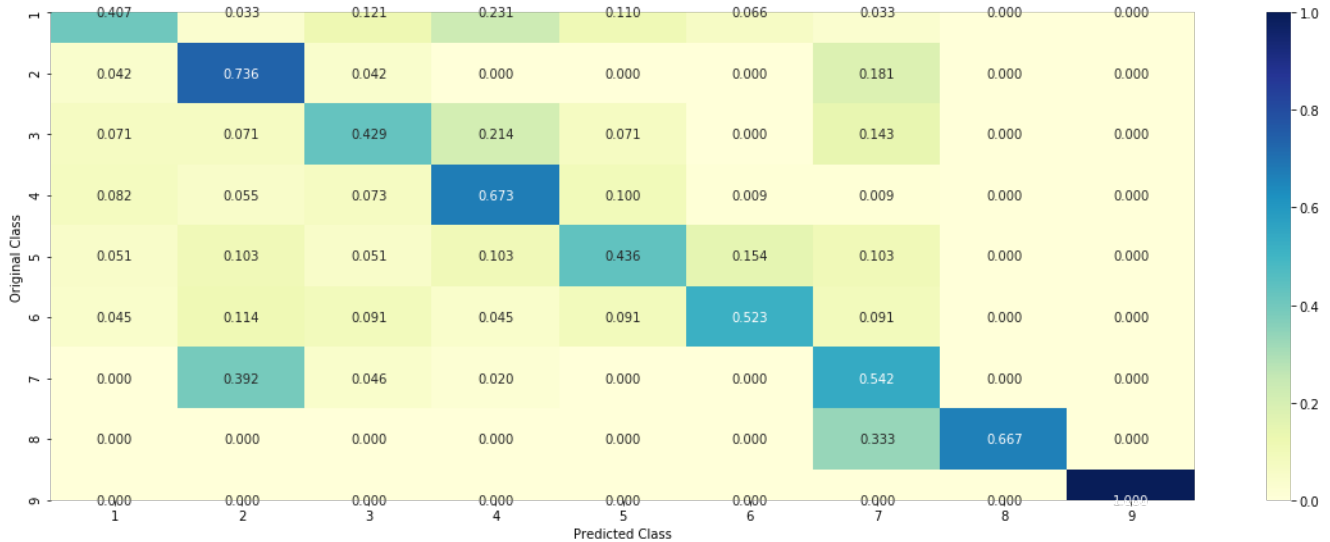
## In [91]:

```
Log loss : 1.288073990122659
Number of mis-classified points : 0.4342105263157895
----- Confusion matrix -----
```





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



## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point

In [93]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.2573 0.0814 0.1509 0.1611 0.0385 0.0902 0.0181 0.1294 0.0732]]
Actual Class : 1
```

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



```

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

```

#### 4.5.5.2. Incorrectly Classified point

In [94]:

```

test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")

```

```

Predicted Class : 2
Predicted Class Probabilities: [[0.0089 0.6509 0.083  0.0109 0.0112 0.0192 0.1949 0.015  0.0059]]
Actual Class : 7

```

```

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

```

## 4.7 STACK THE MODELS

### 4.7.1 testing with hyper parameter tuning

In [95]:

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

```

```

Logistic Regression : Log Loss: 1.04
Support vector machines : Log Loss: 1.47
Naive Bayes : Log Loss: 1.26

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.489
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.100
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.183
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.428

```

## 4.7.2 testing the model with the best hyper parameters

In [96]:

```

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

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

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

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

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

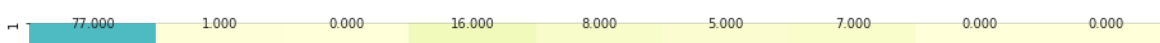
```

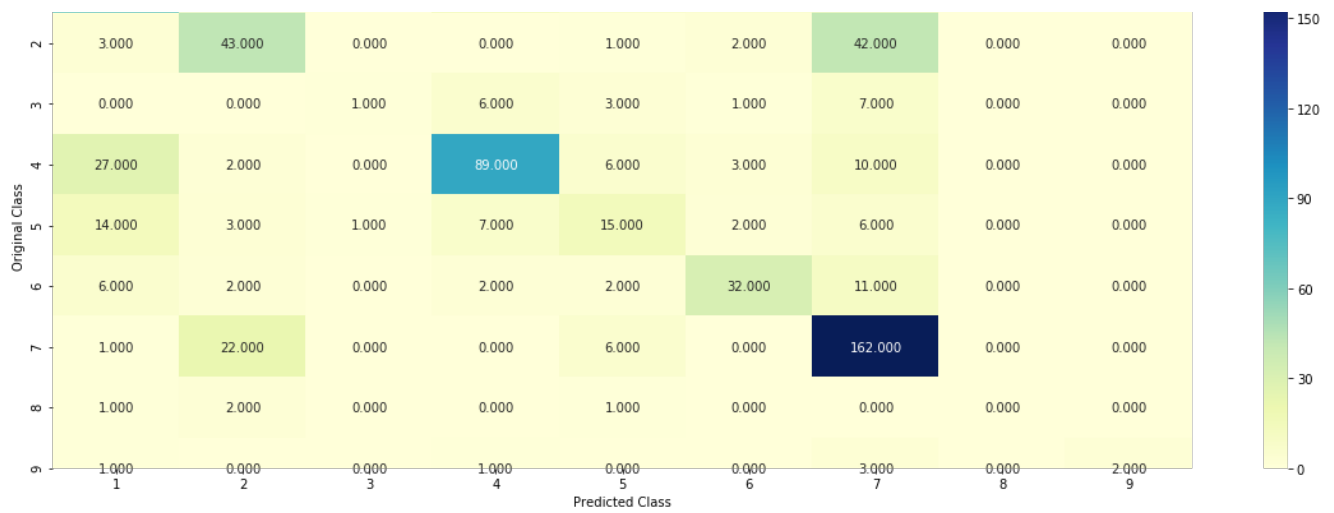
```

Log loss (train) on the stacking classifier : 0.6348331404588331
Log loss (CV) on the stacking classifier : 1.1000128192103225
Log loss (test) on the stacking classifier : 1.1406262531387636
Number of missclassified point : 0.3669172932330827

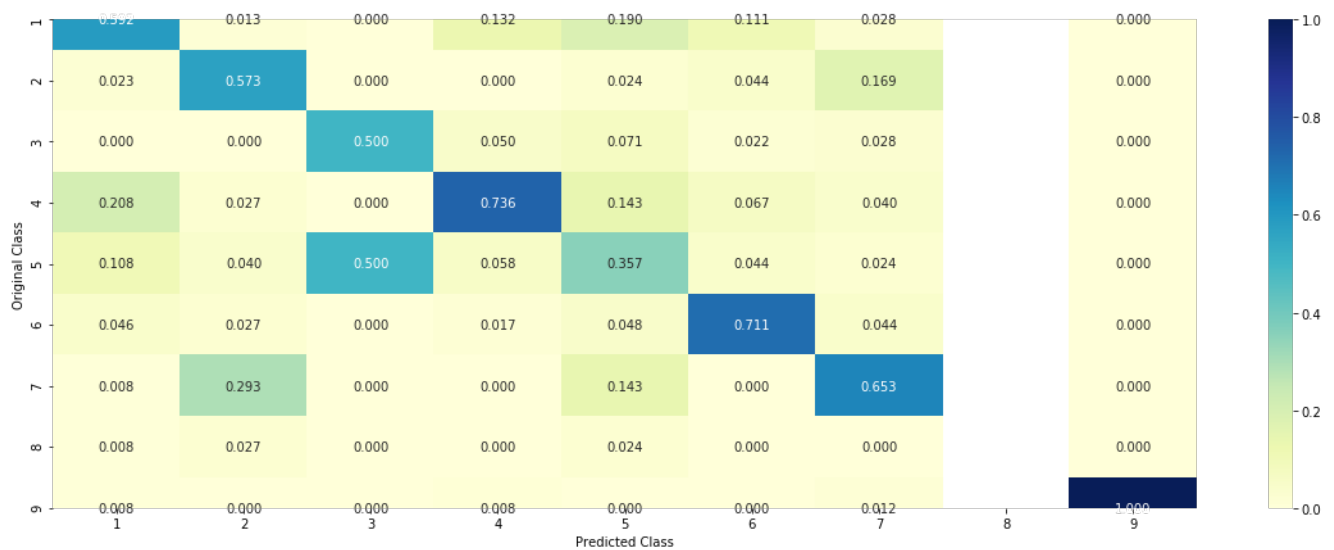
```

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

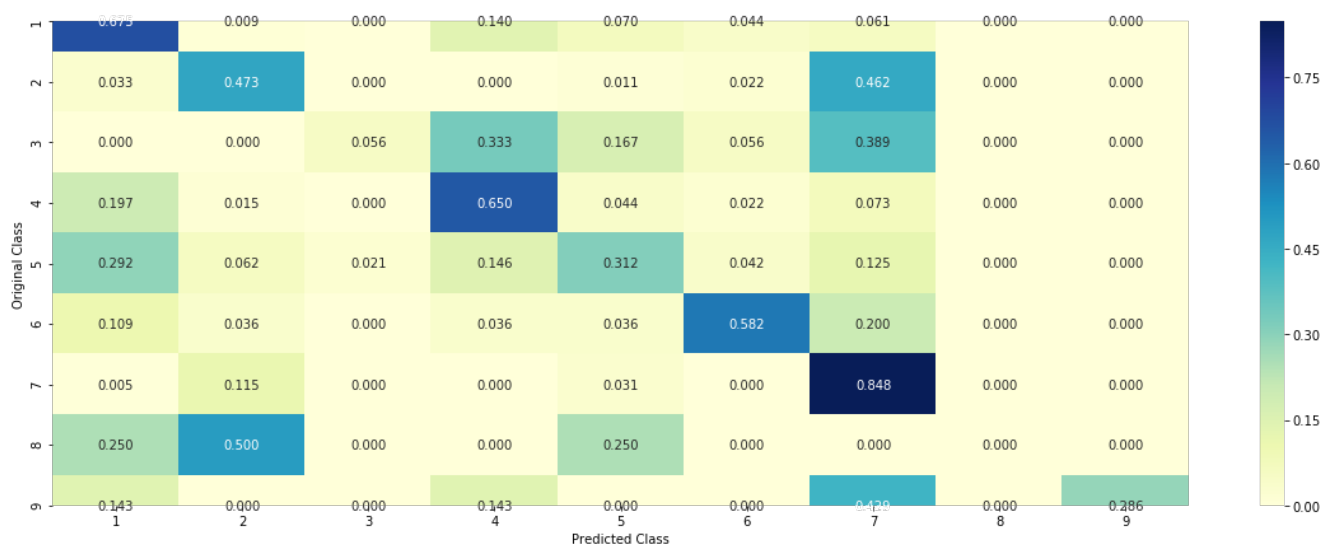




----- Precision matrix (Column Sum=1) -----



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



### 4.7.3 Maximum Voting classifier

In [97]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('xgb', xgb_clf1), ('lgb', lgb_clf2), ('rf', rf_clf3)], voting=
```

```

vclf = VotingClassifier(estimators=[('11', sig_clf1), ('SVC', sig_clf2), ('11', 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))

```

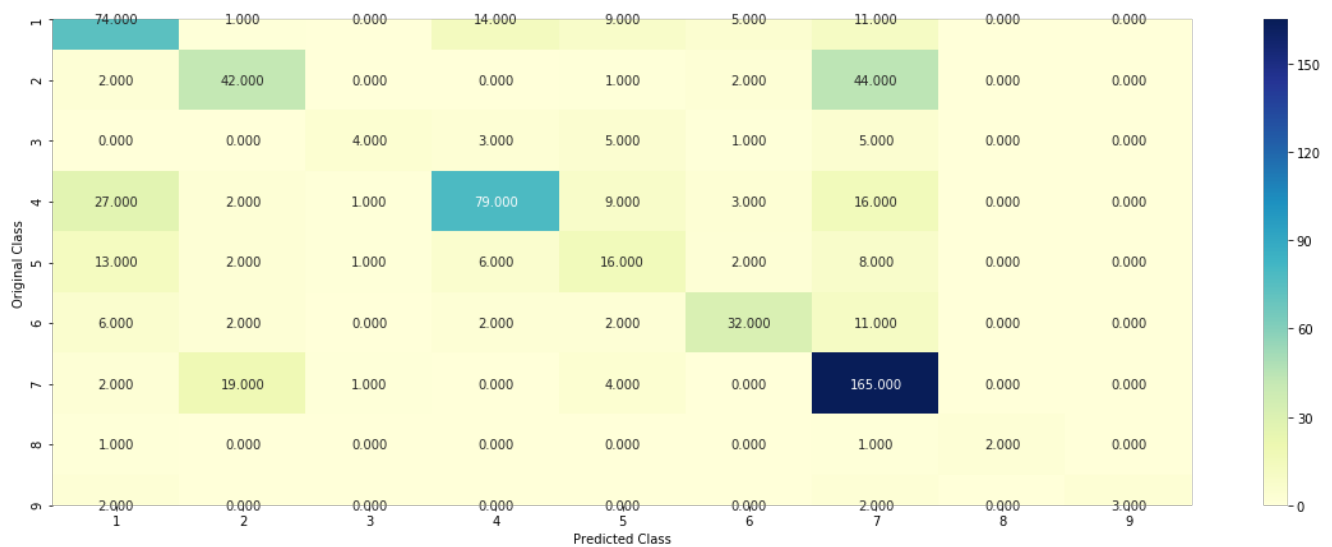
Log loss (train) on the VotingClassifier : 0.8505052966994576

Log loss (CV) on the VotingClassifier : 1.1169584934058965

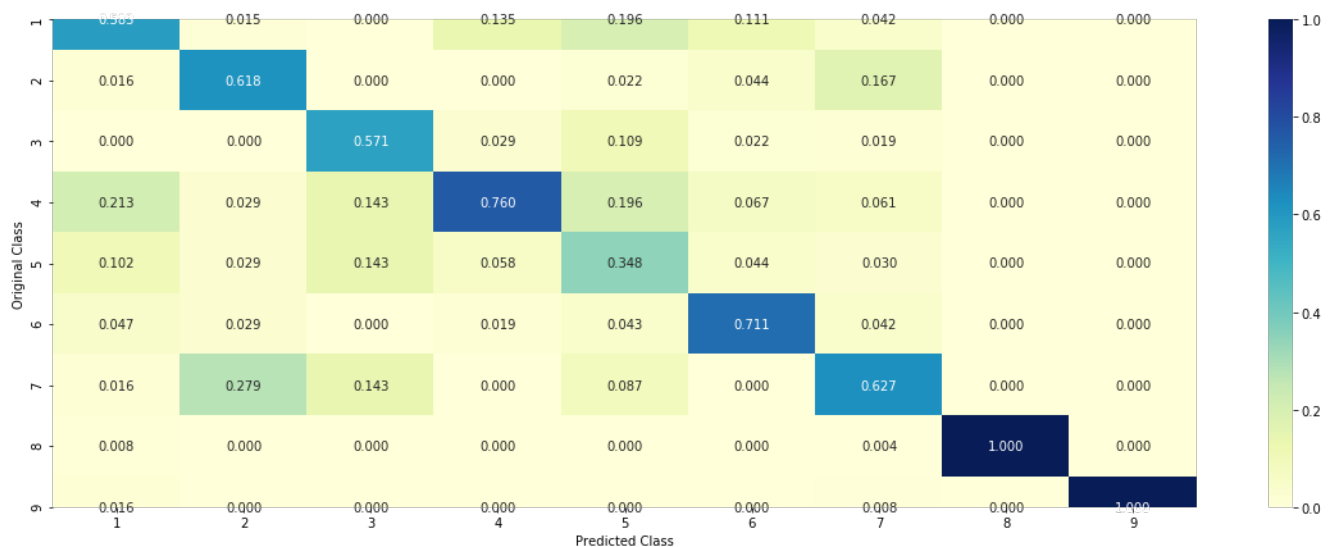
Log loss (test) on the VotingClassifier : 1.1651527978963652

Number of missclassified point : 0.37293233082706767

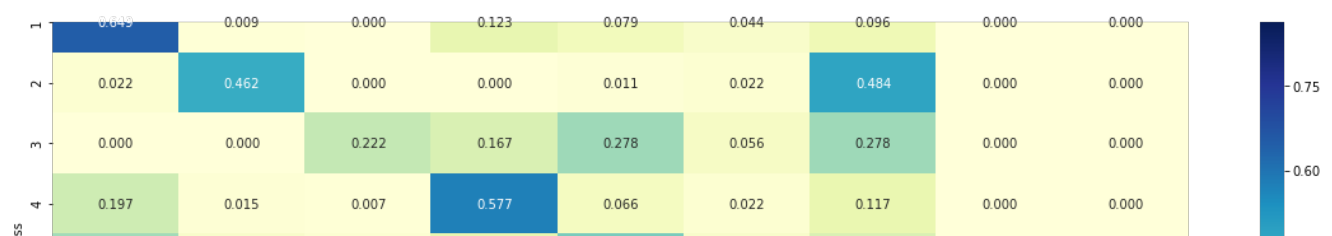
----- Confusion matrix -----

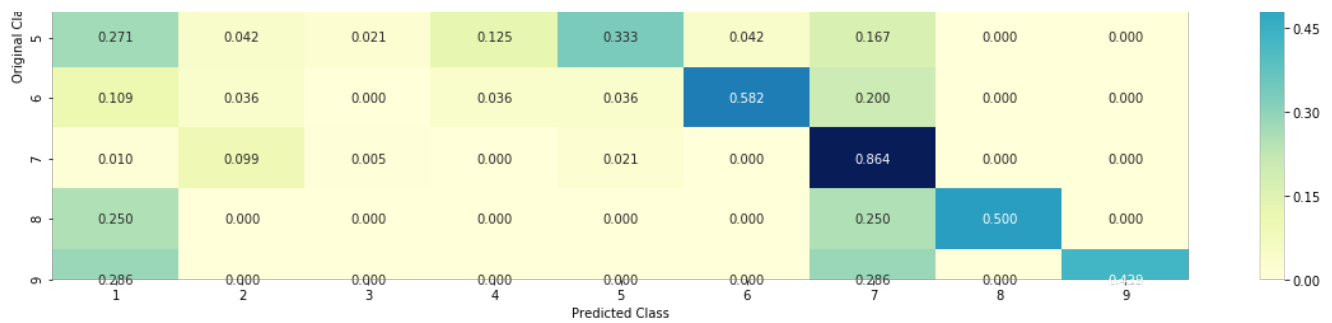


----- Precision matrix (Column Sum=1) -----



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





In [75]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.field_names=["Model Name","Best alpha","Train","CV","Test","% Misclassified Points"]
ptable.add_row(["Naive Bayes","1000","0.91","1.20","1.20","40"])
ptable.add_row(["KNN","0.001","0.71","1.04","1.05","37"])
ptable.add_row(["Logistic Regression With Class balancing","0.001","0.42","1.03","0.99","33"])
ptable.add_row(["Logistic Regression Without Class balancing","0.001","0.53","1.03","1.05","34"])
ptable.add_row(["Linear SVM","0.001","0.58","1.07","1.11","33"])
ptable.add_row(["Random Forest Classifier With One hot Encoding","500","0.65","1.15","1.16","39"])
ptable.add_row(["Random Forest Classifier With Response Coding","50","0.55","1.28","1.30","43"])
ptable.add_row(["Stack Models:LR+NB+SVM","0.1","0.63","1.10","1.14","36"])
ptable.add_row(["Maximum Voting classifier","0.1","0.85","1.11","1.16","37"])
print(ptable)
```

Model Name	Best alpha	Train	CV	Test	% Misclassified Points
Naive Bayes	1000	0.91	1.20	1.20	40
KNN	0.001	0.71	1.04	1.05	37
Logistic Regression With Class balancing	0.001	0.42	1.03	0.99	33
Logistic Regression Without Class balancing	0.001	0.53	1.03	1.05	34
Linear SVM	0.001	0.58	1.07	1.11	33
Random Forest Classifier With One hot Encoding	500	0.65	1.15	1.16	39
Random Forest Classifier With Response Coding	50	0.55	1.28	1.30	43
Stack Models:LR+NB+SVM	0.1	0.63	1.10	1.14	36
Maximum Voting classifier	0.1	0.85	1.11	1.16	37

Logistic Regression with Class Balancing is the best model because Misclassified points percentage is lowest in this model

## 5. Assignments

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

### Logistic Regression including both Unigrams & Bigrams

#### Gene Feature

In [114]:

```
#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 [115]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
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'])

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

## Variation Feature

In [116]:

```
# 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 [117]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
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'])

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

## Text Feature

In [118]:

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

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

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it 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 : 780839

In [119]:

```
#response coding of text features
train_text_feature_responseCoding = get_text_responsecoding(train_df)
test_text_feature_responseCoding = get_text_responsecoding(test_df)
cv_text_feature_responseCoding = get_text_responsecoding(cv_df)

# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).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 [120]:

```
# 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)
```

## Stack all the features

In [122]:

```
# merging gene, variance and text features

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

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

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
```



```

cv_x_onehotCoding = np.hstack((cv_gene_var_onehotCoding, cv_gene_feature_onehotCoding, / / 783134),
cv_y = np.array(list(cv_df['Class']))

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

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

```

In [123]:

```

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, 783134)
(number of data points * number of features) in test data = (665, 783134)
(number of data points * number of features) in cross validation data = (532, 783134)

```

In [124]:

```

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

```

```

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

```

## Apply Model

In [125]:

```

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 probabillites 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))

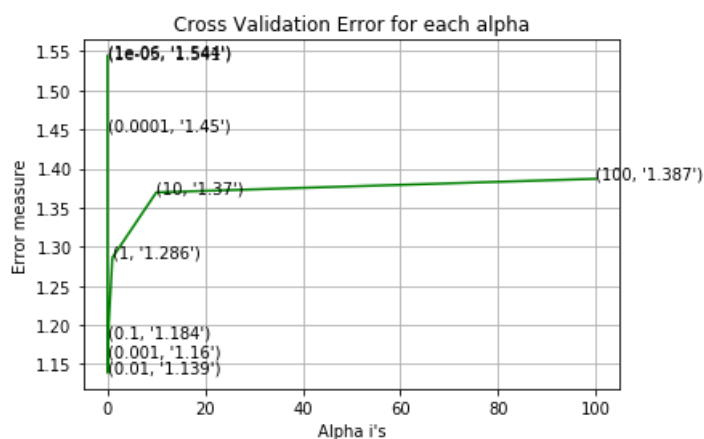
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha], "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```

for alpha = 1e-06
Log Loss : 1.5443621452118714
for alpha = 1e-05
Log Loss : 1.5412859256742546
for alpha = 0.0001
Log Loss : 1.450077987112674
for alpha = 0.001
Log Loss : 1.1602062697160977
for alpha = 0.01
Log Loss : 1.138642028245727
for alpha = 0.1
Log Loss : 1.1844779075765295
for alpha = 1
Log Loss : 1.285994705066521
for alpha = 10
Log Loss : 1.3695328094481085
for alpha = 100
Log Loss : 1.3868985364501103

```



```

For values of best alpha = 0.01 The train log loss is: 0.7391782966199689
For values of best alpha = 0.01 The cross validation log loss is: 1.138642028245727
For values of best alpha = 0.01 The test log loss is: 1.1799385844267543

```

In [126]:

```

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

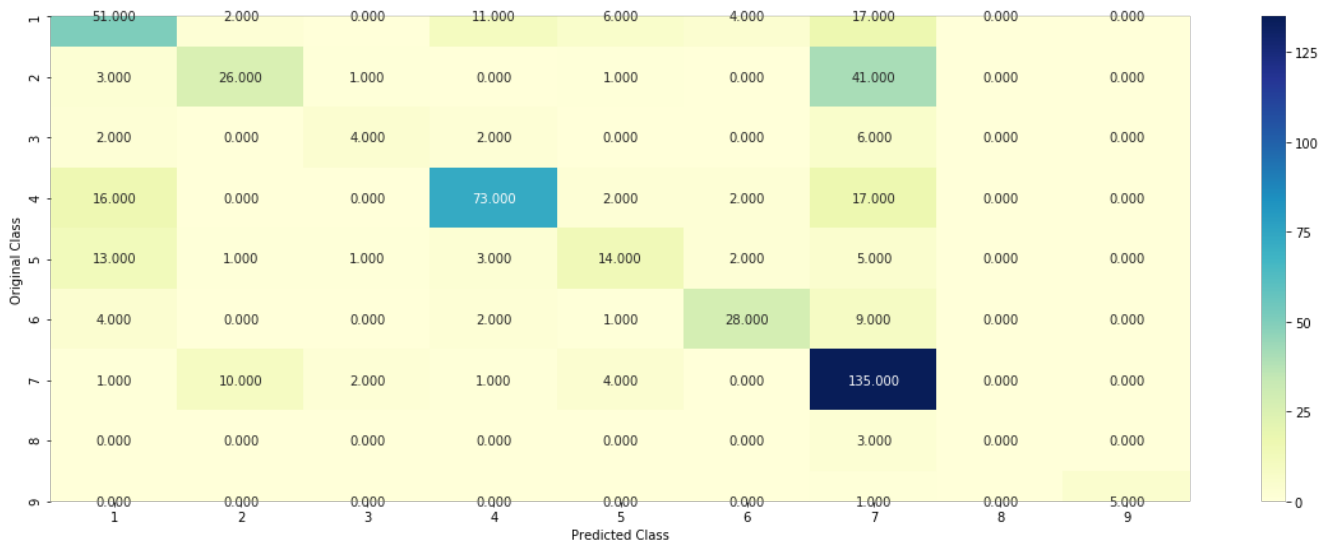
```

```

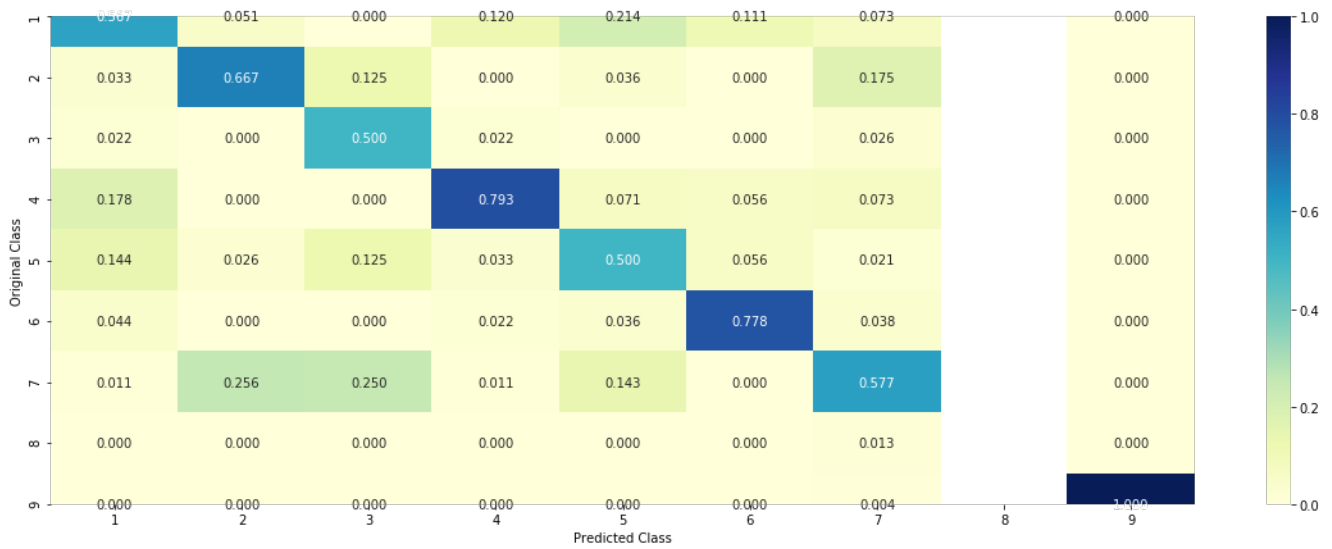
Log loss : 1.138642028245727
Number of mis-classified points : 0.3684210526315789

```

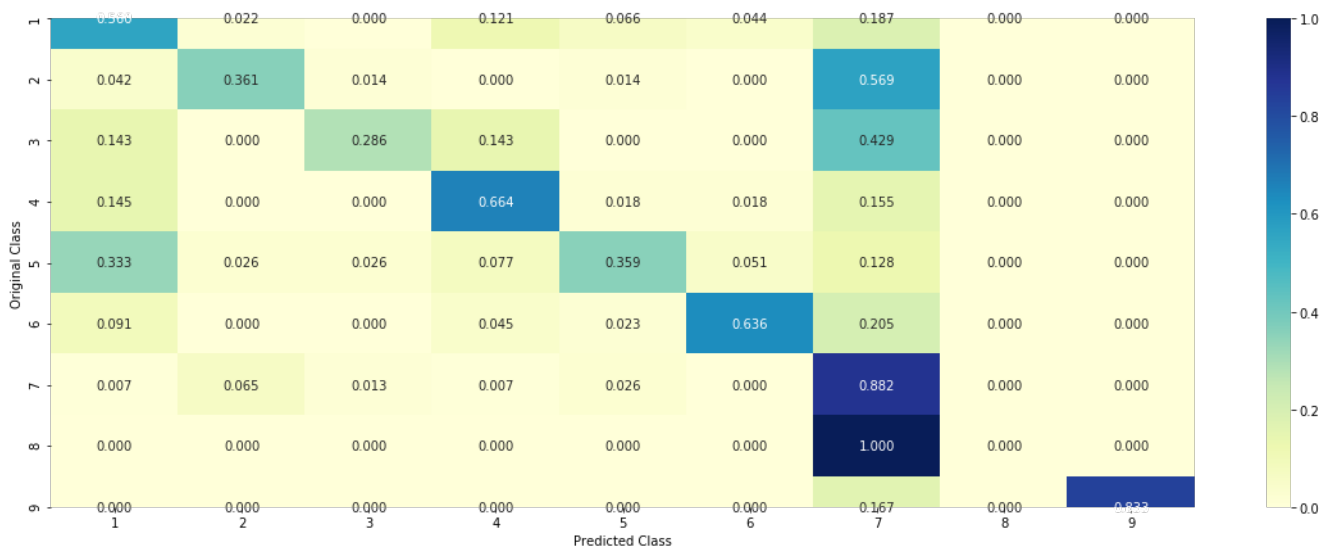
----- Confusion matrix -----



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



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



Value of Log-Loss is still > 1 even after applying both unigrams and bi-grams.

## Feature Engineering

### Merging Gene and Variation Features

In [57]:

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

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

In [58]:

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    gv_fea = []
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    return gv_fea
```

In [59]:

```
#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", x_train))

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

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

In [60]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

### Variation Feature

In [61]:

```
# alpha is used for laplace smoothing
```

```
alpha = 1

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

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

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

In [62]:

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

## Text Feature

In [63]:

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

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

In [64]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,5), max_features=4000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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

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

Total number of unique words in train data : 4000

In [65]:

```
dict_list = []
# dict_list=[] contains 9 dictionaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['Class']==i]
```

```

# build a word dict based on the words in that class
dict_list.append(extract_dictionary_paddle(cls_text))
# append it to dict_list

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

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

```

In [66]:

```

#response coding of text features
train_text_feature_responseCoding = get_text_responsecoding(x_train)
test_text_feature_responseCoding = get_text_responsecoding(x_test)
cv_text_feature_responseCoding = get_text_responsecoding(x_cv)

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

```

In [67]:

```

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

```

## New Features

In [68]:

```

# Collecting all the genes and variations data into a single list
gene_variation = []

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

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

```

In [69]:

```

tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

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

```

## Stack features

In [70]:

```

train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding =

```

```

hstack((test_gene_feature_onehotCoding, test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding, cv_variation_feature_onehotCoding)
)

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

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

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

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

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

```

In [71]:

```

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, 7203)
(number of data points * number of features) in test data = (665, 7203)
(number of data points * number of features) in cross validation data = (532, 7203)

```

In [72]:

```

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

```

Response encoding features :

```

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

```

In [73]:

```

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))

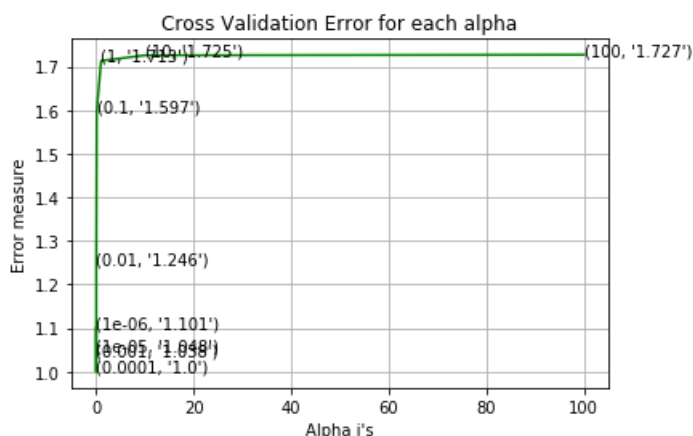
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha], "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```

for alpha = 1e-06
Log Loss : 1.1007467649355196
for alpha = 1e-05
Log Loss : 1.047893615832294
for alpha = 0.0001
Log Loss : 0.9995076977111816
for alpha = 0.001
Log Loss : 1.0378280719433108
for alpha = 0.01
Log Loss : 1.2459790007573117
for alpha = 0.1
Log Loss : 1.5970258391253154
for alpha = 1
Log Loss : 1.7128356329414136
for alpha = 10
Log Loss : 1.7253992969571585
for alpha = 100
Log Loss : 1.7267214782614633

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4548250892599387
For values of best alpha = 0.0001 The cross validation log loss is: 0.9995076977111816

```



For values of best alpha = 0.0001 The test log loss is: 0.9734189596613154

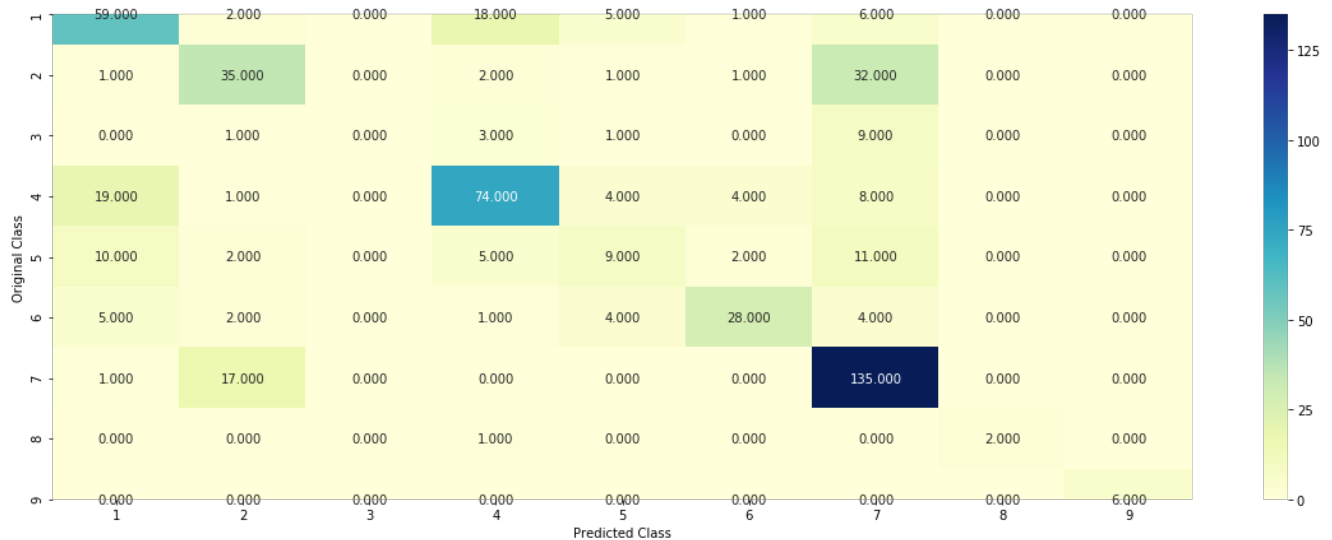
In [74]:

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

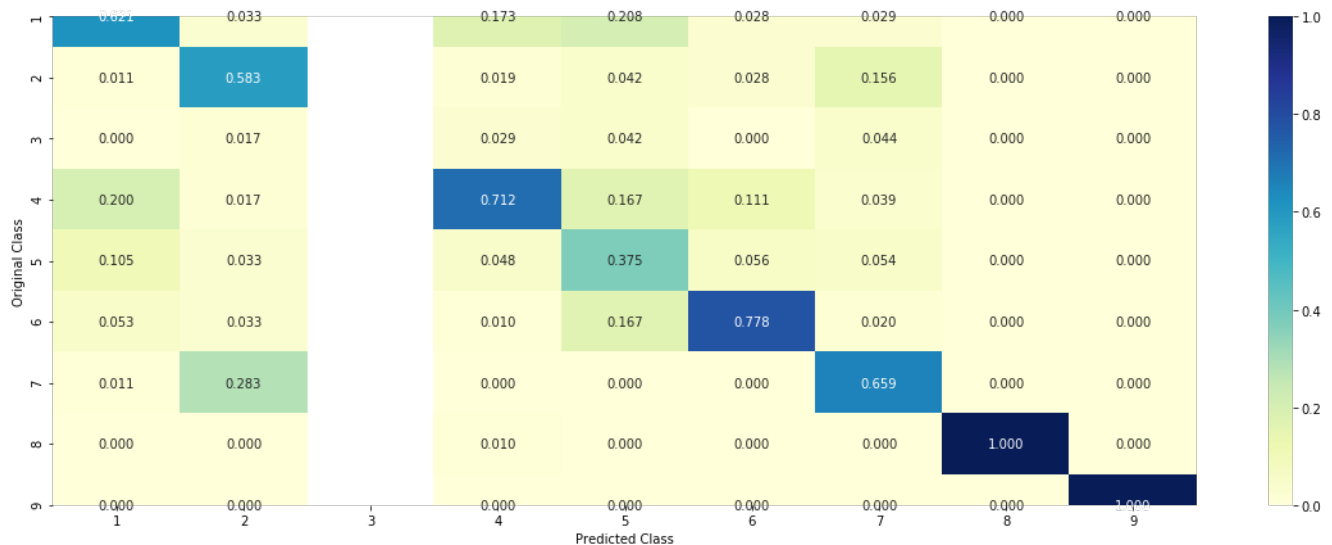
Log loss : 0.9995076977111816

Number of mis-classified points : 0.3458646616541353

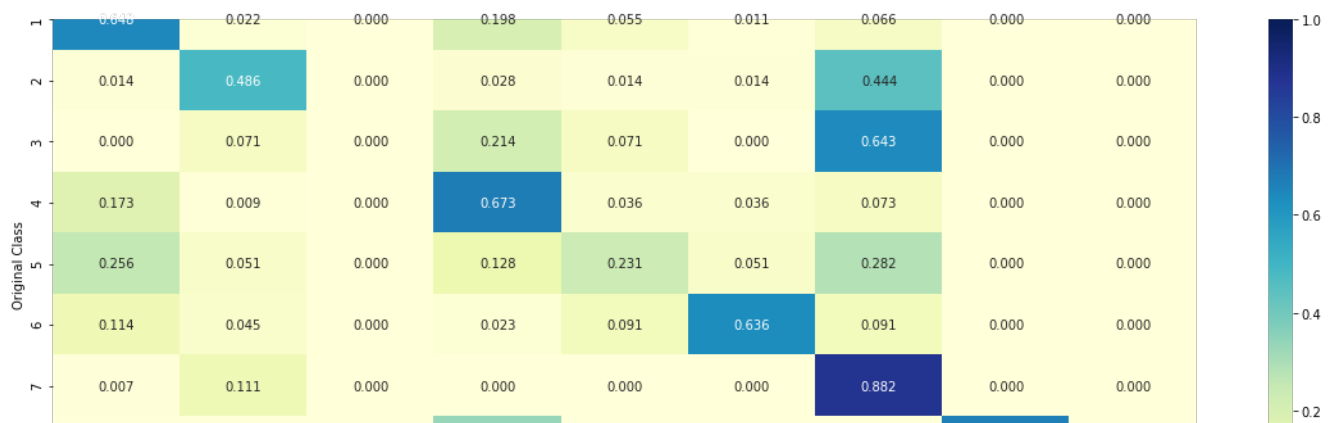
```
----- Confusion matrix -----
```

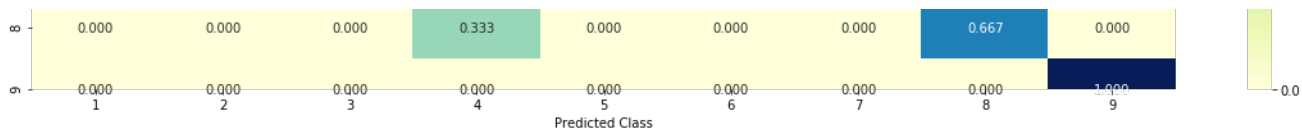


```
----- Precision matrix (Columm Sum=1) -----
```



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





In [76]:

```
from prettytable import PrettyTable
```

```
phtable = PrettyTable()
```

```
phtable.field_names=["Model Name","Best alpha","Train","CV","Test","% Misclassified Points"]
```

```
phtable.add_row(["Linear Regression with Feature Engineering","0.0001","0.45","0.99","0.97","34"])
```

```
print(phtable)
```

Model Name	Best alpha	Train	CV	Test	% Misclassified Points
Linear Regression with Feature Engineering	0.0001	0.45	0.99	0.97	34

## Conclusions

1. By using Feature Engineering, we reduce the value of log-loss to be < 1.
2. For feature Engineering technique, we combined gene feature with variation feature.
3. Value of Log-Loss = 0.95
4. Cross Validation Log-Loss = 0.99 and Test Log-loss = 0.97

In [ ]: