Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [5]:

```
import nltk
nltk.download('stopwords')
# from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\PRIYANKA\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[5]:

```
True
```

In [6]:

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [7]:

```
data = pd.read_csv('training/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[7]:

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

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

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

3.1.2. Reading Text Data

In [8]:

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

```
Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']
```

Out[8]:

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

3.1.3. Preprocessing of text

In [9]:

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

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

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

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

In [10]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 47.237999066004384 seconds
```

In [11]:

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

Out[11]:

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

In [12]:

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

Out[12]:

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

In [13]:

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

In [14]:

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

Out[14]:

	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 [15]:

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

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

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

In [16]:

```
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 [17]:

```
import pandas as pd
pd.__version__
```

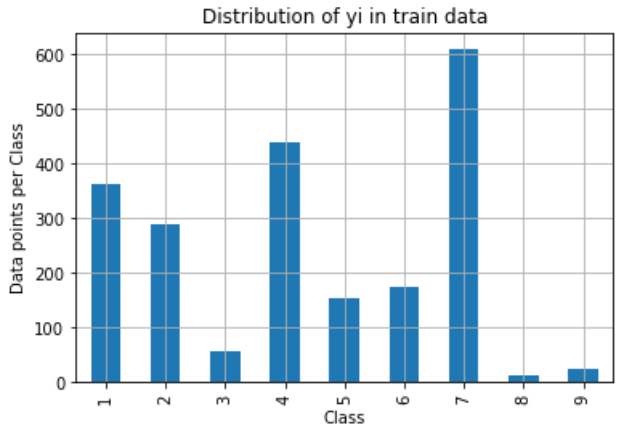
Out[17]:

```
'0.25.3'
```

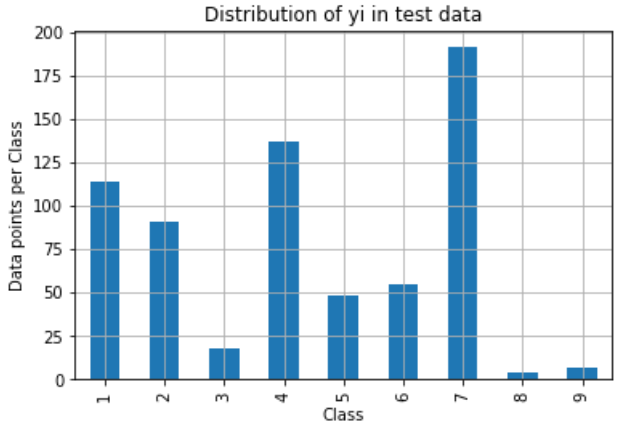
In [18]:

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

my_colors = 'rgbkymc'
```

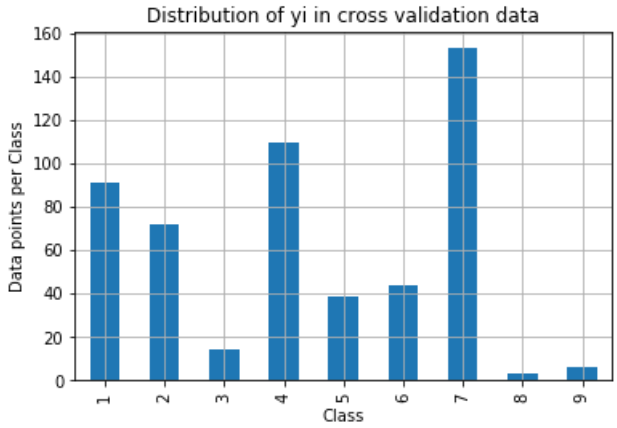
```
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

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


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

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

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

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




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




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




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

3.2 Prediction using a 'Random' Model

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

In [19]:

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

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

    # C = [[1, 2],
    #      [3, 4]]
    # C.T = [[1, 3],
```

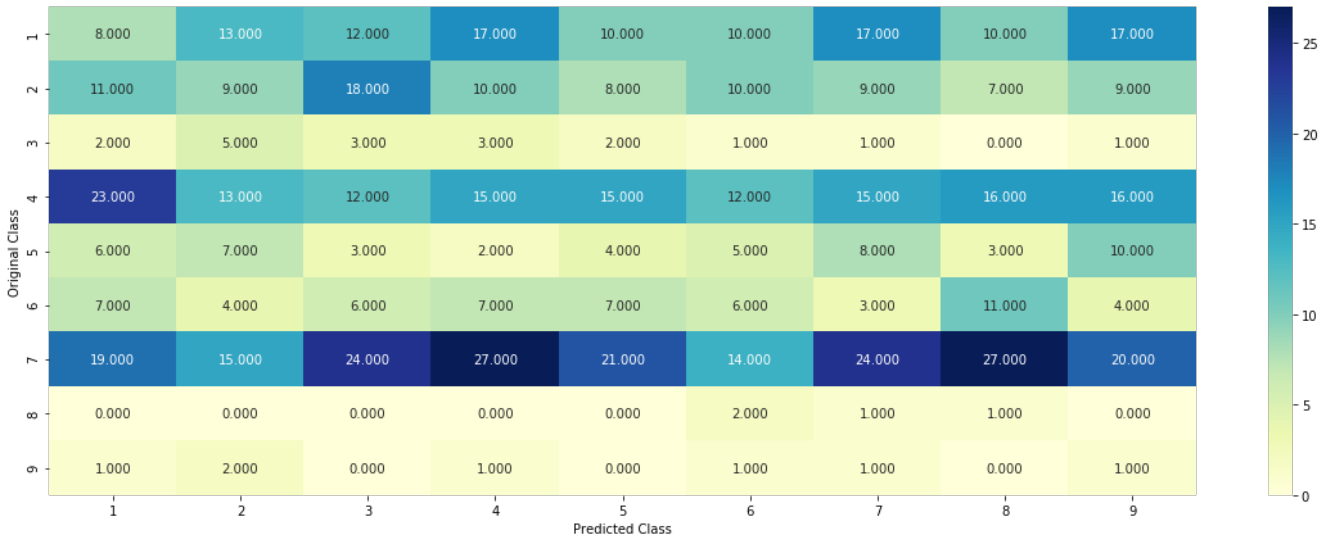
```
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

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

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

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

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

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

In [20]:

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

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-
15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

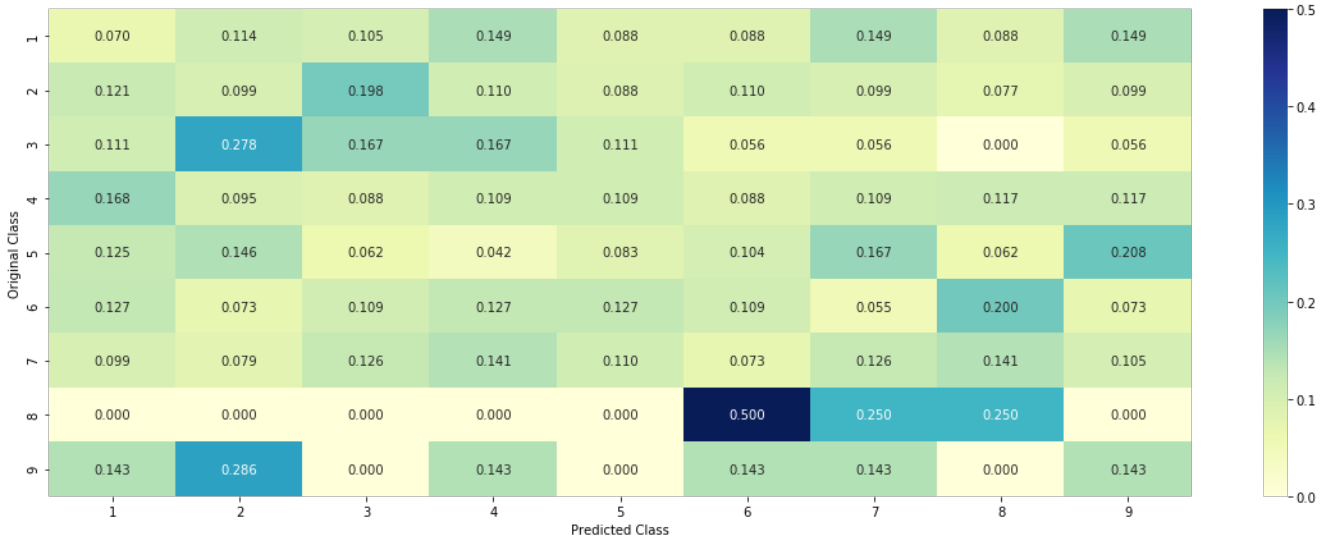
```
Log loss on Cross Validation Data using Random Model 2.4066948219500692
Log loss on Test Data using Random Model 2.4191948169147794
```

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




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


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



3.3 Univariate Analysis

In [21]:

In [21]:

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

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

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

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
```

```
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.0378787878787878, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787878787878, 0.0378787878787878,
0.0378787878787878],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.2159090909090911, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [22]:

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

```
Number of Unique Genes : 229
BRCA1     171
TP53      103
EGFR       92
PTEN       84
BRCA2      76
KIT        69
BRAF       50
```

```
ERBB2       45
PIK3CA      41
PDGFRA      40
Name: Gene, dtype: int64
```

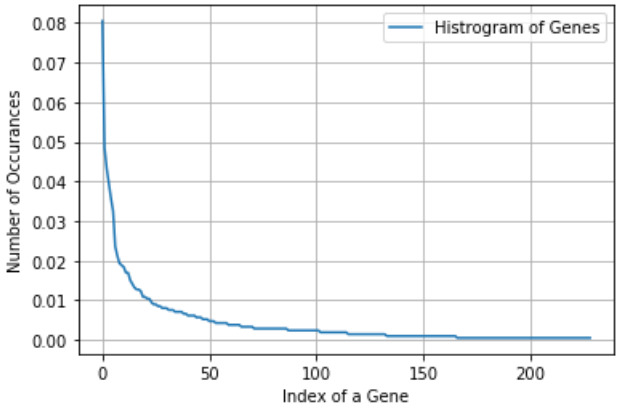
In [23]:

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

```
Ans: There are 229 different categories of genes in the train data, and they are distibuted as follows
```

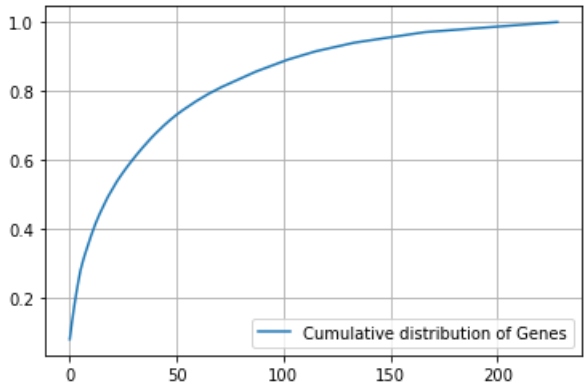
In [24]:

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



In [25]:

```
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 [26]:

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

In [27]:

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

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

In [28]:

```
# 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 [29]:

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

Out[29]:

```
378       TP53
264       EGFR
647     CDKN2A
109       MSH6
2258      PTEN
Name: Gene, dtype: object
```

In [30]:

```
gene_vectorizer.get_feature_names()
```

Out[30]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'b2m',
```

```
'bap1',
'bard1',
'bcl10',
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd2',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxo1',
'fubp1',
'gata3',
'gnas',
'h3f3a',
'hla',
'hnf1a',
```

```
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
```

```
 'ptprd',
 'ptprt',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad51d',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'ros1',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stat3',
 'stk11',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [31]:

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

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

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

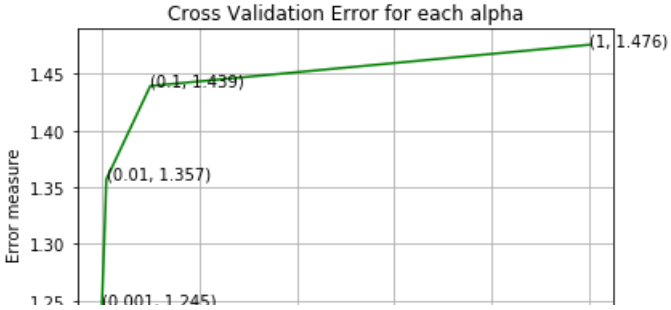
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

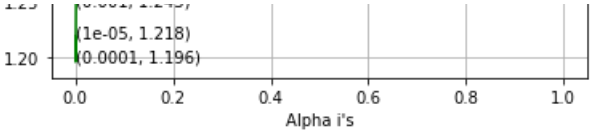
In [32]:

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

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

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

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


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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2177268968699069
For values of alpha =  0.0001 The log loss is: 1.1959637869813546
For values of alpha =  0.001 The log loss is: 1.2454413340698054
For values of alpha =  0.01 The log loss is: 1.3570109295847708
For values of alpha =  0.1 The log loss is: 1.439277561857853
For values of alpha =  1 The log loss is: 1.4756491491327497
```






```
For values of best alpha =  0.0001 The train log loss is: 0.9701598362103167
For values of best alpha =  0.0001 The cross validation log loss is: 1.1959637869813546
For values of best alpha =  0.0001 The test log loss is: 1.2305320948393508
```

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 [33]:

```
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  229  genes in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 516 out of  532 : 96.99248120300751
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [34]:

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

```
Number of Unique Variations : 1927
Truncating_Mutations    64
Amplification           48
Deletion                47
Fusions                 22
Overexpression           3
G12V                     3
M1R                      2
A146T                    2
R173C                    2
G12A                     2
Name: Variation, dtype: int64
```

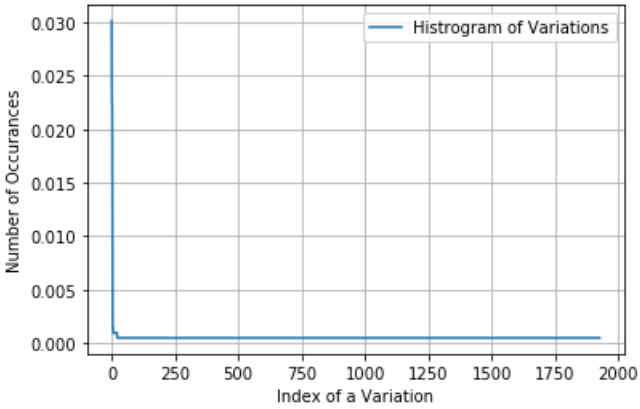
In [35]:

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

```
Ans: There are 1927 different categories of variations in the train data, and they are distibuted
as follows
```

In [36]:

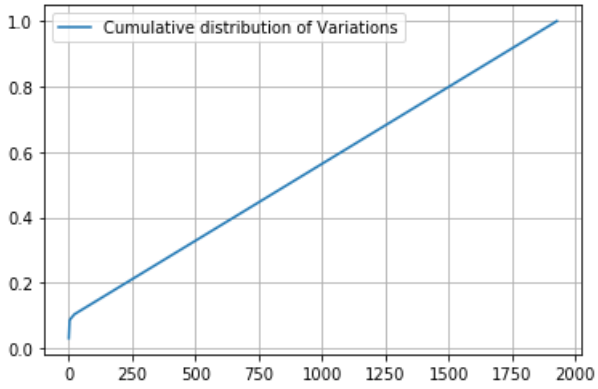
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [37]:

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

```
[0.03013183 0.0527307  0.07485876 ... 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 [38]:

```
# 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 [39]:

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

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

In [40]:

```
# 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 [41]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1957)
```

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

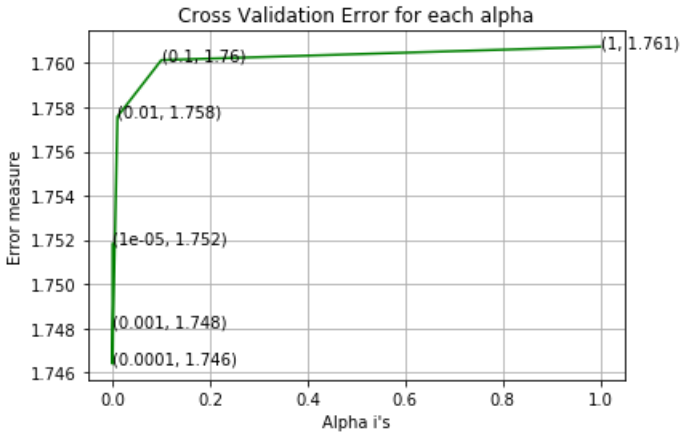
Let's build a model just like the earlier!

In [42]:

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

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

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

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


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

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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7518200385562332
For values of alpha =  0.0001 The log loss is: 1.7463933126836697
For values of alpha =  0.001 The log loss is: 1.748099952937582
For values of alpha =  0.01 The log loss is: 1.7575758919037199
For values of alpha =  0.1 The log loss is: 1.760133449432119
For values of alpha =  1 The log loss is: 1.7607367071487894
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6250017733729076
For values of best alpha =  0.0001 The cross validation log loss is: 1.7463933126836697
For values of best alpha =  0.0001 The test log loss is: 1.7037345515630389
```

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 [43]:

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

```
Q12. How many data points are covered by total  1927  genes in test and cross validation data
sets?
Ans
1. In test data 73 out of 665 : 10.977443609022556
2. In cross validation data 47 out of  532 : 8.834586466165414
```

3.2.3 Univariate Analysis on Text Feature

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

In [44]:

```
# 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 [45]:

```
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 [46]:

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

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

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


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

```
Total number of unique words in train data : 53874
```

In [47]:

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

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


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

In [48]:

```
#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 [49]:

```
# 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 [50]:

```
# 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 [51]:

```
#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 [52]:

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

```
Counter({3: 5467, 4: 3913, 5: 2887, 6: 2855, 8: 2384, 7: 2340, 9: 1786, 10: 1474, 12: 1219, 11: 114
8, 14: 958, 16: 939, 15: 768, 13: 732, 18: 687, 17: 678, 24: 642, 20: 565, 21: 490, 19: 476, 22: 4
34, 23: 426, 28: 405, 30: 361, 27: 359, 25: 348, 26: 326, 46: 307, 34: 300, 32: 290, 55: 270, 31:
268, 33: 264, 36: 261, 29: 257, 35: 254, 40: 212, 39: 200, 42: 197, 43: 195, 48: 189, 37: 187, 38:
177, 49: 176, 45: 174, 44: 174, 41: 173, 54: 166, 51: 163, 50: 152, 57: 143, 53: 143, 47: 141, 56:
140, 60: 133, 52: 131, 63: 128, 59: 117, 58: 117, 64: 114, 72: 110, 65: 107, 66: 103, 74: 99, 70:
96, 69: 96, 68: 96, 86: 95, 67: 93, 71: 91, 61: 91, 62: 90, 87: 83, 78: 81, 73: 79, 88: 77, 92: 7
6, 80: 71, 77: 71, 84: 70, 75: 70, 91: 68, 76: 68, 110: 66, 96: 66, 89: 66, 85: 66, 82: 66, 81: 6
5, 90: 63, 103: 62, 83: 61, 98: 60, 94: 59, 93: 59, 79: 59, 113: 54, 95: 54, 99: 53, 120: 52, 108
: 52, 112: 51, 100: 51, 105: 50, 106: 49, 97: 49, 114: 48, 107: 48, 111: 46, 109: 46, 104: 46, 139
: 45, 144: 44, 116: 44, 138: 43, 135: 43, 132: 43, 121: 43, 140: 42, 102: 42, 101: 42, 126: 41, 12
5: 41, 118: 41, 117: 41, 145: 39, 129: 39, 151: 38, 127: 38, 119: 38, 137: 37, 115: 37, 122: 36, 1
53: 32, 134: 32, 183: 31, 172: 31, 169: 31, 155: 31, 147: 31, 133: 31, 128: 31, 124: 31, 180: 30,
```

142: 30, 123: 30, 193: 29, 168: 29, 148: 29, 141: 29, 136: 29, 163: 28, 162: 28, 161: 28, 219: 27,
196: 27, 177: 27, 165: 27, 146: 27, 130: 27, 215: 26, 190: 26, 187: 26, 175: 26, 167: 26, 158: 26,
152: 26, 150: 26, 143: 26, 131: 26, 159: 25, 157: 25, 149: 25, 205: 24, 166: 24, 160: 24, 203: 23,
200: 23, 186: 23, 184: 23, 174: 23, 243: 22, 211: 22, 209: 22, 201: 22, 194: 22, 176: 22, 225: 21,
216: 21, 189: 21, 182: 21, 164: 21, 154: 21, 337: 20, 259: 20, 231: 20, 228: 20, 213: 20, 212: 20,
191: 20, 181: 20, 249: 19, 233: 19, 220: 19, 218: 19, 214: 19, 202: 19, 188: 19, 185: 19, 178: 19,
156: 19, 282: 18, 269: 18, 245: 18, 239: 18, 227: 18, 222: 18, 207: 18, 204: 18, 278: 17, 253: 17,
248: 17, 230: 17, 199: 17, 173: 17, 171: 17, 170: 17, 346: 16, 296: 16, 288: 16, 283: 16, 266: 16,
255: 16, 226: 16, 223: 16, 195: 16, 424: 15, 344: 15, 321: 15, 309: 15, 293: 15, 280: 15, 279: 15,
275: 15, 273: 15, 264: 15, 262: 15, 250: 15, 237: 15, 235: 15, 234: 15, 232: 15, 221: 15, 217: 15,
206: 15, 531: 14, 326: 14, 323: 14, 303: 14, 299: 14, 298: 14, 276: 14, 258: 14, 251: 14, 246: 14,
198: 14, 192: 14, 421: 13, 327: 13, 304: 13, 268: 13, 242: 13, 229: 13, 224: 13, 356: 12, 340: 12,
322: 12, 313: 12, 312: 12, 291: 12, 277: 12, 263: 12, 257: 12, 240: 12, 210: 12, 378: 11, 373: 11,
367: 11, 357: 11, 329: 11, 325: 11, 307: 11, 301: 11, 284: 11, 274: 11, 256: 11, 252: 11, 445: 10,
389: 10, 386: 10, 380: 10, 352: 10, 349: 10, 324: 10, 308: 10, 305: 10, 300: 10, 290: 10, 287: 10,
285: 10, 270: 10, 267: 10, 265: 10, 241: 10, 238: 10, 197: 10, 179: 10, 524: 9, 469: 9, 459: 9, 43
9: 9, 430: 9, 422: 9, 403: 9, 397: 9, 384: 9, 363: 9, 355: 9, 351: 9, 342: 9, 336: 9, 333: 9, 320:
9, 318: 9, 281: 9, 260: 9, 254: 9, 247: 9, 236: 9, 208: 9, 873: 8, 733: 8, 595: 8, 535: 8, 501: 8,
482: 8, 452: 8, 440: 8, 435: 8, 427: 8, 425: 8, 417: 8, 406: 8, 404: 8, 400: 8, 398: 8, 379: 8,
371: 8, 360: 8, 353: 8, 350: 8, 341: 8, 339: 8, 338: 8, 334: 8, 332: 8, 331: 8, 314: 8, 310: 8,
297: 8, 292: 8, 289: 8, 271: 8, 261: 8, 815: 7, 730: 7, 646: 7, 624: 7, 606: 7, 593: 7, 583: 7,
566: 7, 561: 7, 546: 7, 534: 7, 526: 7, 520: 7, 514: 7, 513: 7, 508: 7, 500: 7, 485: 7, 466: 7,
455: 7, 453: 7, 447: 7, 442: 7, 436: 7, 433: 7, 423: 7, 420: 7, 419: 7, 418: 7, 416: 7, 392: 7,
391: 7, 376: 7, 375: 7, 374: 7, 361: 7, 358: 7, 354: 7, 328: 7, 319: 7, 317: 7, 316: 7, 315: 7,
311: 7, 302: 7, 294: 7, 244: 7, 1258: 6, 1180: 6, 1007: 6, 862: 6, 782: 6, 764: 6, 725: 6, 713: 6,
702: 6, 687: 6, 659: 6, 650: 6, 648: 6, 613: 6, 605: 6, 576: 6, 564: 6, 563: 6, 555: 6, 550: 6,
518: 6, 517: 6, 506: 6, 504: 6, 499: 6, 496: 6, 490: 6, 478: 6, 471: 6, 468: 6, 454: 6, 432: 6,
408: 6, 388: 6, 382: 6, 381: 6, 368: 6, 366: 6, 364: 6, 345: 6, 335: 6, 330: 6, 272: 6, 1753: 5,
1354: 5, 1225: 5, 1210: 5, 1196: 5, 1165: 5, 963: 5, 907: 5, 896: 5, 894: 5, 881: 5, 836: 5, 804: 5
, 787: 5, 759: 5, 755: 5, 754: 5, 746: 5, 745: 5, 743: 5, 732: 5, 715: 5, 705: 5, 685: 5, 670: 5,
669: 5, 661: 5, 658: 5, 656: 5, 647: 5, 644: 5, 640: 5, 634: 5, 630: 5, 618: 5, 608: 5, 594: 5,
586: 5, 577: 5, 575: 5, 553: 5, 548: 5, 543: 5, 541: 5, 532: 5, 525: 5, 516: 5, 503: 5, 494: 5,
479: 5, 475: 5, 474: 5, 472: 5, 467: 5, 463: 5, 462: 5, 450: 5, 449: 5, 448: 5, 446: 5, 444: 5,
443: 5, 441: 5, 438: 5, 434: 5, 431: 5, 428: 5, 415: 5, 414: 5, 412: 5, 405: 5, 395: 5, 394: 5,
393: 5, 370: 5, 362: 5, 306: 5, 286: 5, 1976: 4, 1910: 4, 1865: 4, 1846: 4, 1559: 4, 1473: 4, 1455:
4, 1440: 4, 1421: 4, 1369: 4, 1312: 4, 1297: 4, 1255: 4, 1199: 4, 1191: 4, 1179: 4, 1131: 4, 1090:
4, 1078: 4, 1046: 4, 1040: 4, 977: 4, 961: 4, 957: 4, 940: 4, 921: 4, 886: 4, 871: 4, 868: 4, 866:
4, 832: 4, 801: 4, 797: 4, 796: 4, 774: 4, 769: 4, 744: 4, 742: 4, 736: 4, 728: 4, 727: 4, 720: 4,
717: 4, 704: 4, 701: 4, 693: 4, 690: 4, 679: 4, 674: 4, 662: 4, 652: 4, 649: 4, 642: 4, 639: 4,
637: 4, 631: 4, 619: 4, 617: 4, 616: 4, 614: 4, 611: 4, 604: 4, 602: 4, 601: 4, 597: 4, 592: 4,
590: 4, 588: 4, 580: 4, 562: 4, 558: 4, 554: 4, 552: 4, 545: 4, 537: 4, 536: 4, 512: 4, 509: 4,
507: 4, 505: 4, 498: 4, 487: 4, 483: 4, 480: 4, 477: 4, 473: 4, 470: 4, 465: 4, 464: 4, 461: 4,
458: 4, 457: 4, 451: 4, 429: 4, 426: 4, 413: 4, 411: 4, 410: 4, 409: 4, 407: 4, 401: 4, 399: 4,
396: 4, 377: 4, 372: 4, 365: 4, 348: 4, 295: 4, 7760: 3, 5401: 3, 2780: 3, 2581: 3, 2305: 3, 2134:
3, 2133: 3, 2056: 3, 2003: 3, 1985: 3, 1980: 3, 1858: 3, 1831: 3, 1812: 3, 1782: 3, 1729: 3, 1715:
3, 1696: 3, 1666: 3, 1660: 3, 1597: 3, 1589: 3, 1585: 3, 1571: 3, 1562: 3, 1540: 3, 1503: 3, 1502:
3, 1489: 3, 1463: 3, 1414: 3, 1400: 3, 1389: 3, 1381: 3, 1378: 3, 1357: 3, 1346: 3, 1344: 3, 1315:
3, 1309: 3, 1306: 3, 1283: 3, 1268: 3, 1265: 3, 1262: 3, 1243: 3, 1237: 3, 1233: 3, 1215: 3, 1158:
3, 1155: 3, 1141: 3, 1134: 3, 1114: 3, 1081: 3, 1076: 3, 1066: 3, 1065: 3, 1064: 3, 1063: 3, 1055:
3, 1050: 3, 1049: 3, 1029: 3, 1019: 3, 1014: 3, 1008: 3, 1001: 3, 998: 3, 995: 3, 987: 3, 973: 3, 9
71: 3, 966: 3, 954: 3, 949: 3, 948: 3, 939: 3, 938: 3, 928: 3, 914: 3, 904: 3, 903: 3, 897: 3,
895: 3, 891: 3, 883: 3, 878: 3, 876: 3, 875: 3, 869: 3, 867: 3, 863: 3, 846: 3, 844: 3, 843: 3,
831: 3, 825: 3, 824: 3, 823: 3, 822: 3, 817: 3, 811: 3, 805: 3, 798: 3, 794: 3, 786: 3, 776: 3,
773: 3, 772: 3, 771: 3, 765: 3, 760: 3, 757: 3, 753: 3, 749: 3, 741: 3, 739: 3, 731: 3, 723: 3,
714: 3, 711: 3, 708: 3, 699: 3, 698: 3, 697: 3, 696: 3, 695: 3, 694: 3, 689: 3, 684: 3, 683: 3,
668: 3, 667: 3, 663: 3, 657: 3, 645: 3, 633: 3, 629: 3, 622: 3, 600: 3, 599: 3, 591: 3, 589: 3,
585: 3, 584: 3, 582: 3, 581: 3, 579: 3, 578: 3, 573: 3, 565: 3, 559: 3, 557: 3, 556: 3, 549: 3,
542: 3, 540: 3, 539: 3, 530: 3, 528: 3, 521: 3, 515: 3, 510: 3, 502: 3, 497: 3, 489: 3, 481: 3,
476: 3, 460: 3, 456: 3, 437: 3, 390: 3, 383: 3, 369: 3, 343: 3, 15403: 2, 12264: 2, 12079: 2,
9486: 2, 8643: 2, 7014: 2, 6928: 2, 5884: 2, 5664: 2, 5312: 2, 5262: 2, 5087: 2, 4919: 2, 4636: 2,
4589: 2, 4274: 2, 4267: 2, 4240: 2, 4123: 2, 4030: 2, 3989: 2, 3896: 2, 3860: 2, 3747: 2, 3714: 2,
3662: 2, 3628: 2, 3595: 2, 3542: 2, 3532: 2, 3529: 2, 3523: 2, 3502: 2, 3482: 2, 3450: 2, 3382: 2,
3352: 2, 3338: 2, 3327: 2, 3280: 2, 3194: 2, 3176: 2, 3135: 2, 3082: 2, 3075: 2, 3062: 2, 3018: 2,
3011: 2, 2970: 2, 2959: 2, 2933: 2, 2911: 2, 2841: 2, 2748: 2, 2718: 2, 2684: 2, 2673: 2, 2611: 2,
2597: 2, 2591: 2, 2571: 2, 2547: 2, 2528: 2, 2519: 2, 2511: 2, 2508: 2, 2487: 2, 2481: 2, 2457: 2,
2443: 2, 2441: 2, 2426: 2, 2416: 2, 2371: 2, 2367: 2, 2348: 2, 2342: 2, 2294: 2, 2261: 2, 2257: 2,
2203: 2, 2201: 2, 2189: 2, 2177: 2, 2176: 2, 2131: 2, 2113: 2, 2100: 2, 2097: 2, 2085: 2, 2082: 2,
2079: 2, 2044: 2, 2035: 2, 2029: 2, 2026: 2, 2014: 2, 2012: 2, 1995: 2, 1989: 2, 1972: 2, 1963: 2,
1962: 2, 1958: 2, 1954: 2, 1943: 2, 1940: 2, 1934: 2, 1922: 2, 1921: 2, 1920: 2, 1911: 2, 1881: 2,
1864: 2, 1856: 2, 1842: 2, 1841: 2, 1840: 2, 1827: 2, 1821: 2, 1809: 2, 1806: 2, 1791: 2, 1788: 2,
1787: 2, 1786: 2, 1776: 2, 1761: 2, 1751: 2, 1747: 2, 1730: 2, 1703: 2, 1690: 2, 1684: 2, 1659: 2,
1656: 2, 1648: 2, 1642: 2, 1624: 2, 1619: 2, 1615: 2, 1608: 2, 1607: 2, 1602: 2, 1584: 2, 1579: 2,
1573: 2, 1568: 2, 1566: 2, 1561: 2, 1556: 2, 1555: 2, 1546: 2, 1545: 2, 1543: 2, 1539: 2, 1536: 2,
1533: 2, 1522: 2, 1507: 2, 1501: 2, 1491: 2, 1484: 2, 1479: 2, 1476: 2, 1475: 2, 1468: 2, 1466: 2,
1461: 2, 1460: 2, 1459: 2, 1458: 2, 1457: 2, 1448: 2, 1442: 2, 1435: 2, 1433: 2, 1422: 2, 1408: 2,
1406: 2, 1402: 2, 1399: 2, 1396: 2, 1374: 2, 1360: 2, 1355: 2, 1335: 2, 1331: 2, 1329: 2, 1328: 2,
1321: 2, 1320: 2, 1319: 2, 1317: 2, 1316: 2, 1300: 2, 1295: 2, 1288: 2, 1285: 2, 1282: 2, 1278: 2,

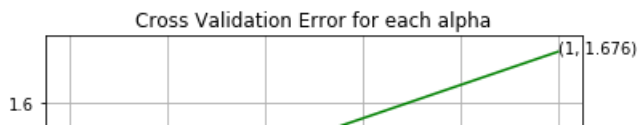
1277: 2, 1274: 2, 1272: 2, 1271: 2, 1263: 2, 1256: 2, 1248: 2, 1247: 2, 1242: 2, 1241: 2, 1235: 2,
1230: 2, 1216: 2, 1212: 2, 1209: 2, 1208: 2, 1203: 2, 1197: 2, 1195: 2, 1194: 2, 1193: 2, 1188: 2,
1178: 2, 1162: 2, 1160: 2, 1151: 2, 1150: 2, 1149: 2, 1145: 2, 1144: 2, 1143: 2, 1142: 2, 1138: 2,
1136: 2, 1135: 2, 1132: 2, 1127: 2, 1126: 2, 1125: 2, 1122: 2, 1111: 2, 1108: 2, 1105: 2, 1101: 2,
1097: 2, 1095: 2, 1088: 2, 1087: 2, 1077: 2, 1073: 2, 1071: 2, 1069: 2, 1061: 2, 1060: 2, 1059: 2,
1054: 2, 1053: 2, 1052: 2, 1042: 2, 1038: 2, 1036: 2, 1033: 2, 1032: 2, 1030: 2, 1022: 2, 1021: 2,
1011: 2, 1009: 2, 1002: 2, 997: 2, 994: 2, 989: 2, 986: 2, 984: 2, 982: 2, 981: 2, 976: 2, 970: 2,
968: 2, 964: 2, 962: 2, 960: 2, 959: 2, 956: 2, 955: 2, 952: 2, 951: 2, 950: 2, 943: 2, 941: 2,
935: 2, 934: 2, 932: 2, 931: 2, 930: 2, 923: 2, 920: 2, 919: 2, 915: 2, 911: 2, 908: 2, 906: 2,
901: 2, 900: 2, 892: 2, 888: 2, 884: 2, 882: 2, 879: 2, 877: 2, 872: 2, 860: 2, 858: 2, 854: 2,
852: 2, 848: 2, 845: 2, 842: 2, 839: 2, 834: 2, 833: 2, 830: 2, 828: 2, 827: 2, 826: 2, 820: 2,
819: 2, 816: 2, 810: 2, 808: 2, 807: 2, 806: 2, 795: 2, 793: 2, 792: 2, 790: 2, 789: 2, 783: 2,
781: 2, 779: 2, 778: 2, 775: 2, 770: 2, 767: 2, 762: 2, 761: 2, 756: 2, 751: 2, 750: 2, 748: 2,
738: 2, 729: 2, 726: 2, 721: 2, 719: 2, 712: 2, 709: 2, 706: 2, 700: 2, 691: 2, 686: 2, 682: 2,
681: 2, 680: 2, 676: 2, 673: 2, 664: 2, 655: 2, 654: 2, 653: 2, 638: 2, 636: 2, 635: 2, 628: 2,
626: 2, 623: 2, 621: 2, 620: 2, 615: 2, 612: 2, 607: 2, 603: 2, 598: 2, 587: 2, 572: 2, 571: 2,
569: 2, 568: 2, 551: 2, 547: 2, 544: 2, 538: 2, 533: 2, 529: 2, 523: 2, 522: 2, 511: 2, 493: 2,
491: 2, 402: 2, 387: 2, 385: 2, 359: 2, 347: 2, 149196: 1, 116892: 1, 80088: 1, 65941: 1, 65805: 1
, 64176: 1, 63978: 1, 62362: 1, 61653: 1, 53897: 1, 53641: 1, 49844: 1, 49172: 1, 46168: 1, 45666:
1, 43407: 1, 42760: 1, 41619: 1, 41323: 1, 41218: 1, 40179: 1, 39985: 1, 39614: 1, 39394: 1, 38359
: 1, 37502: 1, 36485: 1, 36349: 1, 35928: 1, 35316: 1, 34228: 1, 33753: 1, 32724: 1, 32610: 1, 311
13: 1, 30996: 1, 29050: 1, 27701: 1, 26582: 1, 26170: 1, 25717: 1, 25695: 1, 25691: 1, 25595: 1, 2
4753: 1, 24525: 1, 24424: 1, 24132: 1, 24063: 1, 24034: 1, 23788: 1, 23374: 1, 23049: 1, 21906: 1,
21583: 1, 21424: 1, 21345: 1, 21207: 1, 21084: 1, 20923: 1, 20849: 1, 20545: 1, 19909: 1, 19841: 1
, 19834: 1, 19302: 1, 19064: 1, 19062: 1, 18969: 1, 18944: 1, 18800: 1, 18722: 1, 18442: 1, 18420:
1, 18328: 1, 18281: 1, 17916: 1, 17793: 1, 17713: 1, 17647: 1, 17535: 1, 17496: 1, 17489: 1, 17293
: 1, 17291: 1, 17255: 1, 17135: 1, 17036: 1, 16961: 1, 16885: 1, 16839: 1, 16607: 1, 16594: 1, 165
52: 1, 16312: 1, 16130: 1, 16024: 1, 15555: 1, 15483: 1, 15424: 1, 15413: 1, 15380: 1, 15362: 1, 1
5292: 1, 15163: 1, 14920: 1, 14905: 1, 14885: 1, 14713: 1, 14665: 1, 14572: 1, 14475: 1, 14150: 1,
14121: 1, 14092: 1, 14088: 1, 14076: 1, 13770: 1, 13638: 1, 13588: 1, 13456: 1, 13396: 1, 13252: 1
, 13115: 1, 13077: 1, 12960: 1, 12927: 1, 12890: 1, 12856: 1, 12855: 1, 12784: 1, 12770: 1, 12748:
1, 12631: 1, 12626: 1, 12440: 1, 12356: 1, 12283: 1, 12258: 1, 12226: 1, 12202: 1, 12157: 1, 12154
: 1, 12142: 1, 12135: 1, 12124: 1, 12117: 1, 12016: 1, 12010: 1, 11997: 1, 11964: 1, 11936: 1, 119
24: 1, 11898: 1, 11854: 1, 11829: 1, 11800: 1, 11797: 1, 11787: 1, 11780: 1, 11694: 1, 11664: 1, 1
1659: 1, 11603: 1, 11568: 1, 11507: 1, 11424: 1, 11290: 1, 11289: 1, 11267: 1, 11248: 1, 11199: 1,
11109: 1, 10935: 1, 10918: 1, 10753: 1, 10751: 1, 10738: 1, 10712: 1, 10709: 1, 10639: 1, 10538: 1
, 10532: 1, 10443: 1, 10438: 1, 10319: 1, 10241: 1, 10220: 1, 10078: 1, 10066: 1, 9957: 1, 9948: 1
, 9844: 1, 9828: 1, 9811: 1, 9789: 1, 9780: 1, 9730: 1, 9727: 1, 9703: 1, 9649: 1, 9646: 1, 9580: 1
, 9553: 1, 9531: 1, 9512: 1, 9501: 1, 9497: 1, 9433: 1, 9354: 1, 9345: 1, 9285: 1, 9282: 1, 9270: 1
, 9244: 1, 9155: 1, 9130: 1, 9041: 1, 9036: 1, 9007: 1, 8982: 1, 8960: 1, 8950: 1, 8934: 1, 8933: 1
, 8920: 1, 8919: 1, 8900: 1, 8885: 1, 8879: 1, 8873: 1, 8866: 1, 8859: 1, 8827: 1, 8741: 1, 8738: 1
, 8712: 1, 8639: 1, 8604: 1, 8434: 1, 8422: 1, 8385: 1, 8380: 1, 8324: 1, 8181: 1, 8141: 1, 8140: 1
, 8139: 1, 8106: 1, 8105: 1, 8094: 1, 8060: 1, 8035: 1, 7982: 1, 7969: 1, 7940: 1, 7925: 1, 7921: 1
, 7915: 1, 7911: 1, 7902: 1, 7879: 1, 7834: 1, 7831: 1, 7810: 1, 7806: 1, 7722: 1, 7710: 1, 7678: 1
, 7669: 1, 7626: 1, 7623: 1, 7611: 1, 7540: 1, 7531: 1, 7516: 1, 7493: 1, 7492: 1, 7470: 1, 7452: 1
, 7407: 1, 7381: 1, 7363: 1, 7343: 1, 7335: 1, 7327: 1, 7307: 1, 7299: 1, 7295: 1, 7289: 1, 7277: 1
, 7273: 1, 7264: 1, 7258: 1, 7251: 1, 7186: 1, 7184: 1, 7172: 1, 7145: 1, 7131: 1, 7123: 1, 7109: 1
, 7099: 1, 7094: 1, 7093: 1, 7092: 1, 7056: 1, 7018: 1, 6998: 1, 6956: 1, 6954: 1, 6944: 1, 6937: 1
, 6936: 1, 6911: 1, 6886: 1, 6885: 1, 6835: 1, 6831: 1, 6802: 1, 6797: 1, 6780: 1, 6773: 1, 6749: 1
, 6731: 1, 6711: 1, 6704: 1, 6680: 1, 6667: 1, 6653: 1, 6643: 1, 6629: 1, 6604: 1, 6599: 1, 6556: 1
, 6552: 1, 6524: 1, 6478: 1, 6457: 1, 6434: 1, 6414: 1, 6398: 1, 6388: 1, 6374: 1, 6368: 1, 6363: 1
, 6330: 1, 6288: 1, 6278: 1, 6275: 1, 6274: 1, 6272: 1, 6270: 1, 6238: 1, 6236: 1, 6234: 1, 6219: 1
, 6201: 1, 6183: 1, 6176: 1, 6166: 1, 6160: 1, 6120: 1, 6117: 1, 6110: 1, 6107: 1, 6104: 1, 6099: 1
, 6090: 1, 6089: 1, 6083: 1, 6069: 1, 6037: 1, 6023: 1, 6012: 1, 6006: 1, 5984: 1, 5914: 1, 5890: 1
, 5879: 1, 5874: 1, 5865: 1, 5845: 1, 5791: 1, 5776: 1, 5762: 1, 5761: 1, 5742: 1, 5734: 1, 5713: 1
, 5712: 1, 5700: 1, 5697: 1, 5692: 1, 5679: 1, 5652: 1, 5601: 1, 5598: 1, 5589: 1, 5571: 1, 5570: 1
, 5556: 1, 5534: 1, 5533: 1, 5522: 1, 5521: 1, 5498: 1, 5464: 1, 5459: 1, 5439: 1, 5437: 1, 5397: 1
, 5369: 1, 5357: 1, 5350: 1, 5317: 1, 5313: 1, 5288: 1, 5284: 1, 5275: 1, 5244: 1, 5222: 1, 5200: 1
, 5180: 1, 5175: 1, 5173: 1, 5172: 1, 5160: 1, 5155: 1, 5150: 1, 5145: 1, 5138: 1, 5134: 1, 5131: 1
, 5117: 1, 5116: 1, 5090: 1, 5085: 1, 5034: 1, 5031: 1, 5029: 1, 5003: 1, 4993: 1, 4989: 1, 4988: 1
, 4982: 1, 4980: 1, 4952: 1, 4944: 1, 4921: 1, 4911: 1, 4906: 1, 4899: 1, 4888: 1, 4881: 1, 4875: 1
, 4873: 1, 4851: 1, 4839: 1, 4836: 1, 4822: 1, 4821: 1, 4819: 1, 4817: 1, 4809: 1, 4798: 1, 4779: 1
, 4773: 1, 4771: 1, 4769: 1, 4761: 1, 4755: 1, 4751: 1, 4749: 1, 4737: 1, 4731: 1, 4724: 1, 4717: 1
, 4702: 1, 4698: 1, 4697: 1, 4692: 1, 4689: 1, 4680: 1, 4678: 1, 4675: 1, 4662: 1, 4657: 1, 4653: 1
, 4640: 1, 4639: 1, 4630: 1, 4620: 1, 4601: 1, 4590: 1, 4575: 1, 4574: 1, 4565: 1, 4551: 1, 4533: 1
, 4527: 1, 4516: 1, 4480: 1, 4455: 1, 4444: 1, 4436: 1, 4431: 1, 4419: 1, 4417: 1, 4407: 1, 4390: 1
, 4382: 1, 4376: 1, 4372: 1, 4357: 1, 4342: 1, 4341: 1, 4335: 1, 4333: 1, 4332: 1, 4305: 1, 4291: 1
, 4287: 1, 4279: 1, 4266: 1, 4265: 1, 4262: 1, 4251: 1, 4247: 1, 4242: 1, 4241: 1, 4237: 1, 4231: 1
, 4202: 1, 4199: 1, 4194: 1, 4189: 1, 4185: 1, 4168: 1, 4158: 1, 4147: 1, 4142: 1, 4129: 1, 4128: 1
, 4127: 1, 4121: 1, 4118: 1, 4116: 1, 4113: 1, 4109: 1, 4101: 1, 4100: 1, 4096: 1, 4090: 1, 4085: 1
, 4084: 1, 4083: 1, 4077: 1, 4060: 1, 4053: 1, 4042: 1, 4033: 1, 4022: 1, 4021: 1, 4014: 1, 4011: 1
, 4009: 1, 4005: 1, 3999: 1, 3995: 1, 3994: 1, 3968: 1, 3966: 1, 3959: 1, 3955: 1, 3946: 1, 3941: 1
, 3939: 1, 3938: 1, 3935: 1, 3932: 1, 3931: 1, 3930: 1, 3919: 1, 3915: 1, 3905: 1, 3902: 1, 3891: 1
, 3881: 1, 3869: 1, 3868: 1, 3867: 1, 3863: 1, 3857: 1, 3846: 1, 3842: 1, 3834: 1, 3819: 1, 3805: 1
, 3804: 1, 3796: 1, 3795: 1, 3791: 1, 3782: 1, 3771: 1, 3762: 1, 3759: 1, 3758: 1, 3752: 1, 3750: 1
, 3748: 1, 3746: 1, 3740: 1, 3735: 1, 3730: 1, 3728: 1, 3721: 1, 3688: 1, 3684: 1, 3680: 1, 3678: 1
, 3675: 1, 3669: 1, 3663: 1, 3658: 1, 3641: 1, 3619: 1, 3615: 1, 3612: 1, 3610: 1, 3605: 1, 3596: 1

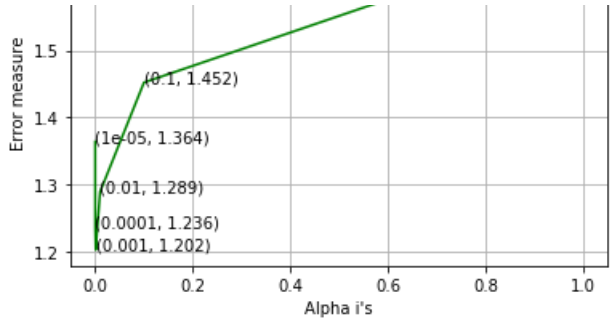
, 3593: 1, 3589: 1, 3588: 1, 3582: 1, 3581: 1, 3578: 1, 3565: 1, 3560: 1, 3552: 1, 3551: 1, 3545: 1
, 3539: 1, 3533: 1, 3531: 1, 3518: 1, 3515: 1, 3513: 1, 3510: 1, 3509: 1, 3506: 1, 3501: 1, 3493: 1
, 3486: 1, 3475: 1, 3474: 1, 3466: 1, 3465: 1, 3463: 1, 3458: 1, 3453: 1, 3452: 1, 3451: 1, 3438: 1
, 3436: 1, 3435: 1, 3431: 1, 3425: 1, 3424: 1, 3420: 1, 3403: 1, 3397: 1, 3391: 1, 3390: 1, 3389: 1
, 3384: 1, 3379: 1, 3378: 1, 3362: 1, 3360: 1, 3357: 1, 3355: 1, 3345: 1, 3339: 1, 3332: 1, 3331: 1
, 3324: 1, 3321: 1, 3316: 1, 3315: 1, 3312: 1, 3310: 1, 3304: 1, 3303: 1, 3300: 1, 3298: 1, 3297: 1
, 3294: 1, 3281: 1, 3276: 1, 3270: 1, 3266: 1, 3265: 1, 3264: 1, 3263: 1, 3260: 1, 3257: 1, 3255: 1
, 3246: 1, 3242: 1, 3236: 1, 3234: 1, 3220: 1, 3218: 1, 3213: 1, 3210: 1, 3204: 1, 3191: 1, 3181: 1
, 3180: 1, 3177: 1, 3168: 1, 3162: 1, 3152: 1, 3151: 1, 3147: 1, 3146: 1, 3143: 1, 3142: 1, 3138: 1
, 3132: 1, 3124: 1, 3122: 1, 3113: 1, 3111: 1, 3104: 1, 3102: 1, 3101: 1, 3100: 1, 3099: 1, 3097: 1
, 3095: 1, 3093: 1, 3086: 1, 3085: 1, 3080: 1, 3078: 1, 3076: 1, 3069: 1, 3068: 1, 3067: 1, 3063: 1
, 3060: 1, 3055: 1, 3032: 1, 3029: 1, 3027: 1, 3022: 1, 3012: 1, 3009: 1, 3002: 1, 3001: 1, 2981: 1
, 2967: 1, 2955: 1, 2951: 1, 2948: 1, 2946: 1, 2940: 1, 2929: 1, 2921: 1, 2918: 1, 2917: 1, 2909: 1
, 2901: 1, 2895: 1, 2893: 1, 2889: 1, 2879: 1, 2874: 1, 2871: 1, 2869: 1, 2868: 1, 2862: 1, 2859: 1
, 2854: 1, 2852: 1, 2851: 1, 2850: 1, 2848: 1, 2845: 1, 2843: 1, 2839: 1, 2832: 1, 2830: 1, 2829: 1
, 2828: 1, 2827: 1, 2822: 1, 2821: 1, 2820: 1, 2811: 1, 2810: 1, 2809: 1, 2807: 1, 2805: 1, 2801: 1
, 2789: 1, 2788: 1, 2785: 1, 2775: 1, 2773: 1, 2769: 1, 2767: 1, 2752: 1, 2751: 1, 2747: 1, 2728: 1
, 2712: 1, 2711: 1, 2710: 1, 2708: 1, 2707: 1, 2690: 1, 2681: 1, 2680: 1, 2678: 1, 2671: 1, 2667: 1
, 2666: 1, 2662: 1, 2658: 1, 2656: 1, 2655: 1, 2654: 1, 2653: 1, 2652: 1, 2647: 1, 2646: 1, 2643: 1
, 2642: 1, 2639: 1, 2638: 1, 2636: 1, 2627: 1, 2626: 1, 2621: 1, 2615: 1, 2608: 1, 2604: 1, 2585: 1
, 2583: 1, 2580: 1, 2575: 1, 2568: 1, 2567: 1, 2563: 1, 2562: 1, 2559: 1, 2556: 1, 2553: 1, 2551: 1
, 2548: 1, 2545: 1, 2544: 1, 2542: 1, 2538: 1, 2536: 1, 2524: 1, 2523: 1, 2520: 1, 2517: 1, 2516: 1
, 2515: 1, 2509: 1, 2501: 1, 2496: 1, 2493: 1, 2490: 1, 2489: 1, 2488: 1, 2477: 1, 2476: 1, 2469: 1
, 2466: 1, 2465: 1, 2464: 1, 2463: 1, 2462: 1, 2453: 1, 2451: 1, 2446: 1, 2427: 1, 2424: 1, 2423: 1
, 2422: 1, 2414: 1, 2413: 1, 2409: 1, 2404: 1, 2401: 1, 2400: 1, 2394: 1, 2393: 1, 2392: 1, 2389: 1
, 2379: 1, 2370: 1, 2365: 1, 2364: 1, 2362: 1, 2361: 1, 2358: 1, 2357: 1, 2356: 1, 2354: 1, 2352: 1
, 2347: 1, 2345: 1, 2344: 1, 2343: 1, 2340: 1, 2339: 1, 2337: 1, 2332: 1, 2326: 1, 2324: 1, 2323: 1
, 2322: 1, 2315: 1, 2311: 1, 2310: 1, 2307: 1, 2301: 1, 2299: 1, 2298: 1, 2296: 1, 2293: 1, 2292: 1
, 2291: 1, 2290: 1, 2283: 1, 2281: 1, 2273: 1, 2271: 1, 2264: 1, 2259: 1, 2256: 1, 2253: 1, 2250: 1
, 2249: 1, 2248: 1, 2246: 1, 2245: 1, 2244: 1, 2234: 1, 2230: 1, 2224: 1, 2222: 1, 2221: 1, 2219: 1
, 2218: 1, 2217: 1, 2215: 1, 2213: 1, 2211: 1, 2207: 1, 2204: 1, 2200: 1, 2198: 1, 2193: 1, 2190: 1
, 2188: 1, 2187: 1, 2185: 1, 2172: 1, 2168: 1, 2161: 1, 2157: 1, 2156: 1, 2152: 1, 2151: 1, 2150: 1
, 2148: 1, 2145: 1, 2143: 1, 2140: 1, 2139: 1, 2136: 1, 2123: 1, 2120: 1, 2118: 1, 2115: 1, 2112: 1
, 2111: 1, 2110: 1, 2108: 1, 2107: 1, 2104: 1, 2103: 1, 2098: 1, 2095: 1, 2091: 1, 2090: 1, 2088: 1
, 2087: 1, 2086: 1, 2084: 1, 2081: 1, 2080: 1, 2075: 1, 2071: 1, 2070: 1, 2069: 1, 2067: 1, 2066: 1
, 2064: 1, 2055: 1, 2054: 1, 2051: 1, 2050: 1, 2045: 1, 2040: 1, 2039: 1, 2036: 1, 2034: 1, 2031: 1
, 2030: 1, 2028: 1, 2022: 1, 2017: 1, 2006: 1, 2001: 1, 1998: 1, 1997: 1, 1996: 1, 1994: 1, 1991: 1
, 1990: 1, 1988: 1, 1984: 1, 1982: 1, 1967: 1, 1960: 1, 1959: 1, 1956: 1, 1952: 1, 1950: 1, 1945: 1
, 1936: 1, 1935: 1, 1932: 1, 1928: 1, 1927: 1, 1925: 1, 1919: 1, 1917: 1, 1912: 1, 1909: 1, 1908: 1
, 1903: 1, 1901: 1, 1900: 1, 1895: 1, 1892: 1, 1889: 1, 1886: 1, 1883: 1, 1880: 1, 1879: 1, 1877: 1
, 1876: 1, 1875: 1, 1874: 1, 1871: 1, 1868: 1, 1863: 1, 1862: 1, 1861: 1, 1859: 1, 1857: 1, 1854: 1
, 1853: 1, 1851: 1, 1847: 1, 1844: 1, 1843: 1, 1837: 1, 1835: 1, 1833: 1, 1824: 1, 1822: 1, 1816: 1
, 1814: 1, 1811: 1, 1804: 1, 1802: 1, 1798: 1, 1796: 1, 1794: 1, 1793: 1, 1792: 1, 1785: 1, 1781: 1
, 1780: 1, 1778: 1, 1766: 1, 1762: 1, 1759: 1, 1758: 1, 1754: 1, 1752: 1, 1750: 1, 1746: 1, 1745: 1
, 1740: 1, 1738: 1, 1735: 1, 1731: 1, 1728: 1, 1726: 1, 1725: 1, 1722: 1, 1721: 1, 1719: 1, 1717: 1
, 1711: 1, 1707: 1, 1700: 1, 1699: 1, 1698: 1, 1693: 1, 1692: 1, 1689: 1, 1687: 1, 1686: 1, 1685: 1
, 1683: 1, 1681: 1, 1680: 1, 1679: 1, 1677: 1, 1676: 1, 1673: 1, 1671: 1, 1670: 1, 1668: 1, 1665: 1
, 1661: 1, 1655: 1, 1654: 1, 1653: 1, 1652: 1, 1651: 1, 1650: 1, 1649: 1, 1647: 1, 1639: 1, 1638: 1
, 1637: 1, 1631: 1, 1630: 1, 1629: 1, 1626: 1, 1625: 1, 1623: 1, 1617: 1, 1609: 1, 1606: 1, 1605: 1
, 1604: 1, 1601: 1, 1599: 1, 1596: 1, 1595: 1, 1594: 1, 1592: 1, 1591: 1, 1587: 1, 1586: 1, 1583: 1
, 1576: 1, 1575: 1, 1574: 1, 1570: 1, 1569: 1, 1567: 1, 1565: 1, 1563: 1, 1558: 1, 1549: 1, 1547: 1
, 1542: 1, 1537: 1, 1535: 1, 1534: 1, 1528: 1, 1526: 1, 1516: 1, 1515: 1, 1511: 1, 1510: 1, 1509: 1
, 1506: 1, 1505: 1, 1504: 1, 1500: 1, 1498: 1, 1488: 1, 1480: 1, 1478: 1, 1471: 1, 1464: 1, 1462: 1
, 1456: 1, 1454: 1, 1452: 1, 1450: 1, 1449: 1, 1438: 1, 1429: 1, 1427: 1, 1425: 1, 1424: 1, 1423: 1
, 1420: 1, 1418: 1, 1417: 1, 1413: 1, 1412: 1, 1411: 1, 1403: 1, 1401: 1, 1395: 1, 1393: 1, 1392: 1
, 1390: 1, 1387: 1, 1384: 1, 1382: 1, 1380: 1, 1376: 1, 1375: 1, 1371: 1, 1370: 1, 1367: 1, 1362: 1
, 1359: 1, 1356: 1, 1351: 1, 1350: 1, 1349: 1, 1347: 1, 1345: 1, 1343: 1, 1342: 1, 1340: 1, 1336: 1
, 1334: 1, 1333: 1, 1322: 1, 1318: 1, 1314: 1, 1313: 1, 1311: 1, 1310: 1, 1307: 1, 1305: 1, 1302: 1
, 1299: 1, 1296: 1, 1291: 1, 1290: 1, 1289: 1, 1287: 1, 1284: 1, 1281: 1, 1280: 1, 1279: 1, 1276: 1
, 1273: 1, 1267: 1, 1264: 1, 1254: 1, 1253: 1, 1252: 1, 1251: 1, 1250: 1, 1249: 1, 1240: 1, 1238: 1
, 1232: 1, 1231: 1, 1229: 1, 1226: 1, 1224: 1, 1223: 1, 1222: 1, 1220: 1, 1219: 1, 1217: 1, 1214: 1
, 1213: 1, 1211: 1, 1205: 1, 1204: 1, 1202: 1, 1201: 1, 1200: 1, 1198: 1, 1192: 1, 1190: 1, 1189: 1
, 1187: 1, 1186: 1, 1185: 1, 1184: 1, 1182: 1, 1176: 1, 1174: 1, 1173: 1, 1172: 1, 1167: 1, 1163: 1
, 1161: 1, 1159: 1, 1153: 1, 1147: 1, 1146: 1, 1139: 1, 1137: 1, 1133: 1, 1130: 1, 1129: 1, 1128: 1
, 1121: 1, 1117: 1, 1115: 1, 1113: 1, 1109: 1, 1106: 1, 1104: 1, 1102: 1, 1100: 1, 1086: 1, 1085: 1
, 1084: 1, 1083: 1, 1082: 1, 1079: 1, 1075: 1, 1068: 1, 1067: 1, 1062: 1, 1058: 1, 1051: 1, 1048: 1
, 1047: 1, 1045: 1, 1043: 1, 1041: 1, 1031: 1, 1028: 1, 1027: 1, 1026: 1, 1025: 1, 1024: 1, 1018: 1
, 1016: 1, 1015: 1, 1013: 1, 1012: 1, 1010: 1, 1006: 1, 1005: 1, 1003: 1, 1000: 1, 999: 1, 996: 1,
993: 1, 991: 1, 985: 1, 983: 1, 979: 1, 978: 1, 975: 1, 972: 1, 969: 1, 967: 1, 953: 1, 947: 1,
944: 1, 942: 1, 937: 1, 933: 1, 929: 1, 926: 1, 924: 1, 922: 1, 918: 1, 917: 1, 916: 1, 913: 1,
912: 1, 910: 1, 905: 1, 902: 1, 899: 1, 898: 1, 890: 1, 889: 1, 887: 1, 870: 1, 864: 1, 861: 1,
859: 1, 856: 1, 855: 1, 851: 1, 849: 1, 847: 1, 840: 1, 838: 1, 837: 1, 835: 1, 829: 1, 818: 1,
814: 1, 813: 1, 812: 1, 802: 1, 800: 1, 791: 1, 784: 1, 780: 1, 777: 1, 768: 1, 763: 1, 752: 1,
740: 1, 737: 1, 734: 1, 722: 1, 718: 1, 710: 1, 707: 1, 692: 1, 688: 1, 678: 1, 677: 1, 675: 1,
672: 1, 671: 1, 666: 1, 660: 1, 651: 1, 643: 1, 627: 1, 610: 1, 609: 1, 596: 1, 570: 1, 560: 1,
519: 1, 495: 1, 492: 1, 484: 1})

In [53]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3638390054875325
For values of alpha =  0.0001 The log loss is: 1.2355136786733723
For values of alpha =  0.001 The log loss is: 1.2024483687618615
For values of alpha =  0.01 The log loss is: 1.2890752771758438
For values of alpha =  0.1 The log loss is: 1.4521994523788857
For values of alpha =  1 The log loss is: 1.6757024152779634
```






```
For values of best alpha =  0.001 The train log loss is: 0.6589161932390051
For values of best alpha =  0.001 The cross validation log loss is: 1.2024483687618615
For values of best alpha =  0.001 The test log loss is: 1.138737653431301
```

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

Ans. Yes, it seems like!

In [54]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    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 [55]:

```
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")
```

```
97.411 % of word of test data appeared in train data
97.828 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [56]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [57]:

```
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 = 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 [58]:

```
# 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 = CountVectorizer(min_df=3)

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

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

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

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

Stacking the three types of features

In [59]:

```
# merging gene, variance and text features

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

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

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

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

```
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 [60]:

```
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, 56059)
(number of data points * number of features) in test data =  (665, 56059)
(number of data points * number of features) in cross validation data = (532, 56059)
```

In [61]:

```
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

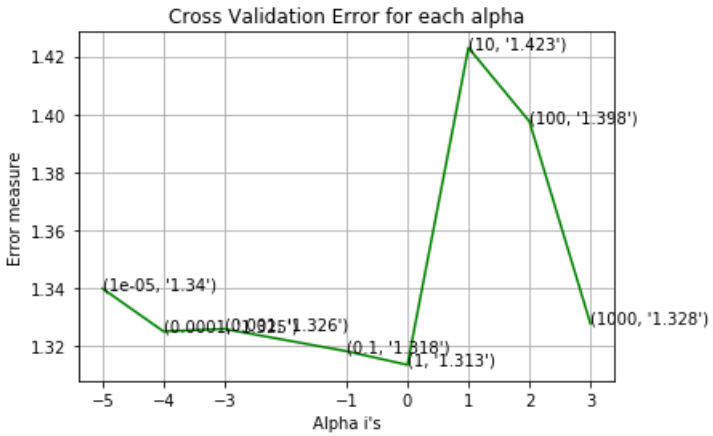
4.1.1.1. Hyper parameter tuning

In [62]:

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

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

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


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.3398244517493882
for alpha = 0.0001
Log Loss : 1.325010897745486
for alpha = 0.001
Log Loss : 1.3258527152817345
for alpha = 0.1
Log Loss : 1.318165381913671
for alpha = 1
Log Loss : 1.3134507139897358
for alpha = 10
Log Loss : 1.4229816460108489
for alpha = 100
Log Loss : 1.39752307518821
for alpha = 1000
Log Loss : 1.3277397032413891
```



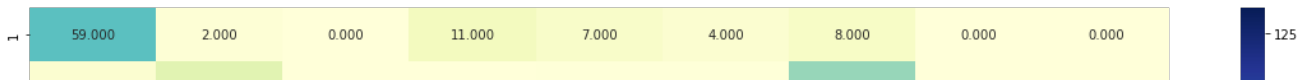

```
For values of best alpha =  1 The train log loss is: 0.8626871163288372
For values of best alpha =  1 The cross validation log loss is: 1.3134507139897358
For values of best alpha =  1 The test log loss is: 1.3057301806029227
```

4.1.1.2. Testing the model with best hyper paramters

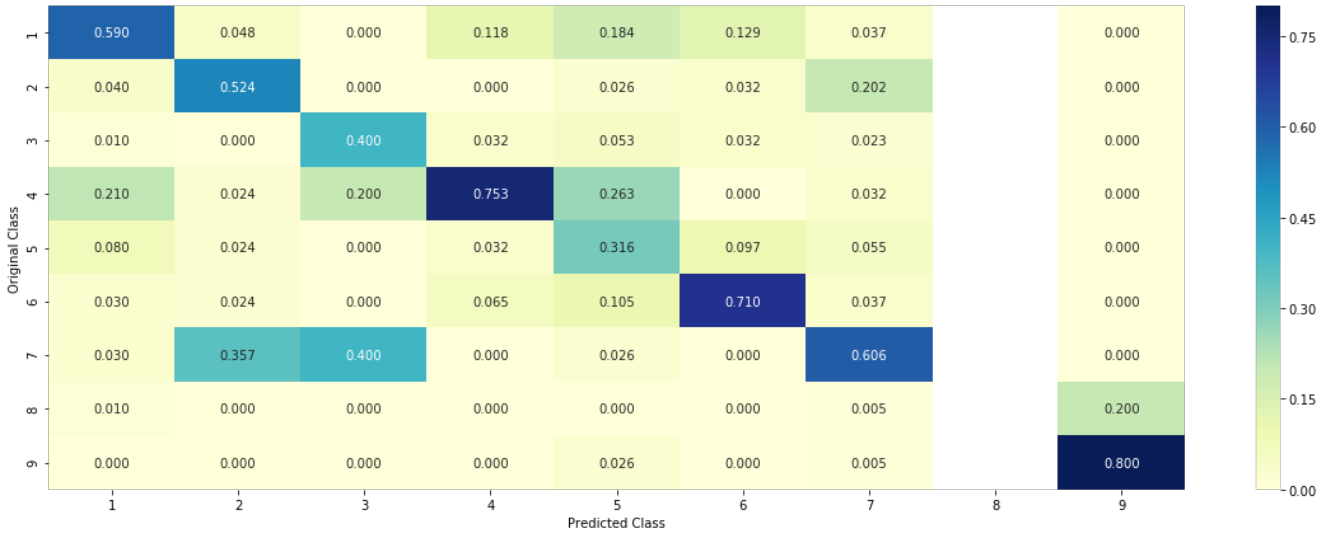
In [63]:

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

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


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

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

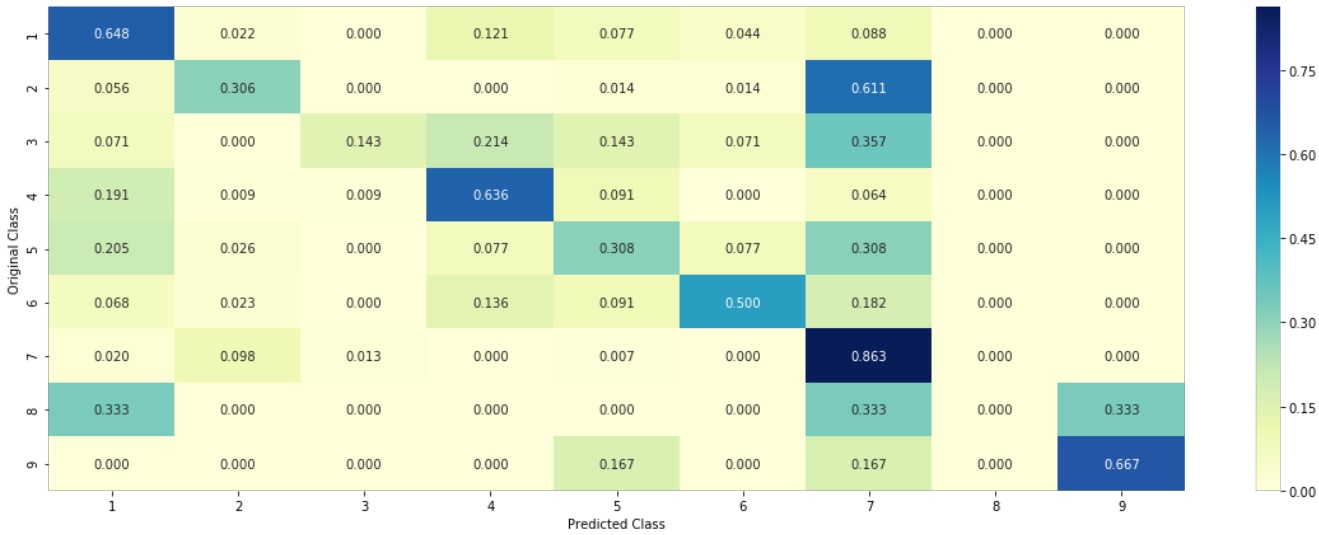
```
Log Loss : 1.3134507139897358
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [64]:

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

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0852 0.0984 0.0273 0.1097 0.5293 0.0502 0.0874 0.0078 0.0048]]
Actual Class : 7
--------------------------------------------------
6 Text feature [assays] present in test data point [True]
8 Text feature [functional] present in test data point [True]
10 Text feature [variants] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [65]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0701 0.0811 0.0227 0.0888 0.1345 0.5213 0.0714 0.0063 0.0038]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

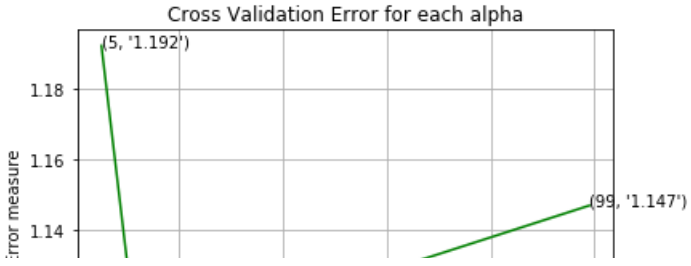
4.2.1. Hyper parameter tuning

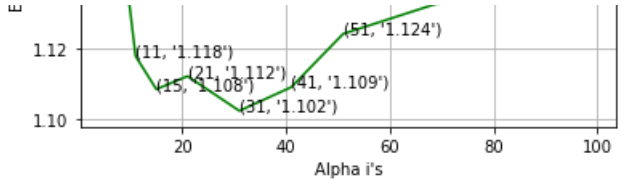
In [66]:

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


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

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


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

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


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

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

```
for alpha = 5
Log Loss : 1.1920097021622382
for alpha = 11
Log Loss : 1.1177539927138633
for alpha = 15
Log Loss : 1.1083857826189762
for alpha = 21
Log Loss : 1.112078238224817
for alpha = 31
Log Loss : 1.10227225719733
for alpha = 41
Log Loss : 1.1089717008687086
for alpha = 51
Log Loss : 1.1240602493529364
for alpha = 99
Log Loss : 1.1467856200518796
```





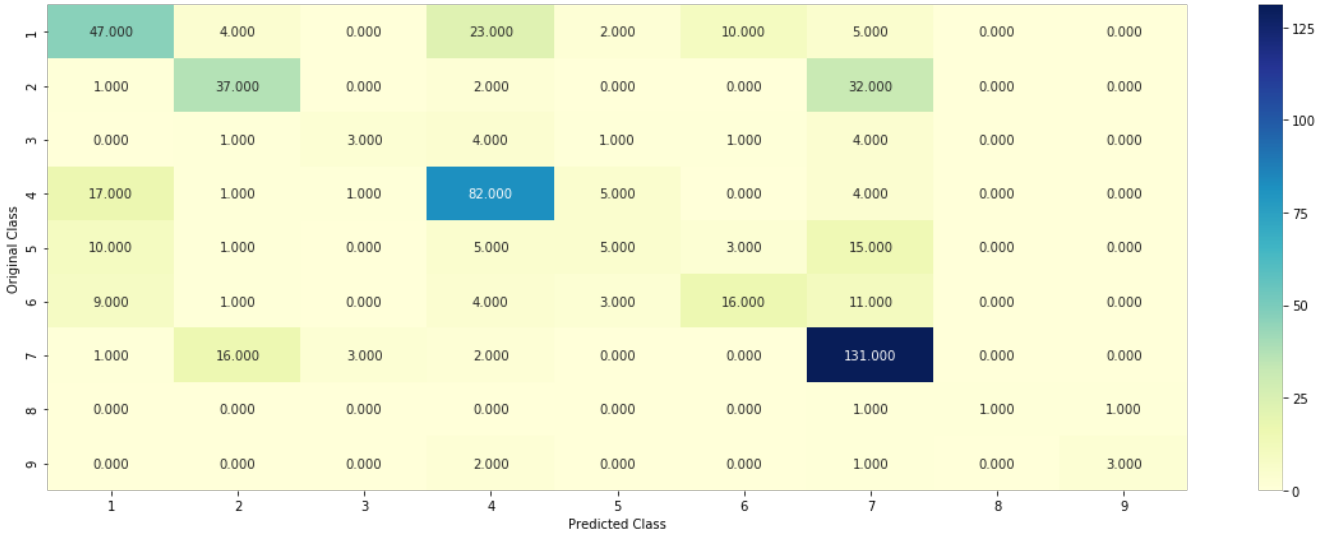

```
For values of best alpha =  31 The train log loss is: 0.7860078119040521
For values of best alpha =  31 The cross validation log loss is: 1.10227225719733
For values of best alpha =  31 The test log loss is: 1.1304945684743055
```

4.2.2. Testing the model with best hyper paramters

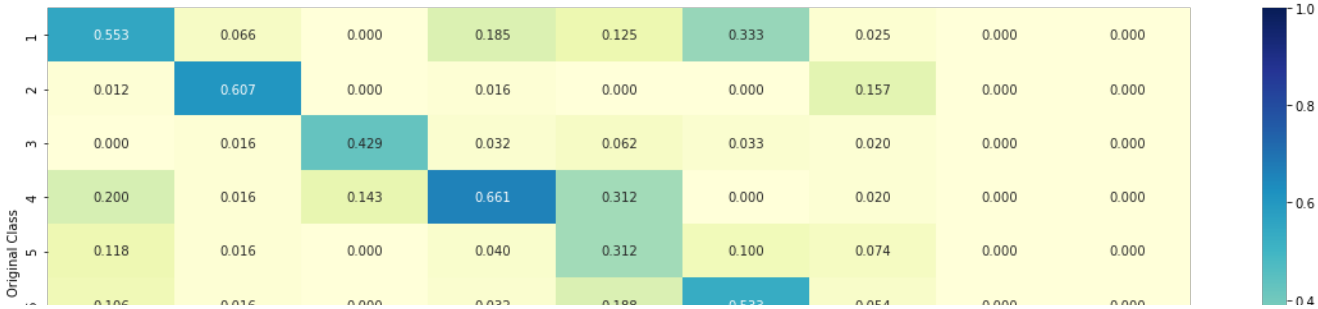
In [67]:

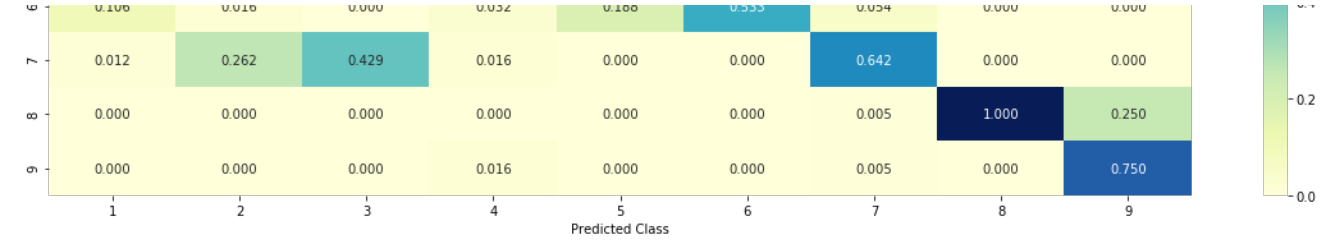
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 1.10227225719733
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```

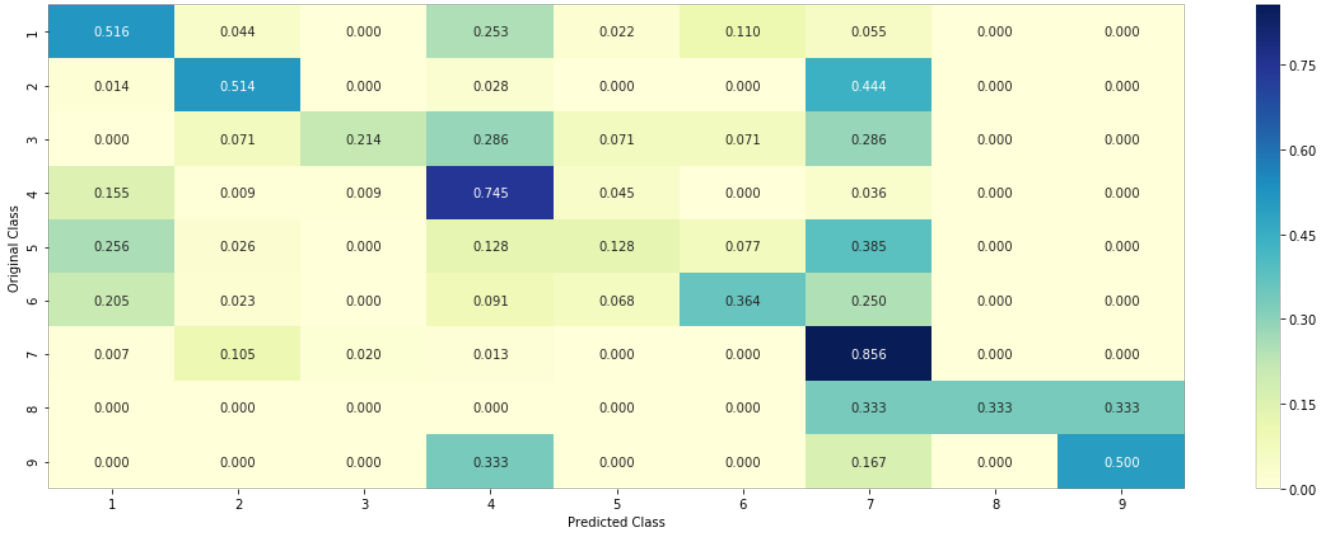



```
-------------------- Precision matrix (Column Sum=1) --------------------
```





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



4.2.3.Sample Query point -1

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

```
Predicted Class : 4
Actual Class : 7
The  31  nearest neighbours of the test points belongs to classes [1 2 5 7 7 5 5 5 5 7 5 7 5 7 7 7
7 7 2 2 2 7 7 5 5 5 7 7 7 7 7]
Fequency of nearest points : Counter({7: 16, 5: 10, 2: 4, 1: 1})
```

4.2.4. Sample Query Point-2

In [69]:

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

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
```

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

```
Predicted Class : 6
Actual Class : 1
the k value for knn is 31 and the nearest neighbours of the test points belongs to classes [5 5 6
6 6 6 6 3 1 3 6 5 4 1 6 6 1 1 4 5 4 4 6 6 5 5 6 5 1 1 1]
Fequency of nearest points : Counter({6: 11, 5: 7, 1: 7, 4: 4, 3: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

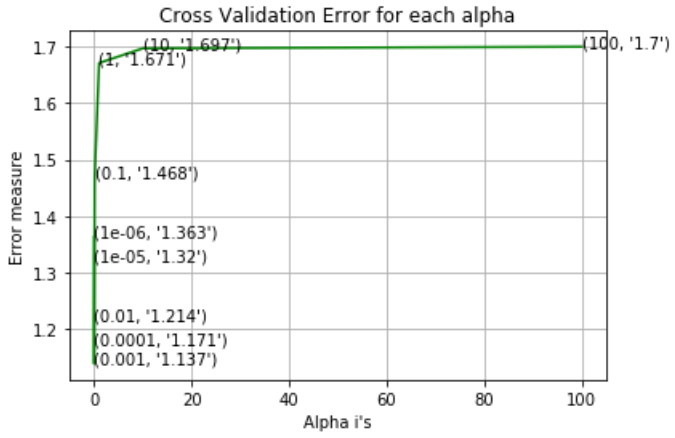
4.3.1.1. Hyper paramter tuning

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

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

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


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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

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


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

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

```
for alpha = 1e-06
Log Loss : 1.362760932088151
for alpha = 1e-05
Log Loss : 1.3201457956536469
for alpha = 0.0001
Log Loss : 1.171102284206199
for alpha = 0.001
Log Loss : 1.1374938663568557
for alpha = 0.01
Log Loss : 1.2143848313959358
for alpha = 0.1
Log Loss : 1.4678245193129924
for alpha = 1
Log Loss : 1.671292635607499
for alpha = 10
Log Loss : 1.6972664053392836
for alpha = 100
Log Loss : 1.7001266529817456
```




```
For values of best alpha =  0.001 The train log loss is: 0.5181283959464157
For values of best alpha =  0.001 The cross validation log loss is: 1.1374938663568557
For values of best alpha =  0.001 The test log loss is: 1.0885605715868265
```

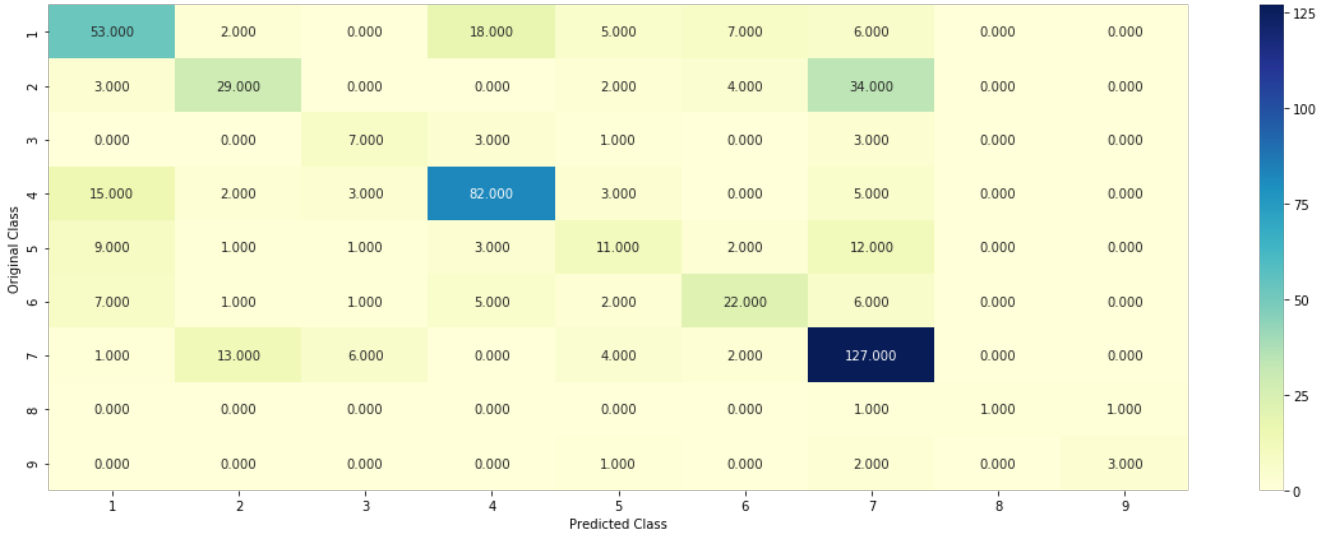
4.3.1.2. Testing the model with best hyper paramters

In [71]:

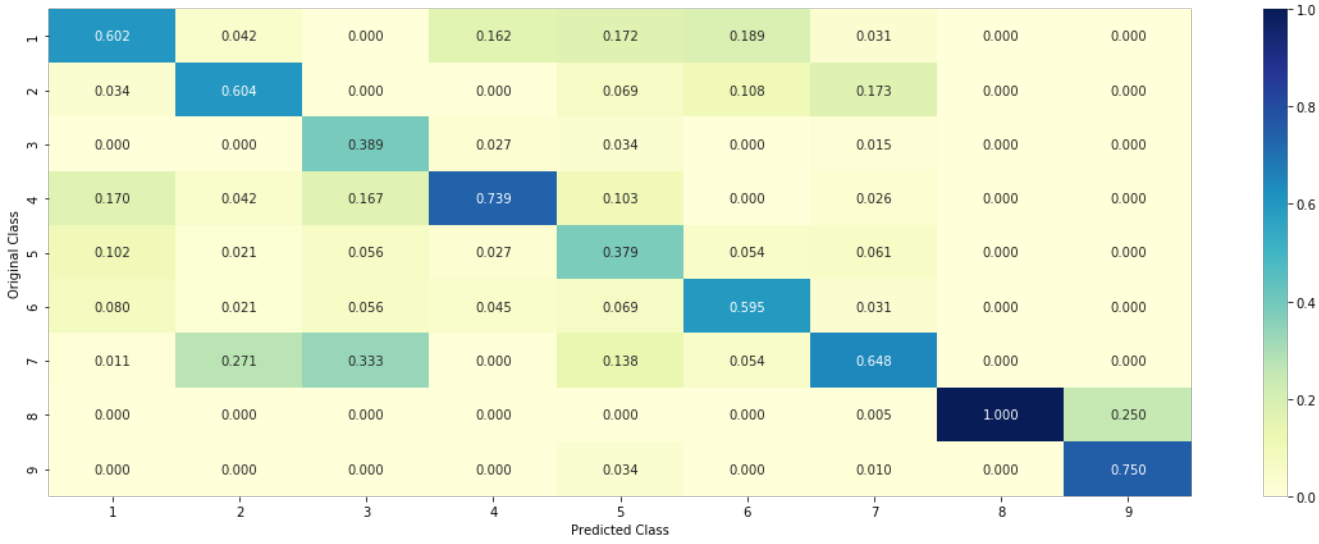
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
```

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

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

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

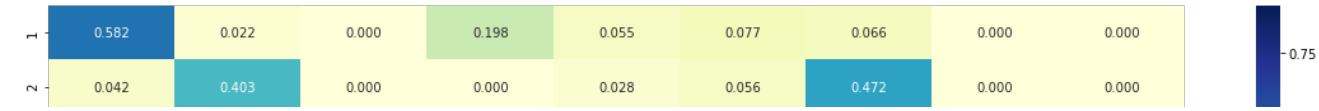
```
Log loss : 1.1374938663568557
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```

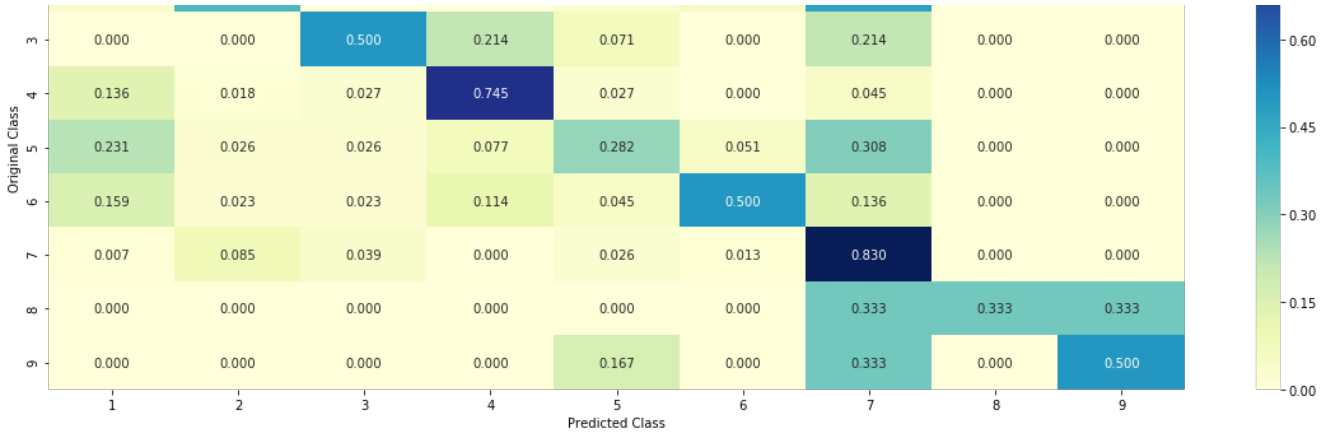



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.1.3. Feature Importance

In [72]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [73]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0541 0.0857 0.004  0.0102 0.6521 0.0092 0.1757 0.0066 0.0025]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [74]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[3.100e-03 4.600e-03 1.100e-03 2.300e-03 2.494e-01 7.326e-01 2.600
e-03
  3.700e-03 6.000e-04]]
Actual Class : 1
--------------------------------------------------
151 Text feature [inhibitors] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2. Without Class balancing

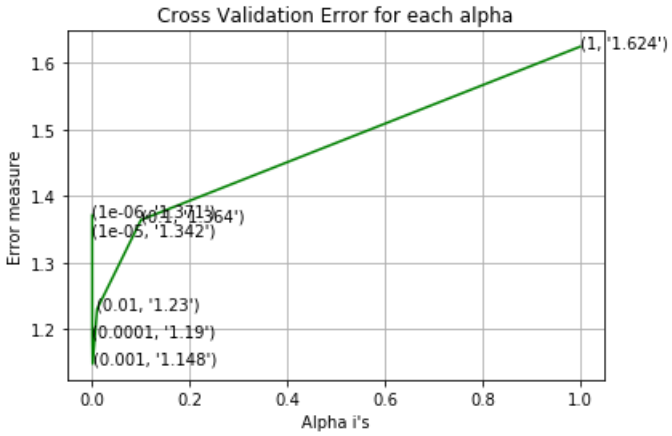
4.3.2.1. Hyper paramter tuning

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



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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
```

```
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.3709809237670505
for alpha = 1e-05
Log Loss : 1.3416172607453885
for alpha = 0.0001
Log Loss : 1.1895202328740493
for alpha = 0.001
Log Loss : 1.1477863479359725
for alpha = 0.01
Log Loss : 1.2301192915635595
for alpha = 0.1
Log Loss : 1.3640768701839239
for alpha = 1
Log Loss : 1.6242772832070649
```




```
For values of best alpha =  0.001 The train log loss is: 0.5200620416971534
For values of best alpha =  0.001 The cross validation log loss is: 1.1477863479359725
For values of best alpha =  0.001 The test log loss is: 1.0892188524090243
```

4.3.2.2. Testing model with best hyper parameters

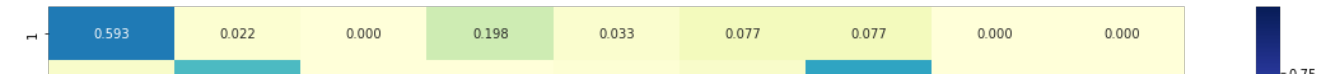
In [76]:

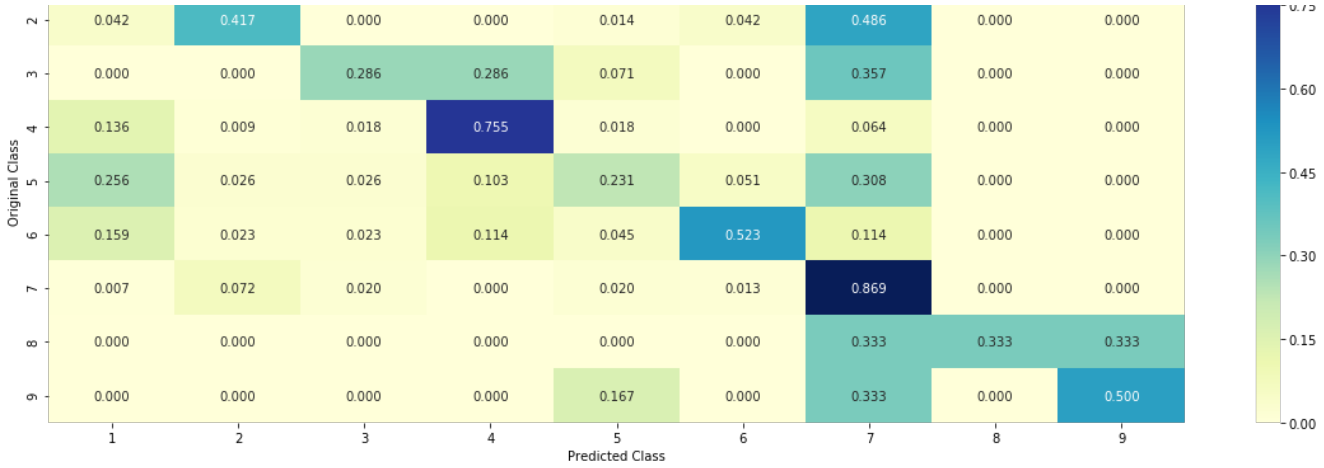
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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 : 1.1477863479359725
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```


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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0687 0.0888 0.0034 0.0094 0.5334 0.0081 0.2815 0.0058 0.0012]]
Actual Class : 7
--------------------------------------------------
40 Text feature [assume] present in test data point [True]
Out of the top  500  features  1 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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[3.800e-03 5.100e-03 1.000e-03 2.900e-03 2.168e-01 7.626e-01 3.700
e-03
  3.800e-03 4.000e-04]]
Actual Class : 1
--------------------------------------------------
148 Text feature [ms] present in test data point [True]
213 Text feature [melanoma] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.4. Linear Support Vector Machines

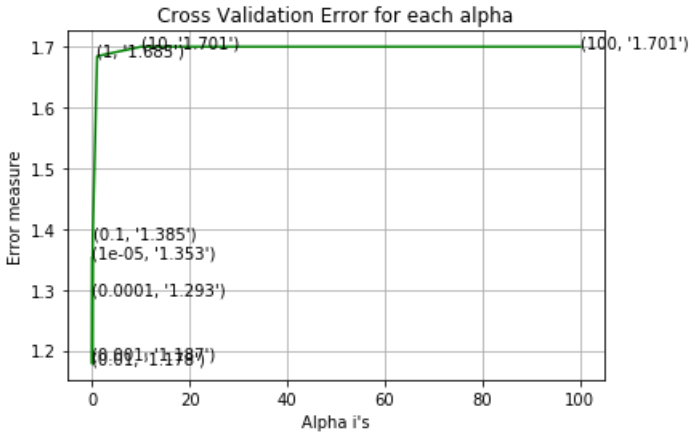
4.4.1. Hyper paramter tuning

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

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



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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.3529830105915466
for C = 0.0001
Log Loss : 1.2931131596182297
for C = 0.001
Log Loss : 1.186890467433792
for C = 0.01
Log Loss : 1.1781794001059505
for C = 0.1
Log Loss : 1.3854494874531171
for C = 1
Log Loss : 1.6850058809388804
for C = 10
Log Loss : 1.7006012980728027
for C = 100
Log Loss : 1.7005997960013555
```




```
For values of best alpha =  0.01 The train log loss is: 0.7353906623280283
For values of best alpha =  0.01 The cross validation log loss is: 1.1781794001059505
For values of best alpha =  0.01 The test log loss is: 1.1544164462075237
```

4.4.2. Testing model with best hyper parameters

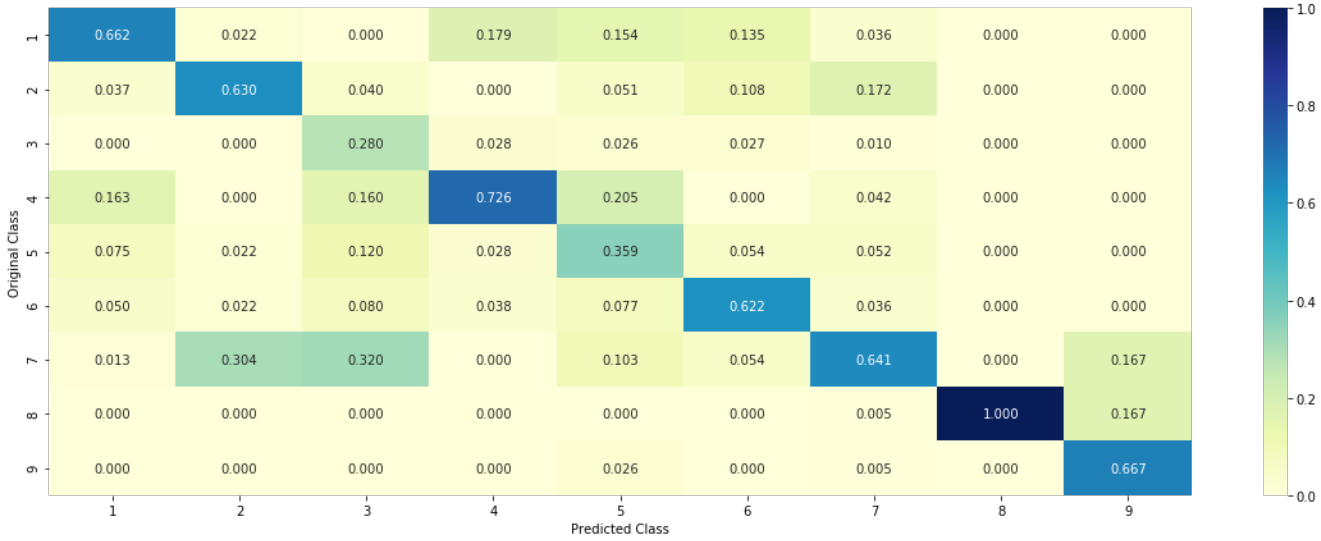
In [80]:

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

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

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


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

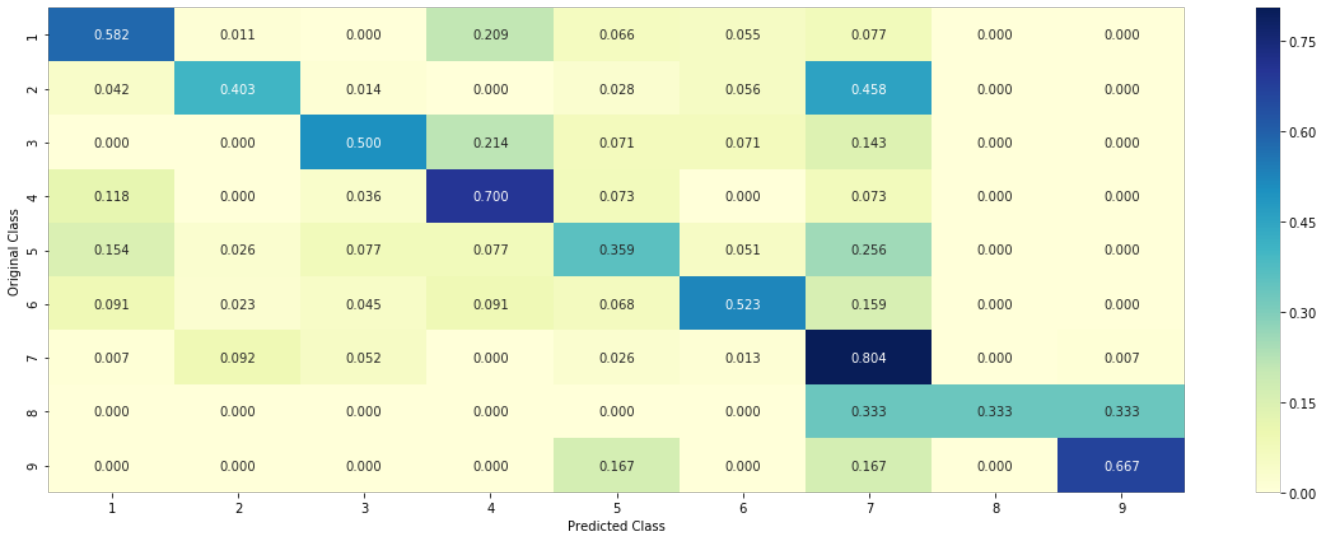
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1781794001059505
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```


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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [81]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.063  0.0634 0.0129 0.0637 0.6926 0.026  0.0683 0.0062 0.0038]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [82]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0337 0.0288 0.0067 0.0303 0.1174 0.7446 0.0327 0.0037 0.002 ]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [83]:

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

```
# class_weight=None)

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

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

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


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

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

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

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

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

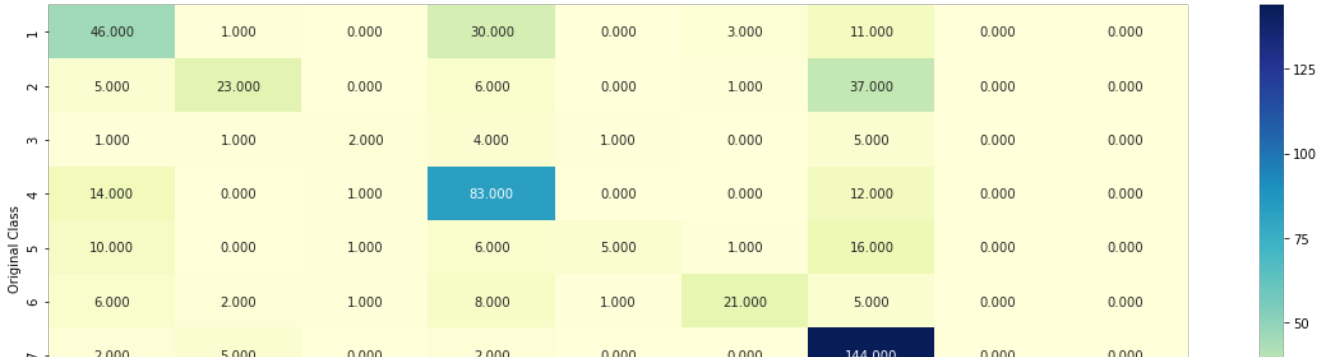
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2812774130289128
for n_estimators = 100 and max depth =  10
Log Loss : 1.1984461284310193
for n_estimators = 200 and max depth =  5
Log Loss : 1.2648573362268318
for n_estimators = 200 and max depth =  10
Log Loss : 1.1837770722033625
for n_estimators = 500 and max depth =  5
Log Loss : 1.2572637393129547
for n_estimators = 500 and max depth =  10
Log Loss : 1.1812811326254076
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2525407430375062
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1814146398819152
for n_estimators = 2000 and max depth =  5
Log Loss : 1.253227702693939
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1786418436032602
For values of best estimator =  2000 The train log loss is: 0.6808470414911972
For values of best estimator =  2000 The cross validation log loss is: 1.1786418436032602
For values of best estimator =  2000 The test log loss is: 1.1296569623748673
```

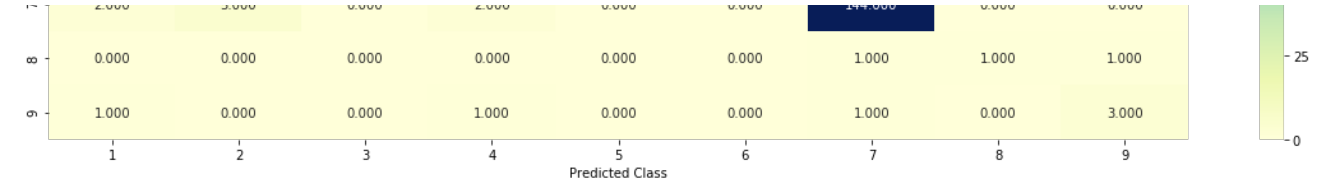
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [84]:

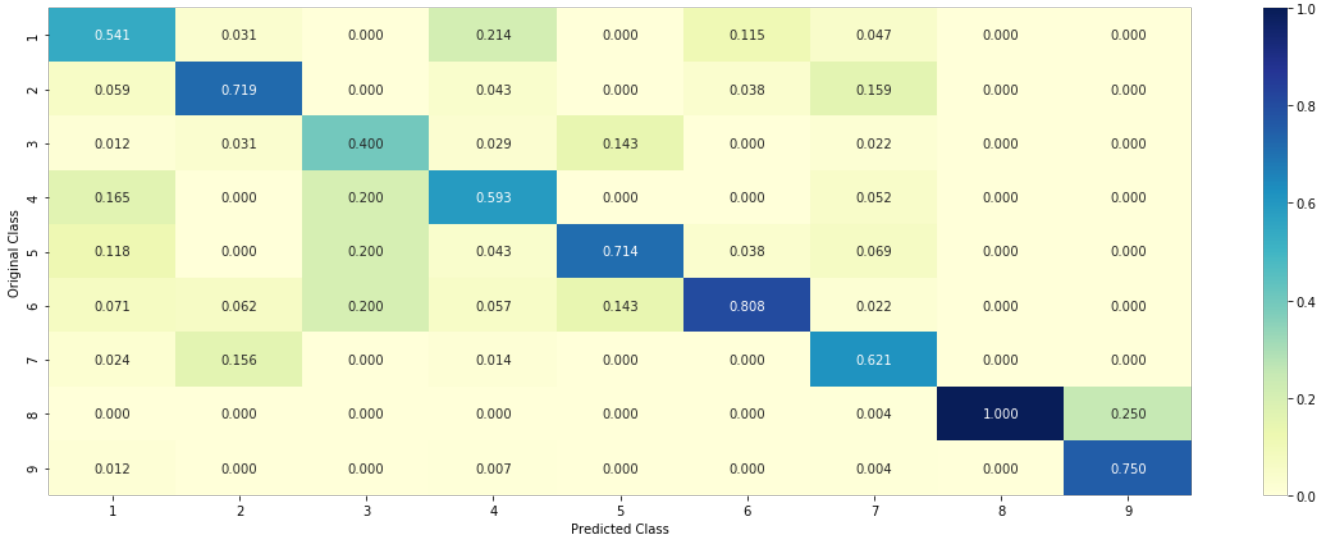
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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 : 1.1786418436032602
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```

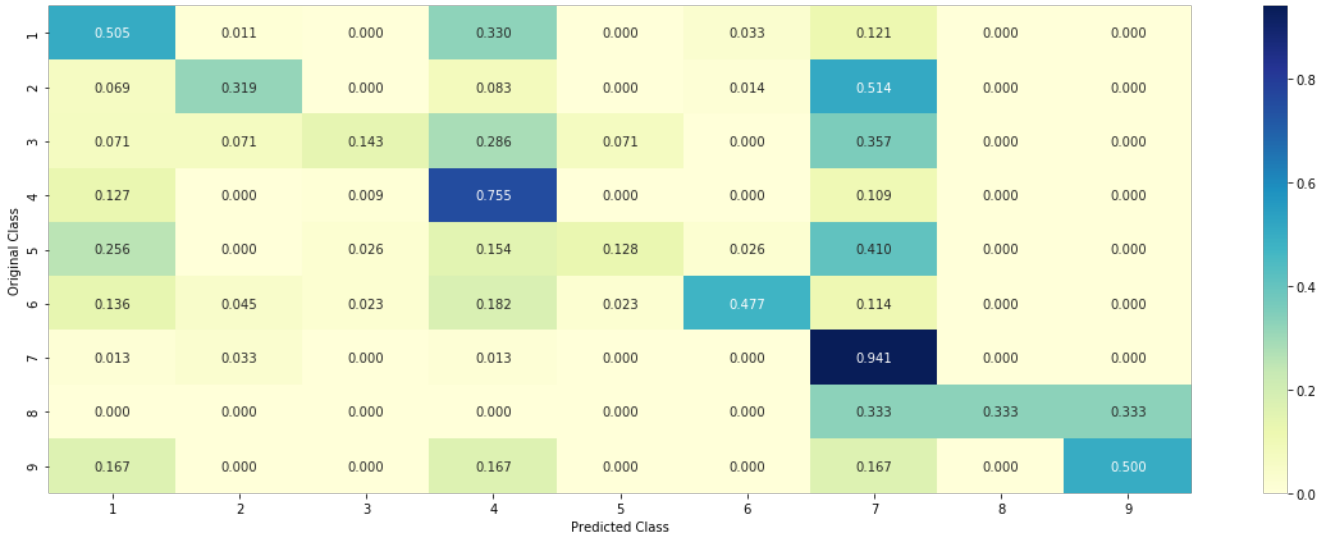




---------- Precision matrix (Columm Sum=1) ----------



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [85]:

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

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

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1144 0.1495 0.0255 0.0843 0.1499 0.0589 0.4011 0.0076 0.0088]]
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 [tyrosine] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [inhibitors] present in test data point [True]
7 Text feature [missense] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
9 Text feature [function] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [therapy] present in test data point [True]
14 Text feature [signaling] present in test data point [True]
15 Text feature [loss] present in test data point [True]
16 Text feature [treatment] present in test data point [True]
17 Text feature [oncogenic] present in test data point [True]
18 Text feature [receptor] present in test data point [True]
20 Text feature [therapeutic] present in test data point [True]
23 Text feature [growth] present in test data point [True]
24 Text feature [cells] present in test data point [True]
25 Text feature [downstream] present in test data point [True]
26 Text feature [resistance] present in test data point [True]
28 Text feature [pathogenic] present in test data point [True]
30 Text feature [variants] present in test data point [True]
31 Text feature [extracellular] present in test data point [True]
33 Text feature [functional] present in test data point [True]
34 Text feature [transforming] present in test data point [True]
37 Text feature [stability] present in test data point [True]
39 Text feature [kinases] present in test data point [True]
44 Text feature [cell] present in test data point [True]
45 Text feature [drug] present in test data point [True]
49 Text feature [expressing] present in test data point [True]
52 Text feature [proliferation] present in test data point [True]
56 Text feature [trials] present in test data point [True]
57 Text feature [expression] present in test data point [True]
59 Text feature [protein] present in test data point [True]
61 Text feature [phospho] present in test data point [True]
66 Text feature [efficacy] present in test data point [True]
70 Text feature [advanced] present in test data point [True]
76 Text feature [clinical] present in test data point [True]
78 Text feature [tkis] present in test data point [True]
80 Text feature [enhanced] present in test data point [True]
90 Text feature [variant] present in test data point [True]
91 Text feature [amplification] present in test data point [True]
93 Text feature [response] present in test data point [True]
98 Text feature [receptors] present in test data point [True]
Out of the top  100  features  44 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [86]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2089 0.0487 0.0227 0.2025 0.0996 0.3394 0.0562 0.0115 0.0106]]
Actuall Class : 1
--------------------------------------------------
4 Text feature [inhibitors] present in test data point [True]
9 Text feature [function] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
15 Text feature [loss] present in test data point [True]
20 Text feature [therapeutic] present in test data point [True]
24 Text feature [cells] present in test data point [True]
29 Text feature [patients] present in test data point [True]
33 Text feature [functional] present in test data point [True]
41 Text feature [inhibition] present in test data point [True]
44 Text feature [cell] present in test data point [True]
57 Text feature [expression] present in test data point [True]
59 Text feature [protein] present in test data point [True]
89 Text feature [repair] present in test data point [True]
Out of the top  100  features  13 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [87]:

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

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

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

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


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

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

```
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max
_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 : 2.0973300879233676
for n_estimators = 10 and max depth =  3
Log Loss : 1.7847818038274799
for n_estimators = 10 and max depth =  5
Log Loss : 1.499264842049269
for n_estimators = 10 and max depth =  10
Log Loss : 2.0771523667487166
for n_estimators = 50 and max depth =  2
Log Loss : 1.82722332662612
for n_estimators = 50 and max depth =  3
Log Loss : 1.5837093806655085
for n_estimators = 50 and max depth =  5
Log Loss : 1.4747077889194722
for n_estimators = 50 and max depth =  10
Log Loss : 1.7110763360160897
for n_estimators = 100 and max depth =  2
Log Loss : 1.692436656594775
for n_estimators = 100 and max depth =  3
Log Loss : 1.580958009506553
for n_estimators = 100 and max depth =  5
Log Loss : 1.3850274682115042
for n_estimators = 100 and max depth =  10
Log Loss : 1.713646818359853
for n_estimators = 200 and max depth =  2
Log Loss : 1.7466174048799914
for n_estimators = 200 and max depth =  3
Log Loss : 1.6176975334056443
for n_estimators = 200 and max depth =  5
Log Loss : 1.435046847294524
for n_estimators = 200 and max depth =  10
Log Loss : 1.73285878188717
for n_estimators = 500 and max depth =  2
Log Loss : 1.7867776395118276
for n_estimators = 500 and max depth =  3
Log Loss : 1.6606215534288697
for n_estimators = 500 and max depth =  5
Log Loss : 1.493654923722769
for n_estimators = 500 and max depth =  10
Log Loss : 1.7776895187417205
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7793009933631463
for n_estimators = 1000 and max depth =  3
```

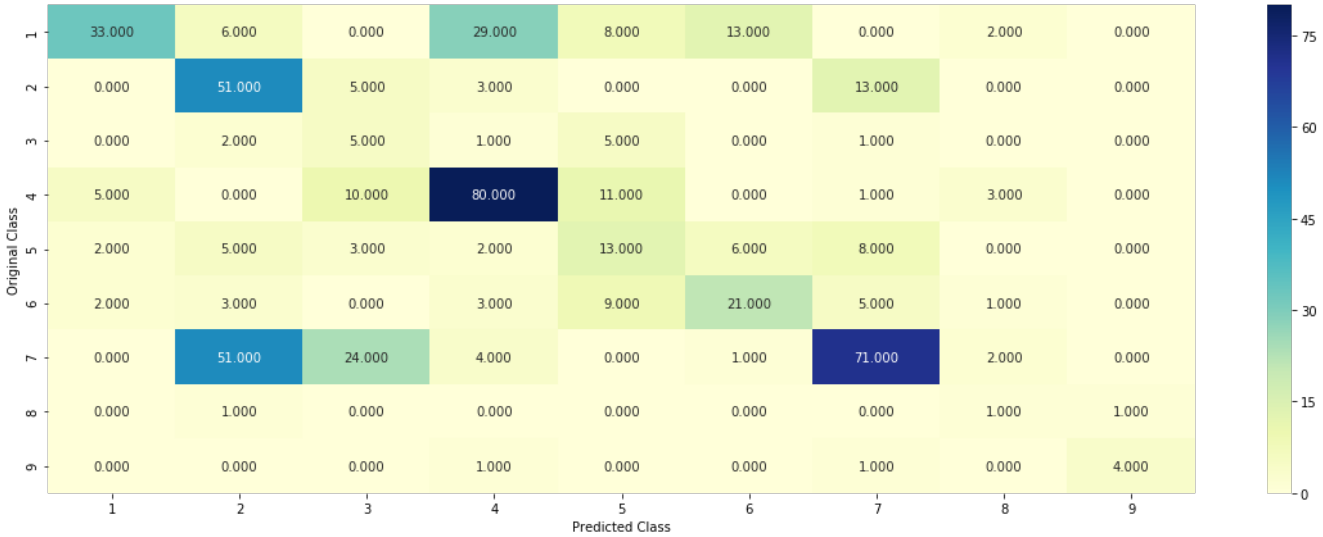
```
Log Loss : 1.6797121177566368
for n_estimators = 1000 and max depth =  5
Log Loss : 1.481285106143352
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7761661313271
For values of best alpha =  100 The train log loss is: 0.061568829146774466
For values of best alpha =  100 The cross validation log loss is: 1.3850274682115042
For values of best alpha =  100 The test log loss is: 1.2989191727436729
```

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

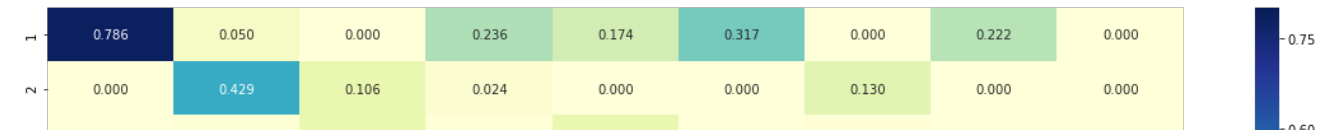
In [88]:

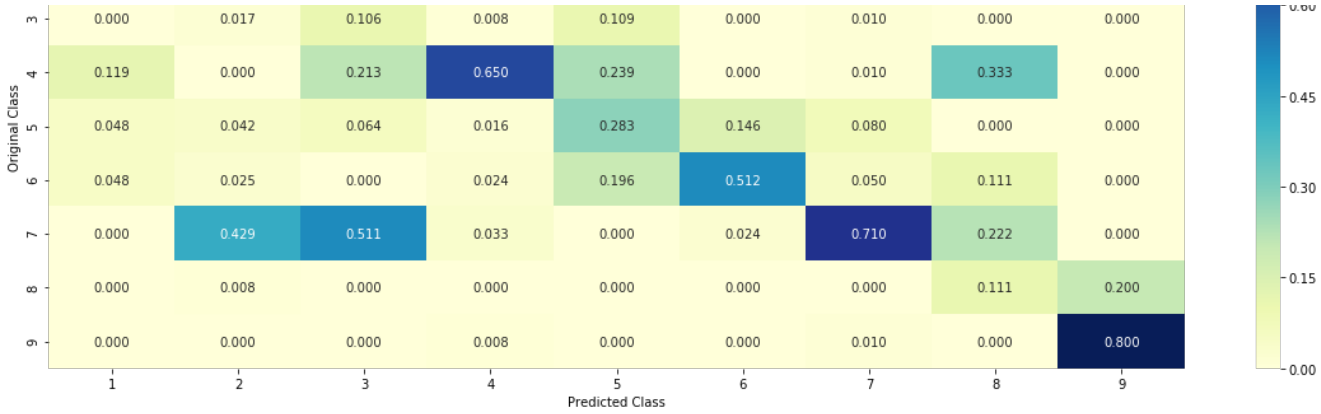
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, 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/
# -------------------------------

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

```
Log loss : 1.3850274682115042
Number of mis-classified points : 0.4755639097744361
-------------------- Confusion matrix --------------------
```

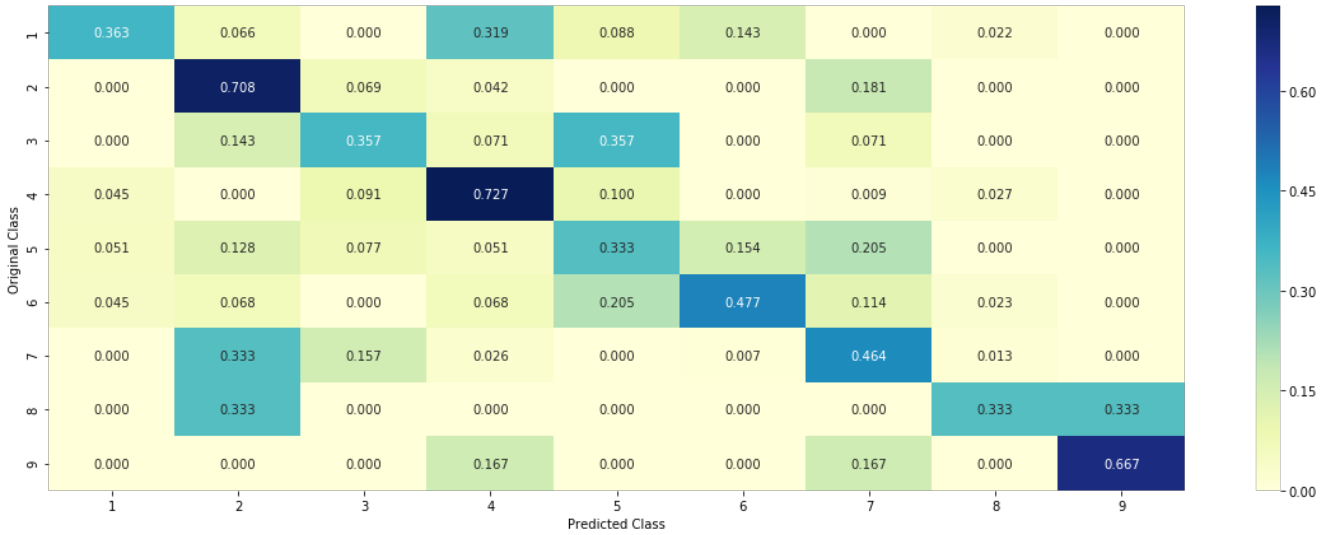



```
-------------------- Precision matrix (Column Sum=1) --------------------
```





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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [89]:

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


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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0185 0.2277 0.0964 0.0337 0.1426 0.0809 0.3459 0.0349 0.0193]]
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
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [90]:

```
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 : 6
Predicted Class Probabilities: [[0.0187 0.0091 0.0346 0.0256 0.1705 0.6626 0.0074 0.0376 0.0339]]
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
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [91]:

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

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

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


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

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


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

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

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

# ---------------------------------
```

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

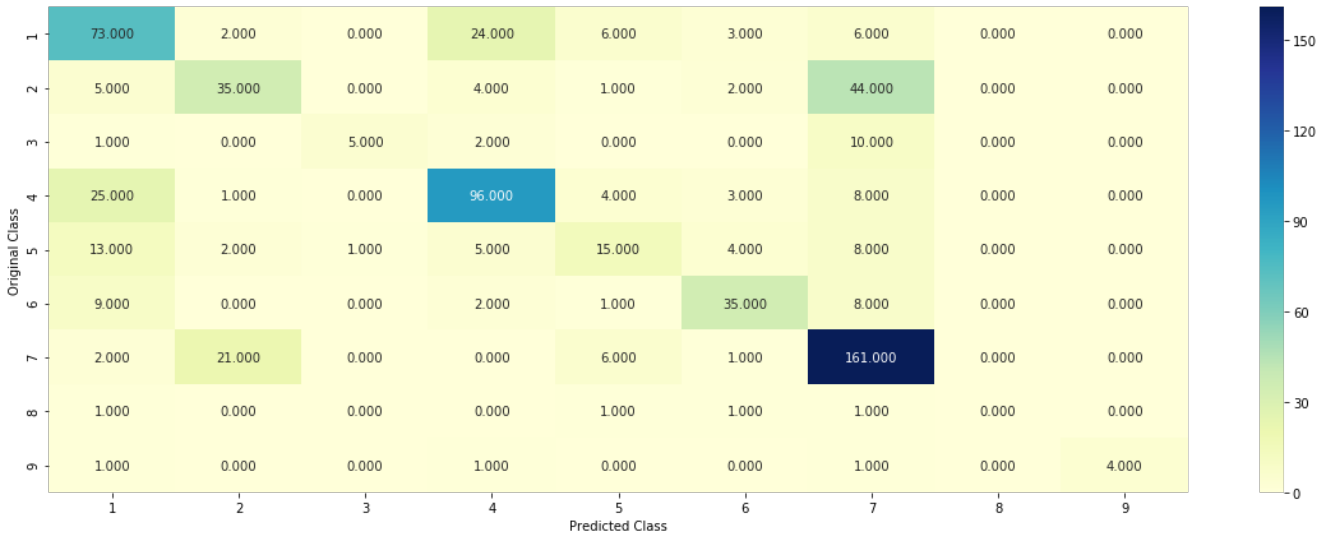
```
Logistic Regression :  Log Loss: 1.14
Support vector machines : Log Loss: 1.69
Naive Bayes : Log Loss: 1.33
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.819
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.727
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.351
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.216
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.495
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.835
```

4.7.2 testing the model with the best hyper parameters

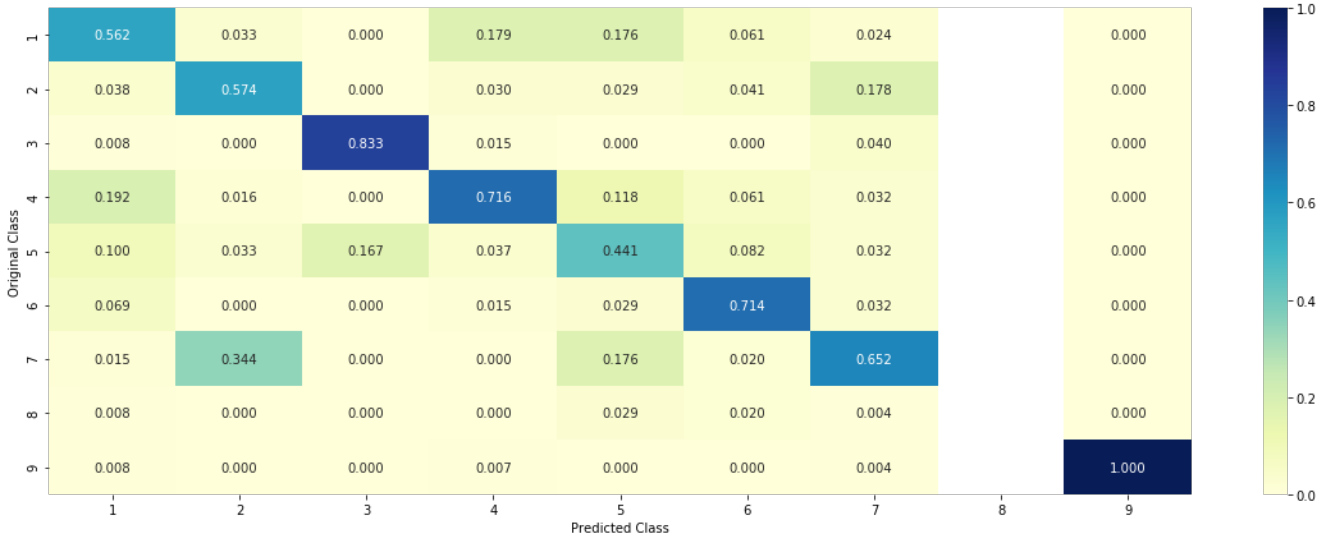
In [92]:

```
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))
```

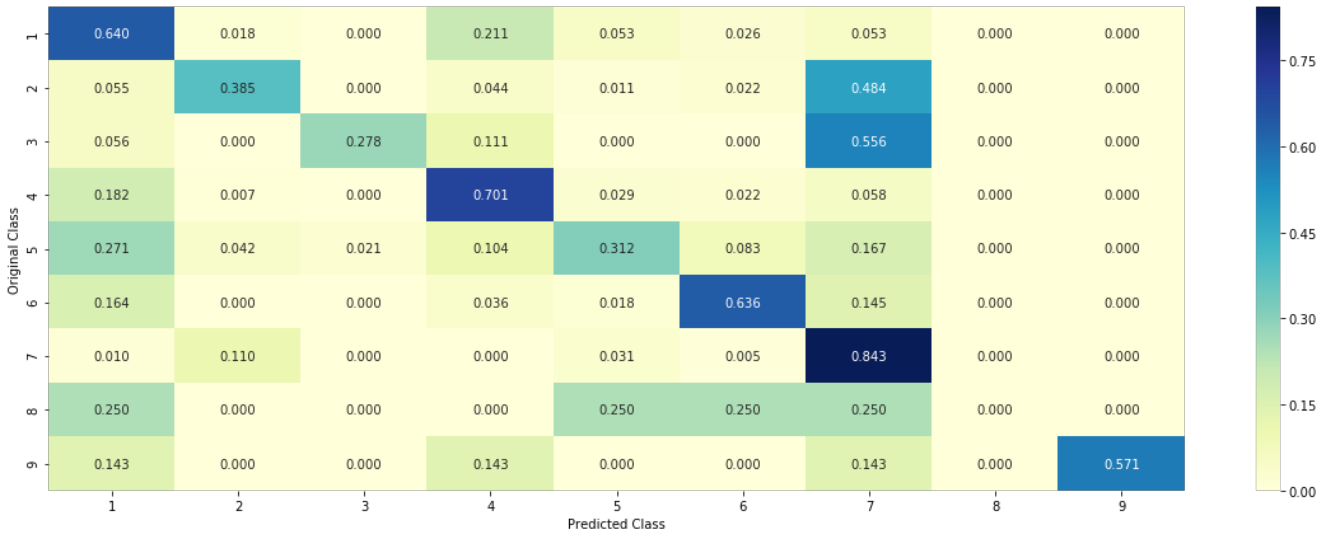
```
Log loss (train) on the stacking classifier : 0.4868024601891316
Log loss (CV) on the stacking classifier : 1.2156110188002596
Log loss (test) on the stacking classifier : 1.20906287859514
Number of missclassified point : 0.362406015037594
-------------------- Confusion matrix --------------------
```




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




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

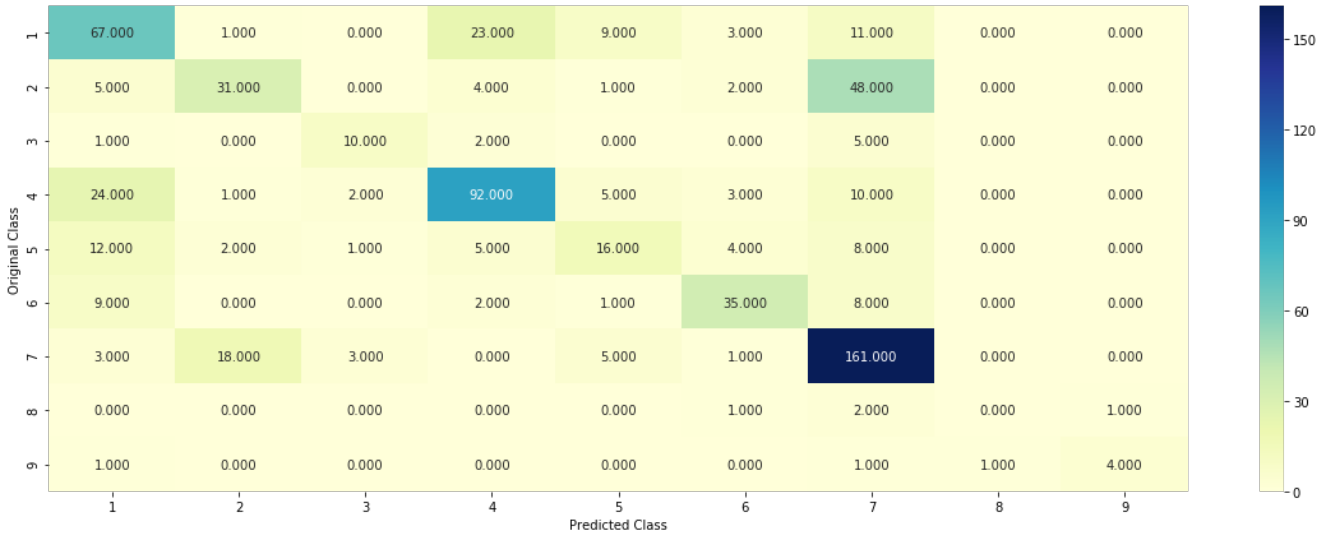


4.7.3 Maximum Voting classifier

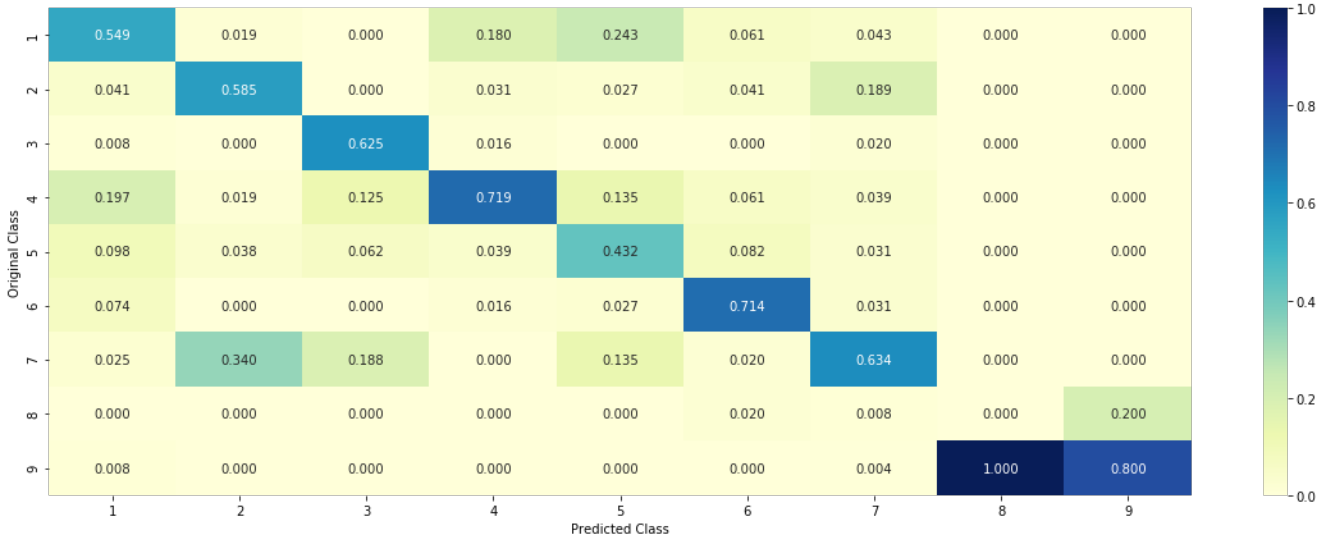
In [93]:

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

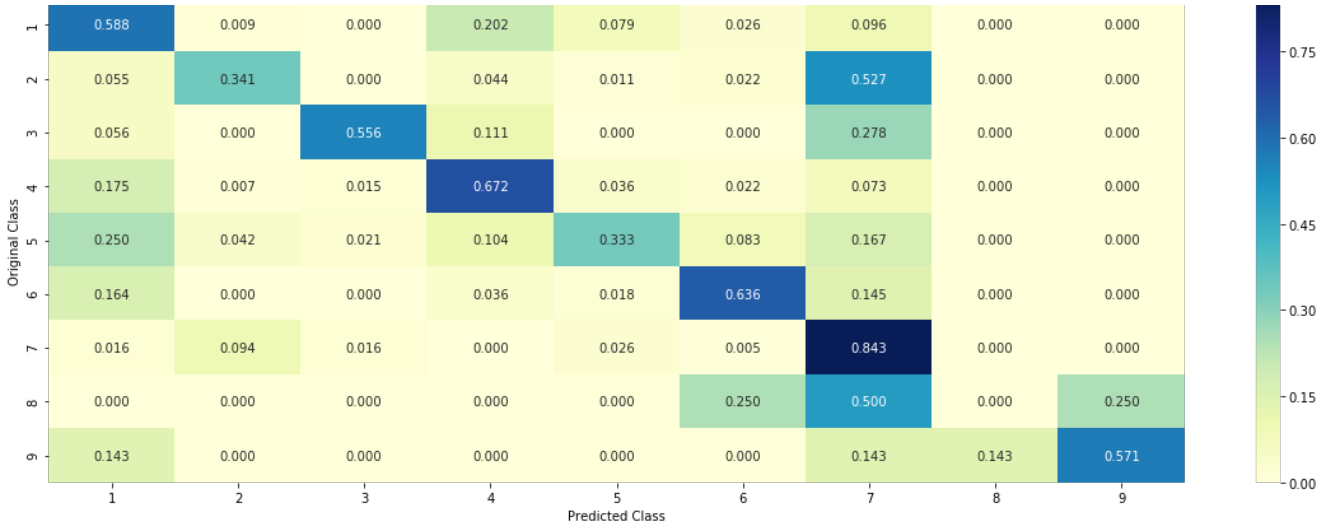
```
Log loss (train) on the VotingClassifier : 0.8535140149069098
Log loss (CV) on the VotingClassifier : 1.2301597924314016
Log loss (test) on the VotingClassifier : 1.193415715379608
Number of missclassified point : 0.3744360902255639
-------------------- Confusion matrix --------------------
```




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




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



5.</xh1>

1: Replacing CountVectorizer with TfidfVectorizer
2: Using top 1000 words based of tf-idf values

5.1. Reading Data

5.1.1. Reading Gene and Variation Data

In [94]:

```
data = pd.read_csv('training/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[94]:

	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

5.1.2. Reading Text Data

In [95]:

```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skip
rows=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[95]:

	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...

5.1.3. Preprocessing of text

In [96]:

```
# 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 [97]:

```
#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 : 101.322794391773408 seconds
```

In [98]:

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

Out[98]:

	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 [99]:

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

Out[99]:

	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 [100]:

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

In [101]:

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

Out[101]:

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

5.1.4. Test, Train and Cross Validation Split

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

In [102]:

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

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

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

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

In [103]:

```
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
```

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

In [104]:

```
import pandas as pd
pd.__version__
```

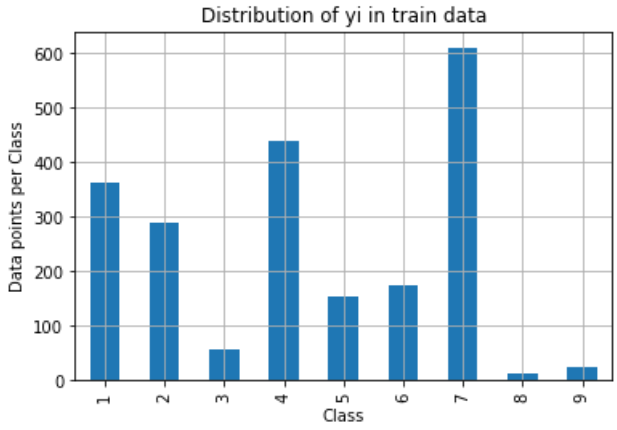
Out[104]:

```
'0.25.3'
```

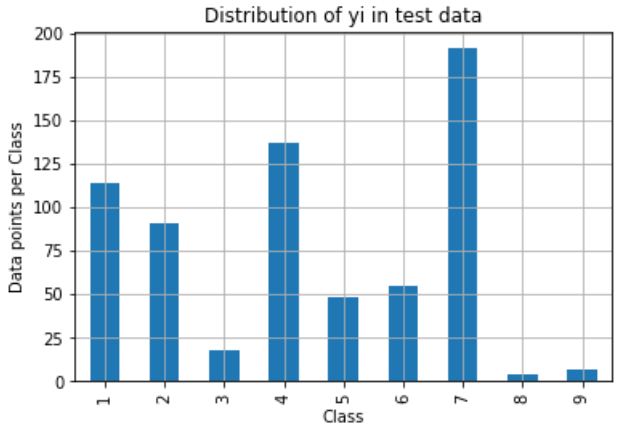
In [105]:

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

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

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


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

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

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
```

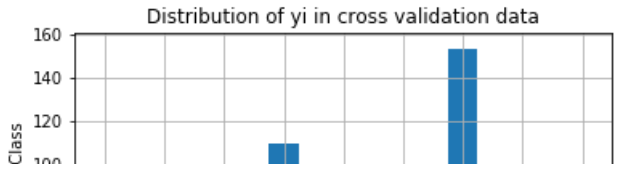
```
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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

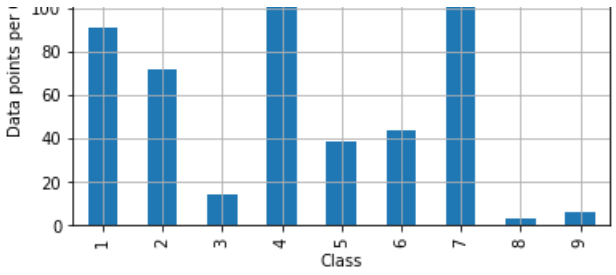



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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




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






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

5.2 Prediction using a 'Random' Model

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

In [106]:

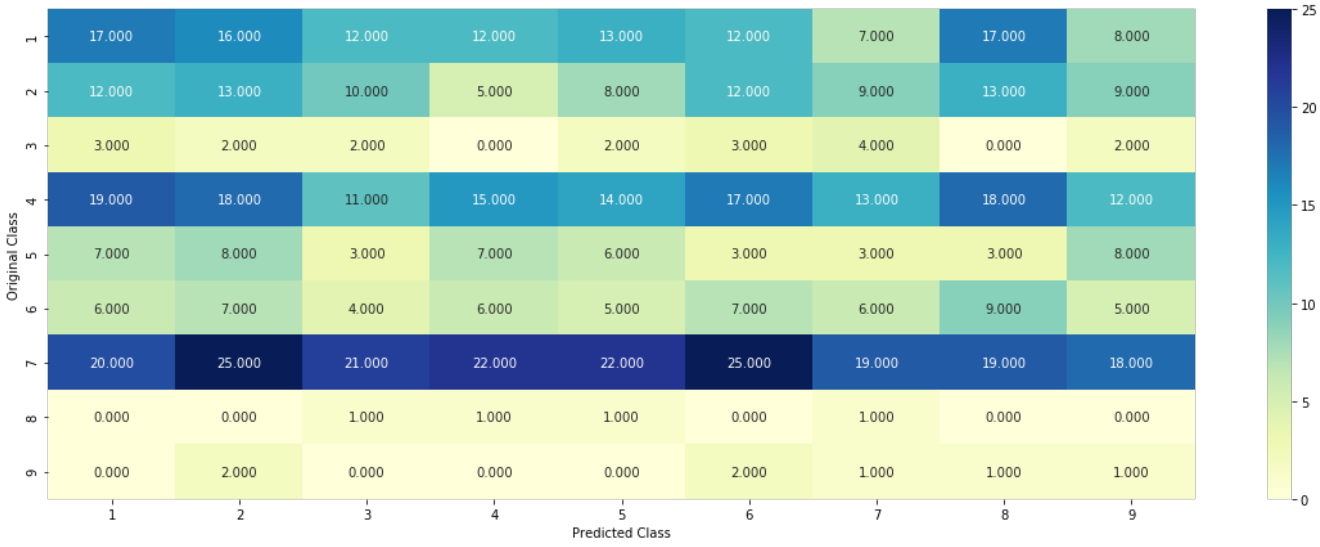
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

```
    plt.show()

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

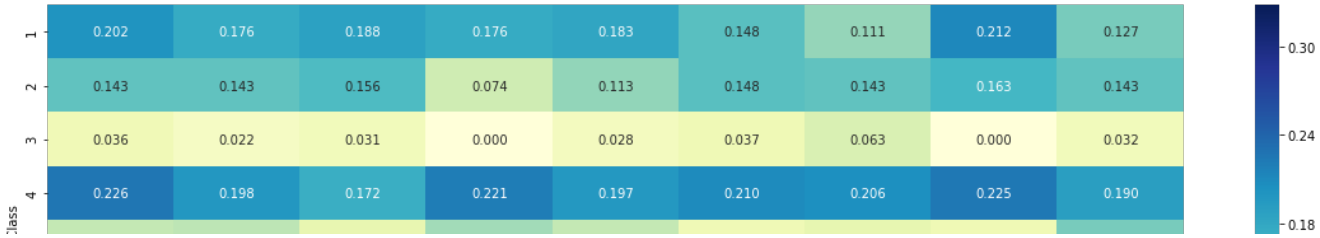
In [107]:

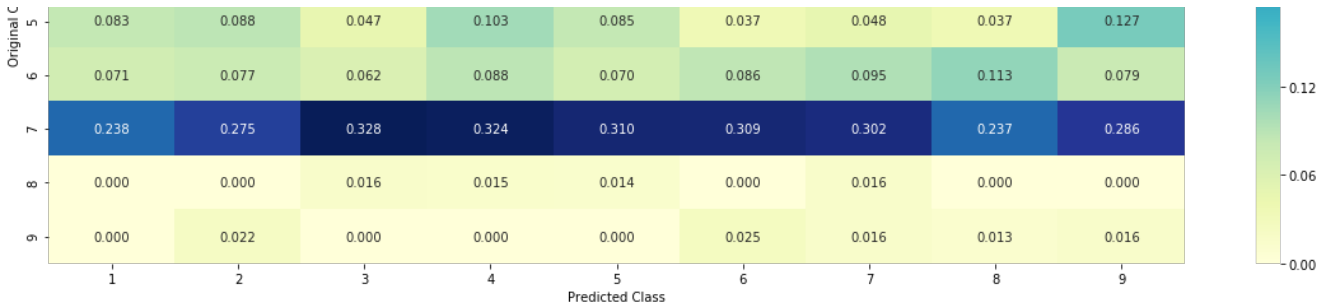
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

```
Log loss on Cross Validation Data using Random Model 2.4812578087403776
Log loss on Test Data using Random Model 2.436035428729115
-------------------- Confusion matrix --------------------
```

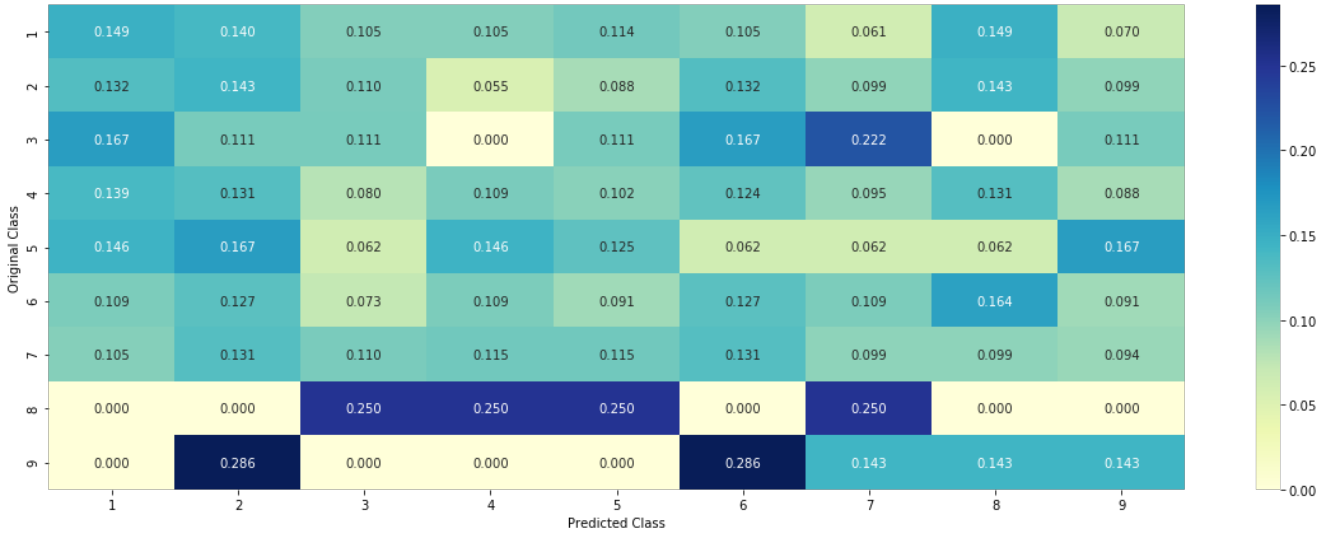



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



5.3 Univariate Analysis

In [108]:

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

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
```

```
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #          ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

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

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the da
ta
    gv_fea = []
```

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

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

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

5.3.1 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 [109]:

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

```
Number of Unique Genes : 239
BRCA1      163
TP53       104
EGFR        96
PTEN        83
BRCA2       78
KIT         68
BRAF        62
ERBB2       49
ALK         43
PDGFRA      39
Name: Gene, dtype: int64
```

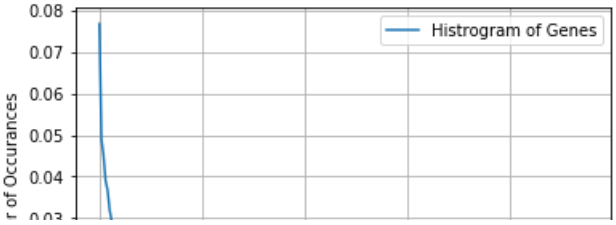
In [110]:

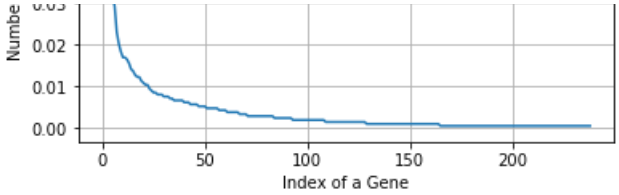
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
d they are distibuted as follows",)
```

```
Ans: There are 239 different categories of genes in the train data, and they are distibuted as fol
lows
```

In [111]:

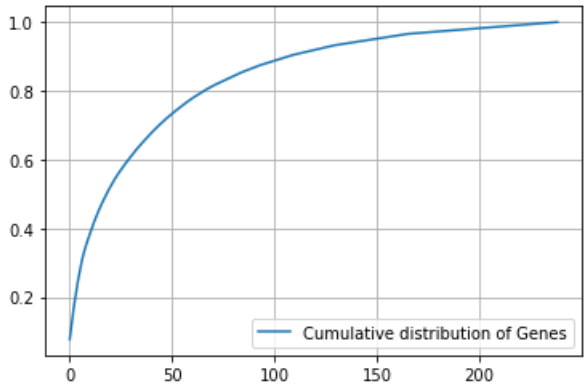
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```





In [112]:

```
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 [113]:

```
#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 [114]:

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

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

In [115]:

```
# 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'])
```

```
cv_gene_feature_onehotcoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [116]:

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

Out[116]:

```
571     SMAD3
2731     BRAF
1727      APC
2498    BRCA1
1448     SPOP
Name: Gene, dtype: object
```

In [117]:

```
gene_vectorizer.get_feature_names()
```

Out[117]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
 'bap1',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
```

```
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm4',
```

```
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'runx1',
'rxra',
'rybp',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'shq1',
'smad2',
'smad3',
'smad4',
```

```
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [118]:

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

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

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

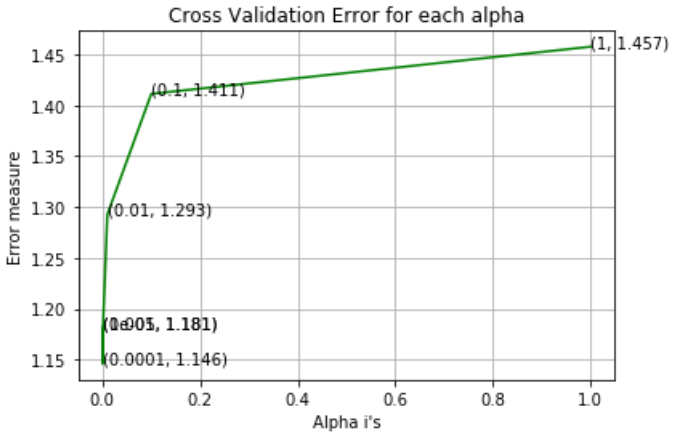
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [119]:

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

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

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
```

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1812527780176447
For values of alpha =  0.0001 The log loss is: 1.1459946439284083
For values of alpha =  0.001 The log loss is: 1.1808488790628733
For values of alpha =  0.01 The log loss is: 1.292676704633847
For values of alpha =  0.1 The log loss is: 1.4110688976661194
For values of alpha =  1 The log loss is: 1.4573717358232532
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0056890427733707
For values of best alpha =  0.0001 The cross validation log loss is: 1.1459946439284083
For values of best alpha =  0.0001 The test log loss is: 1.1868247913139323
```

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 [120]:

```
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  239  genes in train dataset?
Ans
1. In test data 650 out of 665 : 97.74436090225564
2. In cross validation data 518 out of  532 : 97.36842105263158
```

5.3.2 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 [121]:

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

```
Number of Unique Variations : 1922
Truncating_Mutations    61
Deletion                55
Amplification           42
Fusions                 26
Overexpression           4
G12V                     3
C618R                    2
E17K                     2
G13D                     2
R170W                    2
Name: Variation, dtype: int64
```

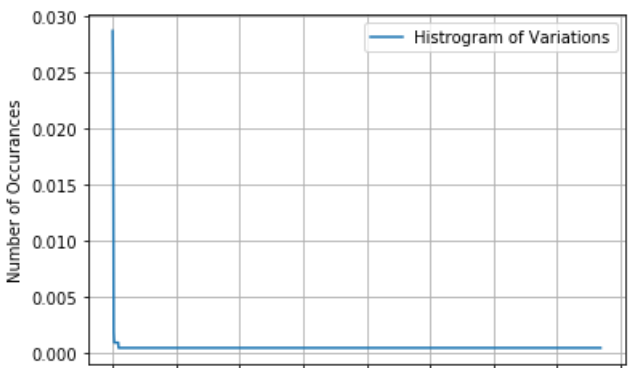
In [122]:

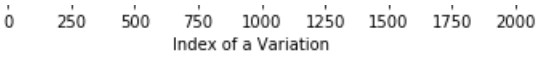
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1922 different categories of variations in the train data, and they are distibuted
as follows
```

In [123]:

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

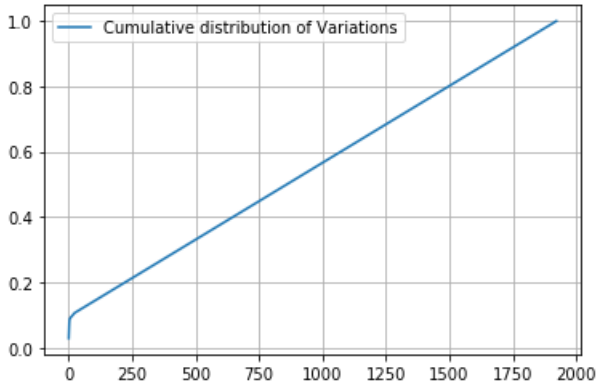




In [124]:

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

```
[0.0287194  0.05461394 0.07438795 ... 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 [125]:

```
# 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 [126]:

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

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

In [127]:

```
# 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 [128]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1951)
```

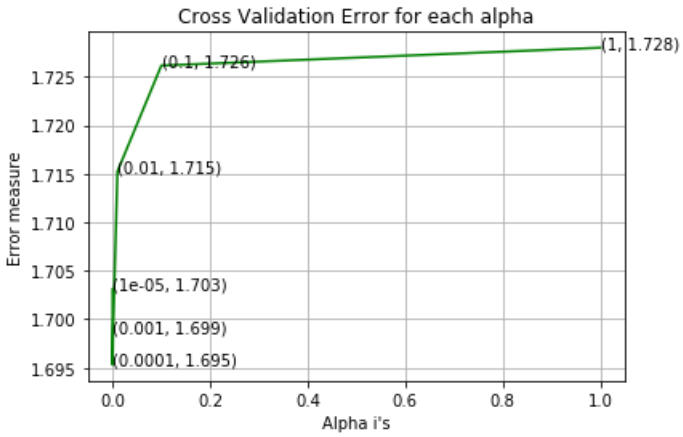
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [129]:

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

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

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

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


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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7031144096224204
For values of alpha =  0.0001 The log loss is: 1.6952936720200913
For values of alpha =  0.001 The log loss is: 1.6985699740939988
For values of alpha =  0.01 The log loss is: 1.715252799861994
For values of alpha =  0.1 The log loss is: 1.7261627481581083
For values of alpha =  1 The log loss is: 1.7280012246650098
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6747205685061664
For values of best alpha =  0.0001 The cross validation log loss is: 1.6952936720200913
For values of best alpha =  0.0001 The test log loss is: 1.7419034851515884
```

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 [130]:

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

```
Q12. How many data points are covered by total  1922  genes in test and cross validation data
sets?
Ans
1. In test data 63 out of 665 : 9.473684210526317
2. In cross validation data 55 out of  532 : 10.338345864661653
```

5.3.3 Univariate Analysis on Text Feature

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

In [131]:

```
# 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 [132]:

```
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 [133]:

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

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

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


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

```
Total number of unique words in train data : 1000
```

In [134]:

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

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


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

In [135]:

```
#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 [136]:

```
# 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 [137]:

```
# 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 [138]:

```
#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 [139]:

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

```
Counter({256.241129578046: 1, 173.10695525596563: 1, 135.52538408900756: 1, 127.52896561589567: 1,
125.17469283028814: 1, 116.65682765161499: 1, 116.59574252364807: 1, 116.06482990538302: 1,
109.14190945304372: 1, 107.0383716203931: 1, 104.05806736655332: 1, 92.70736185338573: 1,
88.36682224442916: 1, 87.43276027955072: 1, 84.4659404389729: 1, 82.79157584933124: 1,
80.85100320153185: 1, 77.4338731716746: 1, 76.64817705091829: 1, 76.40511133104418: 1,
76.23938400131945: 1, 72.477255682567: 1, 70.58358445515648: 1, 69.34451879648839: 1,
66.97863742077438: 1, 66.5331448244344: 1, 66.03821165197478: 1, 64.93978779154955: 1,
62.96609934966018: 1, 62.76907061285698: 1, 62.1485187213016: 1, 61.81282105551466: 1,
60.64880159512491: 1, 58.64797974879591: 1, 58.292746593680626: 1, 57.89503974010945: 1,
55.71962476715555: 1, 54.990428931728715: 1, 53.82437931042262: 1, 53.08913294645455: 1,
50.44952161949981: 1, 49.81567071180134: 1, 48.19637196882876: 1, 47.50292974386925: 1,
45.75142097757206: 1, 45.588863654150614: 1, 44.990625330681084: 1, 44.71829592565254: 1,
44.520297941499784: 1, 44.26003725751944: 1, 44.07280957707106: 1, 43.77416108485414: 1, 42.6794847
58677006: 1, 42.46669409769397: 1, 42.3855599350553: 1, 42.353484186250284: 1, 42.32214613297197:
1, 42.293567672006176: 1, 41.516483240619515: 1, 41.3203486761522: 1, 40.934417622050795: 1,
40.8171670387391: 1, 40.28379576573329: 1, 40.177767266710546: 1, 40.002900081470955: 1,
39.659373524162994: 1, 39.55686099305768: 1, 39.4571508883613: 1, 38.84499852166827: 1,
38.81890851994844: 1, 38.73424335368985: 1, 38.46486007738261: 1, 37.17007533974068: 1,
36.982695246533375: 1, 36.73022706362472: 1, 36.673416227227975: 1, 36.64242147896073: 1,
36.505713121214534: 1, 36.225974893963894: 1, 36.10748460681743: 1, 35.83083397469174: 1,
35.60795152378112: 1, 35.05644771663115: 1, 34.92390477706406: 1, 34.44129593343923: 1,
34.40745564696436: 1, 34.204975519044076: 1, 33.825159124926444: 1, 33.680776009684244: 1,
33.622139122439336: 1, 33.223511235719286: 1, 32.86526391084689: 1, 32.85084601499067: 1,
32.836628820926514: 1, 32.48745410860855: 1, 32.357670591534614: 1, 32.31808691506623: 1,
32.265036683014: 1, 31.95764394218282: 1, 31.886290462552065: 1, 31.381435766228794: 1,
31.347832523594235: 1, 31.282416591302034: 1, 31.209925806289792: 1, 31.11698750676617: 1,
31.06562194098767: 1, 30.895804241162015: 1, 30.854164370850086: 1, 30.832120472789576: 1,
30.388263927035474: 1, 30.33213968758015: 1, 30.26275474441892: 1, 30.226807804075666: 1,
30.225443962055913: 1, 30.17106520797126: 1, 30.076717291357475: 1, 29.846237824820648: 1,
29.654377924946644: 1, 29.640003015564698: 1, 29.226092780296774: 1, 29.20734840322491: 1,
29.16382075784358: 1, 29.14573276018559: 1, 29.108520564743333: 1, 28.9669257970474: 1,
28.515334572139672: 1, 28.462449152198506: 1, 28.232524449303977: 1, 28.22796972871288: 1,
27.632346174410305: 1, 27.519096040622873: 1, 27.480601607831165: 1, 27.418815165335765: 1,
27.287676659700427: 1, 27.262065931931684: 1, 27.153727532038094: 1, 27.05191618100654: 1,
26.614376346329735: 1, 26.423341679331525: 1, 26.358058207007545: 1, 26.233331590231042: 1,
26.12504213372699: 1, 26.016568005737717: 1, 25.879344799065997: 1, 25.743342414237606: 1,
25.68794670726416: 1, 25.417938767259102: 1, 25.412146484869517: 1, 25.333359626384528: 1,
25.327168710875497: 1, 25.206467959696557: 1, 25.067888455674158: 1, 24.9318185575402 54: 1,
```

24.916164055932278: 1, 24.884733988742806: 1, 24.757645717848572: 1, 24.664754091971407: 1,
24.397403185886155: 1, 24.3959800551607: 1, 24.38999457929582: 1, 24.369657880804795: 1,
24.359838799491136: 1, 24.190241285085904: 1, 24.149712016341308: 1, 24.05063288595791: 1,
24.021428485080925: 1, 23.999018947593697: 1, 23.988663699389637: 1, 23.94274545838491: 1,
23.879551176951455: 1, 23.850862457006286: 1, 23.84646884795781: 1, 23.844574722790817: 1,
23.82820610304965: 1, 23.80759892499455: 1, 23.411156910831252: 1, 23.282396648144584: 1,
23.16661158527257: 1, 23.065557431761352: 1, 23.03490843494606: 1, 22.913103599099905: 1,
22.855571560828523: 1, 22.85061004334986: 1, 22.818599062123045: 1, 22.630628378789645: 1,
22.596425772913044: 1, 22.53709592212203: 1, 22.502319621747: 1, 22.486974878365668: 1,
22.454964247335656: 1, 22.41374052401375: 1, 22.164307218092848: 1, 22.142269621357247: 1,
22.110423702338068: 1, 22.054112228010407: 1, 21.943071915538155: 1, 21.901940666342085: 1,
21.896848478493155: 1, 21.841276611231567: 1, 21.828764181147328: 1, 21.82681285029208: 1,
21.822480746764352: 1, 21.779878520725745: 1, 21.73527630650553: 1, 21.673354863921002: 1,
21.620295322031456: 1, 21.505790269013083: 1, 21.377456760010983: 1, 21.331461396824537: 1,
21.329939811944737: 1, 21.327871857847658: 1, 21.31070138801554: 1, 21.299558674373703: 1,
21.24079123981232: 1, 21.173079762614176: 1, 21.086477522499205: 1, 21.067399751152 63: 1,
21.06191272403879: 1, 21.031590942089718: 1, 21.015787561765116: 1, 21.001181524450686: 1,
20.99856745269536: 1, 20.956457651126378: 1, 20.896882330672607: 1, 20.866783653086635: 1,
20.851245563085197: 1, 20.845883167818517: 1, 20.83866053017489: 1, 20.8336350840071: 1,
20.797666752105705: 1, 20.692971027806788: 1, 20.665240567290585: 1, 20.63814405757575: 1,
20.457439077183135: 1, 20.403522282355933: 1, 20.375274219259683: 1, 20.373523512987937: 1,
20.35247504374091: 1, 20.228153793234064: 1, 19.981456516607352: 1, 19.964896984804266: 1,
19.953261970126178: 1, 19.90361178432 02: 1, 19.903105633083396: 1, 19.880365754932303: 1,
19.841588996959214: 1, 19.81067944482213: 1, 19.673132916383157: 1, 19.567504273361997: 1,
19.513362564765366: 1, 19.46829094074155: 1, 19.41566344947255: 1, 19.406176332393827: 1,
19.393658136496825: 1, 19.362646020796337: 1, 19.33815073928876: 1, 19.32017510099251: 1,
19.308788312589833: 1, 19.29925666593078: 1, 19.28078835896499: 1, 19.264792519951474: 1,
19.26288279916738: 1, 19.211138217903862: 1, 19.173342069865157: 1, 19.173185270232207: 1,
19.146249552541384: 1, 18.981836128924748: 1, 18.951557873958908: 1, 18.931796888475812: 1,
18.90011968876797: 1, 18.88608749728139: 1, 18.848317262863702: 1, 18.684717269202654: 1,
18.628648808419726: 1, 18.624411880661096: 1, 18.56191468363326: 1, 18.53705577144176: 1,
18.484627283703603: 1, 18.46372754095935: 1, 18.426660685927267: 1, 18.410918163013537: 1,
18.396875389575055: 1, 18.393388583218087: 1, 18.32069087539059: 1, 18.31683182145446: 1,
18.290696527470462: 1, 18.249641250278245: 1, 18.181434719306456: 1, 18.166962321116063: 1,
18.119292237405787: 1, 18.101957741949004: 1, 18.087187427629082: 1, 17.986727289592366: 1,
17.959032268933296: 1, 17.905181853296902: 1, 17.90072429489245: 1, 17.88711008904507: 1,
17.878522129455998: 1, 17.87676191409831: 1, 17.854877404343977: 1, 17.845904782705446: 1,
17.84346335293244: 1, 17.832951103796344: 1, 17.83237915409625: 1, 17.79242905837648: 1,
17.768950433991375: 1, 17.75298214851803: 1, 17.643995264934617: 1, 17.625540529640038: 1,
17.520458288871 92: 1, 17.50911662186954: 1, 17.50280395559212: 1, 17.491272239770968: 1,
17.37324657977793: 1, 17.350247816508485: 1, 17.329279719504058: 1, 17.317864280602485: 1,
17.312308435793064: 1, 17.224337272559254: 1, 17.217711545022194: 1, 17.212707514363178: 1,
17.211429570231637: 1, 17.166849024033 99: 1, 17.150541068739 58: 1, 17.121165505908337: 1,
17.116995210474414: 1, 17.103887742636964: 1, 17.077667016730814: 1, 17.049773356286924: 1,
17.021629954107546: 1, 17.012243128072704: 1, 16.98473521821695: 1, 16.980408647804307: 1,
16.979637236298867: 1, 16.950723408005 36: 1, 16.937239833029874: 1, 16.89646725208584: 1,
16.873449614827866: 1, 16.845063632645 95: 1, 16.793628824684994: 1, 16.778617091174496: 1,
16.764751854174644: 1, 16.741132009185844: 1, 16.65810157642793: 1, 16.643044724281616: 1,
16.630110613799665: 1, 16.609950574801317: 1, 16.59751995335759: 1, 16.56304082786 54: 1,
16.536517780771735: 1, 16.533339349286262: 1, 16.490876841290 19: 1, 16.466754732601 56: 1,
16.445795936768246: 1, 16.442545920831 68: 1, 16.431962323708785: 1, 16.419608029978 28: 1,
16.355088637537758: 1, 16.344387848469644: 1, 16.343796839527197: 1, 16.293309453199985: 1,
16.289761368309364: 1, 16.244911512313227: 1, 16.194993157047 66: 1, 16.135494710025 67: 1,
16.125806379360 03: 1, 16.039552151695 65: 1, 16.034387937151422: 1, 16.000074532080156: 1,
15.960830498903197: 1, 15.960433230637477: 1, 15.957372381685824: 1, 15.889195398585205: 1,
15.858907077701202: 1, 15.849351463264048: 1, 15.839271537684423: 1, 15.812552249781 12: 1,
15.755546605150 43: 1, 15.660647478791018: 1, 15.655508581034598: 1, 15.588903855019 06: 1,
15.513391934359062: 1, 15.457044234616689: 1, 15.431618523831055: 1, 15.389213324177176: 1,
15.381045075185995: 1, 15.371601390437569: 1, 15.362465939509 51: 1, 15.348221014248404: 1,
15.339385537047532: 1, 15.320549955618738: 1, 15.281102312303995: 1, 15.264606698873077: 1,
15.252654279821025: 1, 15.227819851089661: 1, 15.176653169297 47: 1, 15.175053884998311: 1,
15.137393639433341: 1, 15.131138968019608: 1, 15.120914955634987: 1, 15.041095381655 38: 1,
15.035557643204818: 1, 15.031376320834797: 1, 14.995607588618668: 1, 14.905244211638 45: 1,
14.897135861910686: 1, 14.892306760472025: 1, 14.879786804654515: 1, 14.878445189069703: 1,
14.807948980774015: 1, 14.802024998284798: 1, 14.801032358670637: 1, 14.798434494145 55: 1,
14.795759719767213: 1, 14.789837564061147: 1, 14.762116961273 18: 1, 14.734835680674 41: 1,
14.73244803158637: 1, 14.722094187393134: 1, 14.709556268943366: 1, 14.666671466830504: 1,
14.653234204227333: 1, 14.629306220651008: 1, 14.625214182225825: 1, 14.614554377916 07: 1,
14.607457978474754: 1, 14.588869064270904: 1, 14.586121298921146: 1, 14.553587745406928: 1,
14.513277342236906: 1, 14.501628252517015: 1, 14.482048400189688: 1, 14.480465542264964: 1,
14.4802155390662: 1, 14.471266987995357: 1, 14.45362172500787: 1, 14.447377393922277: 1,
14.44620432420769: 1, 14.443918358237056: 1, 14.428879929720523: 1, 14.427994701632809: 1,
14.41598322844315: 1, 14.402839183370613: 1, 14.387359039742252: 1, 14.317309477439126: 1,
14.309870533320 67: 1, 14.298830693894384: 1, 14.282135601485281: 1, 14.269670121264431: 1,
14.253826826808064: 1, 14.252144640592082: 1, 14.241387444147017: 1, 14.237918951951068: 1,
14.156338268529689: 1, 14.155459979294147: 1, 14.148520056890066: 1, 14.147855718184967: 1,
14.125706580140461: 1, 14.086778952776127: 1, 14.070742372628906: 1, 14.047685561323773: 1,

14.039985372011362: 1, 13.996280039637863: 1, 13.967277626462604: 1, 13.94685550861941: 1,
13.94540787772586: 1, 13.891012742747522: 1, 13.858244324497546: 1, 13.831769617923602: 1,
13.785617490117382: 1, 13.754016403812996: 1, 13.746442045527084: 1, 13.734582025524192: 1,
13.729506511583798: 1, 13.711100102426675: 1, 13.689553348342711: 1, 13.661896342809833: 1,
13.656733415391678: 1, 13.637641920195675: 1, 13.623337957575957: 1, 13.604965882901892: 1,
13.556853391086168: 1, 13.542044038392705: 1, 13.52126255818205: 1, 13.5074921377543 78: 1,
13.4998095176927: 1, 13.43410052155147: 1, 13.433417148311232: 1, 13.432279010646546: 1,
13.393784534711276: 1, 13.392141106300453: 1, 13.37835347854993: 1, 13.366387717714376: 1,
13.365841519283105: 1, 13.31220014899707: 1, 13.248568911592187: 1, 13.198123338556396: 1,
13.179040809161648: 1, 13.142253641795607: 1, 13.081033012664028: 1, 13.080028264156303: 1,
13.076561124699419: 1, 13.06931553156405: 1, 13.069282697565406: 1, 13.066329746008257: 1,
13.025751694770728: 1, 12.875159760198159: 1, 12.87137420580669: 1, 12.870220552511228: 1,
12.870150621553709: 1, 12.860842390430536: 1, 12.85528583309557: 1, 12.840648752350955: 1,
12.830276977923964: 1, 12.800733455527016: 1, 12.77923169896985: 1, 12.768739320482807: 1,
12.767434534320525: 1, 12.754014332163633: 1, 12.73541892459233: 1, 12.725412259443672: 1,
12.724583667244744: 1, 12.701905302390154: 1, 12.633245712256109: 1, 12.618104043495743: 1,
12.554913793439301: 1, 12.542889337155758: 1, 12.525995187524746: 1, 12.515120351501507: 1,
12.506600761372324: 1, 12.504113576401165: 1, 12.494171226973645: 1, 12.481386059926031: 1,
12.434048366862962: 1, 12.432496210970891: 1, 12.419768501647859: 1, 12.395642156696564: 1,
12.391914139048593: 1, 12.374446098217925: 1, 12.36708466147569: 1, 12.3440010652170 62: 1,
12.336868669663792: 1, 12.329043271501861: 1, 12.288524580849673: 1, 12.286087816781203: 1,
12.255273493403163: 1, 12.235263074808945: 1, 12.230946535744069: 1, 12.211775819223075: 1,
12.189648712566413: 1, 12.172408540250593: 1, 12.169771000882786: 1, 12.134754803693909: 1,
12.073710927892199: 1, 12.072170333013148: 1, 12.05952467181948: 1, 12.047394678211205: 1,
12.021576664259602: 1, 12.015353717693468: 1, 12.008668623472179: 1, 11.983211880331279: 1,
11.981303079749951: 1, 11.971649627798623: 1, 11.959952012700796: 1, 11.942539022817602: 1,
11.906721184523652: 1, 11.90010678894679: 1, 11.894291109365723: 1, 11.890345252152265: 1,
11.890343160116977: 1, 11.876914588835453: 1, 11.876540407812294: 1, 11.850235640333294: 1,
11.807183694894062: 1, 11.796578931746804: 1, 11.78064056240228: 1, 11.775819047106072: 1,
11.710270808206802: 1, 11.706609739753043: 1, 11.705126965750846: 1, 11.69284970611656: 1,
11.667098457932246: 1, 11.64360539338719: 1, 11.629148524095937: 1, 11.615222259084028: 1,
11.603535098873506: 1, 11.596287750149509: 1, 11.57070858744606: 1, 11.561989031470773: 1,
11.556259051994758: 1, 11.526296731641807: 1, 11.51936614296361: 1, 11.489487162072107: 1,
11.487432748309184: 1, 11.48005081584417: 1, 11.477519437821451: 1, 11.46667794177704: 1,
11.466381956714564: 1, 11.444601674955518: 1, 11.44335310560736: 1, 11.442949526571 97: 1,
11.437058618607788: 1, 11.436649499662845: 1, 11.402234367766225: 1, 11.391614105625878: 1,
11.372390036248483: 1, 11.35902948080245: 1, 11.331656252965518: 1, 11.329540355446515: 1,
11.322719966340156: 1, 11.276001422837721: 1, 11.23821164502551: 1, 11.226129020481716: 1,
11.22235923026227: 1, 11.216476081029334: 1, 11.211817541014966: 1, 11.136627378546418: 1,
11.126817668258482: 1, 11.12079396829313: 1, 11.113791257713533: 1, 11.103164597211913: 1,
11.071592518618885: 1, 11.063613561257268: 1, 11.057140286503518: 1, 11.052097348251882: 1,
11.010346641045993: 1, 10.992413730324639: 1, 10.967883016402983: 1, 10.952773483693319: 1,
10.947712043233194: 1, 10.924889587677 49: 1, 10.903334689246991: 1, 10.865972342670135: 1,
10.864720712106799: 1, 10.861863762776585: 1, 10.860472922139827: 1, 10.852428826421507: 1,
10.851977271276905: 1, 10.850202908896563: 1, 10.846392379159775: 1, 10.820077926237 39: 1,
10.813113368780714: 1, 10.811982695342955: 1, 10.791985901295083: 1, 10.785108529066573: 1,
10.781475203688123: 1, 10.775392882384 94: 1, 10.767557820431017: 1, 10.763662220170628: 1,
10.76319583794291: 1, 10.754219411446194: 1, 10.716546337757467: 1, 10.696800810927561: 1,
10.696173686118488: 1, 10.695212964086197: 1, 10.684167689868366: 1, 10.683224047347242: 1,
10.672217282962677: 1, 10.663099872176952: 1, 10.634153669199513: 1, 10.627514945933925: 1,
10.624339659430694: 1, 10.621163695129223: 1, 10.618663497804024: 1, 10.616494044050553: 1,
10.608206569424238: 1, 10.586460050387455: 1, 10.583787457441117: 1, 10.534642396519269: 1,
10.523874646885206: 1, 10.501102049903002: 1, 10.48391914919605: 1, 10.471589319457559: 1,
10.468966353780091: 1, 10.454903298509581: 1, 10.439102552772416: 1, 10.409704832217 55: 1,
10.402589234028765: 1, 10.389337533482365: 1, 10.38688745453854: 1, 10.384767884405 27: 1,
10.383325481879549: 1, 10.379284535943134: 1, 10.362021585086316: 1, 10.330433179891651: 1,
10.328786206858567: 1, 10.313413568969134: 1, 10.305765733108933: 1, 10.304753080732942: 1,
10.300484526600643: 1, 10.276066959860078: 1, 10.249159017594412: 1, 10.244267231607692: 1,
10.235855480510246: 1, 10.227445814624657: 1, 10.220502032771671: 1, 10.196495362873 01: 1,
10.194221933890711: 1, 10.192512243341744: 1, 10.191238958688517: 1, 10.188280375483 29: 1,
10.184182539903782: 1, 10.175221369704426: 1, 10.15949055500805: 1, 10.146964767287088: 1,
10.145876919976722: 1, 10.139712522186244: 1, 10.138232700640822: 1, 10.134535047723489: 1,
10.119970281599347: 1, 10.119297601984378: 1, 10.067353787852564: 1, 10.065797596959039: 1,
10.056763478997729: 1, 10.055396729229557: 1, 10.053144175728308: 1, 10.032368870167284: 1,
10.030846276600746: 1, 10.030379916883525: 1, 10.023980261724946: 1, 10.007362526003654: 1,
10.003599278807593: 1, 9.97772383615429: 1, 9.973176369354846: 1, 9.970960155551706: 1,
9.945172187091124: 1, 9.944528817320467: 1, 9.921434230459715: 1, 9.891133084840922: 1,
9.838496182705729: 1, 9.834967832080258: 1, 9.82669990997617: 1, 9.826233884348435: 1,
9.781090475609052: 1, 9.777660466557538: 1, 9.769989222309702: 1, 9.754361842609082: 1,
9.710576889518148: 1, 9.703569258576765: 1, 9.703302637175042: 1, 9.677239892414686: 1,
9.668612619761607: 1, 9.662679974267203: 1, 9.633352496386465: 1, 9.612742917561791: 1,
9.603800444077823: 1, 9.590212308487756: 1, 9.584554107720965: 1, 9.580809331682195: 1,
9.5575762040551: 1, 9.545536999306922: 1, 9.52750153809375: 1, 9.520039280358999: 1,
9.513480256047881: 1, 9.501312812901308: 1, 9.493473864552204: 1, 9.49343191230657: 1,
9.486850564487694: 1, 9.481610526110305: 1, 9.467495852823948: 1, 9.456767355480475: 1,
9.441540717277814: 1, 9.423243741325356: 1, 9.412386238256817: 1, 9.403832066945046: 1,
9.388183209974729: 1, 9.372492648020096: 1, 9.371649136543649: 1, 9.344777466469646: 1,

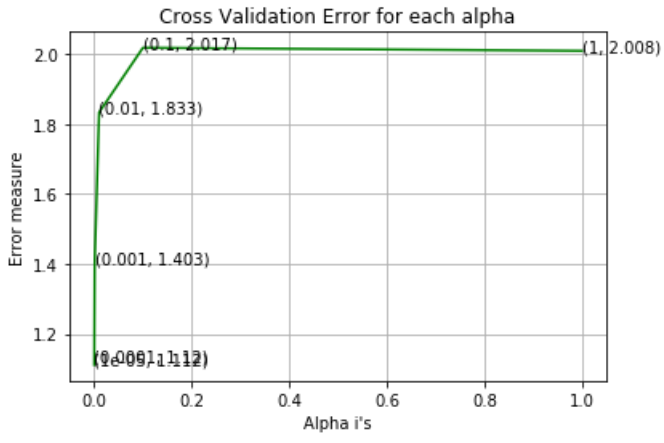
9.336082963654382: 1, 9.327989360006217: 1, 9.31663802803089: 1, 9.315801134418638: 1,
9.315202844059728: 1, 9.294053377246552: 1, 9.285408758416223: 1, 9.270896124345256: 1,
9.270615556980932: 1, 9.263545282921237: 1, 9.246689262467426: 1, 9.244679139553955: 1,
9.243958081445804: 1, 9.233380574408313: 1, 9.226401810047085: 1, 9.225038959575416: 1,
9.221632367066931: 1, 9.220651149459307: 1, 9.216453017886476: 1, 9.20522553292736: 1,
9.204601740840737: 1, 9.204262548612638: 1, 9.186711555430017: 1, 9.17510102475807: 1,
9.17102405322799: 1, 9.170641443583044: 1, 9.166214929789323: 1, 9.154831962555527: 1,
9.153621231292338: 1, 9.130396126611585: 1, 9.130113980976168: 1, 9.129641115328296: 1,
9.113073498038075: 1, 9.109968602005344: 1, 9.09741225860463: 1, 9.094654970367666: 1,
9.094414861776542: 1, 9.086592275829389: 1, 9.084222438675681: 1, 9.082096606325713: 1,
9.081001613402039: 1, 9.042300155419023: 1, 9.04106548802284: 1, 9.040364770785317: 1,
9.037847849585189: 1, 9.031696388782253: 1, 9.02931901391699: 1, 9.02484202159557: 1,
9.021127205173965: 1, 9.019419963088007: 1, 9.019417269226057: 1, 9.009723800195522: 1,
9.007634106748192: 1, 9.007291058557124: 1, 9.004210266909528: 1, 9.003399129928374: 1,
8.997485402233357: 1, 8.991921357206245: 1, 8.97981102091358: 1, 8.966980825752353: 1,
8.965442645890358: 1, 8.962265338147425: 1, 8.93837516047409: 1, 8.915324379924744: 1,
8.892763787249551: 1, 8.865745444612685: 1, 8.852462599605403: 1, 8.85150636930772: 1,
8.846374536638281: 1, 8.844987206226206: 1, 8.838975205988238: 1, 8.835431865080292: 1,
8.832814292401853: 1, 8.792823093700887: 1, 8.788068564067874: 1, 8.787817801437091: 1,
8.777284635414041: 1, 8.770632071469347: 1, 8.767870275668532: 1, 8.738013334433894: 1,
8.725838318342868: 1, 8.703903079065046: 1, 8.693619611704731: 1, 8.692495460555582: 1,
8.690996359596147: 1, 8.679068036988943: 1, 8.67094111058961: 1, 8.658444689340223: 1,
8.658153090563417: 1, 8.657413583601102: 1, 8.655067089726058: 1, 8.648211125233711: 1,
8.608834061942133: 1, 8.593902817449525: 1, 8.585372631695027: 1, 8.570322421804423: 1,
8.557371870024598: 1, 8.556269880593632: 1, 8.54741306330435: 1, 8.524137253631311: 1,
8.516686664678094: 1, 8.507333000539711: 1, 8.50308968313488: 1, 8.491029845901977: 1,
8.461470709835332: 1, 8.460040321537225: 1, 8.431970664383838: 1, 8.39709044119714: 1,
8.391739516432807: 1, 8.38241858078879: 1, 8.362627053577768: 1, 8.317710888594142: 1,
8.293267220747813: 1, 8.290859235236491: 1, 8.289180578859229: 1, 8.279131189840108: 1,
8.237805444850341: 1, 8.231232565352585: 1, 8.224101223834058: 1, 8.221936356250557: 1,
8.191145929279383: 1, 8.184993464560327: 1, 8.168374261777055: 1, 8.14914967900307: 1,
8.14426213347528: 1, 8.132433503466443: 1, 8.127795277023246: 1, 8.116446589192797: 1,
8.115594286816433: 1, 8.108024607630831: 1, 8.104772136916605: 1, 8.098612426960754: 1,
8.09217451895405: 1, 8.086845274044478: 1, 8.079199898515382: 1, 8.076778381210115: 1,
8.071139616046812: 1, 8.046481338159067: 1, 8.043321143457588: 1, 8.018944955631648: 1,
8.005385838506118: 1, 7.996648542144168: 1, 7.990011438016025: 1, 7.9837665061743195: 1,
7.97909285814601: 1, 7.972337295151084: 1, 7.9660852601412255: 1, 7.953282257673386: 1,
7.940481273350145: 1, 7.940155199399057: 1, 7.9310280437817395: 1, 7.917412767574831: 1,
7.905992797930823: 1, 7.9053684755035: 1, 7.8848942815329 39: 1, 7.876482021941843: 1,
7.872956251210587: 1, 7.8568699460915585: 1, 7.835569616072241: 1, 7.822857232337688: 1,
7.820754401911614: 1, 7.806903095161838: 1, 7.805431148747097: 1, 7.7753346150516: 1,
7.772298165541157: 1, 7.768415589008731: 1, 7.754146962732126: 1, 7.752825297168659: 1,
7.745848327430046: 1, 7.734505876822271: 1, 7.714649257726347: 1, 7.705221474523398: 1,
7.70440741456666: 1, 7.704300795723799: 1, 7.691442950166827: 1, 7.633397432448687: 1,
7.631558160522331: 1, 7.610301830022325: 1, 7.587794535836684: 1, 7.582652328742374: 1,
7.581707844629469: 1, 7.577710665512408: 1, 7.531864639447067: 1, 7.522962502532703: 1,
7.4899149076500455: 1, 7.476811516801988: 1, 7.473960128467515: 1, 7.473273618126238: 1, 7.47110621
7241388: 1, 7.471061784655087: 1, 7.452584983679342: 1, 7.449302301080701: 1, 7.441944238393469: 1,
7.4367293506583945: 1, 7.415445421051101: 1, 7.414395547320937: 1, 7.408637606737477: 1, 7.35988026
519101: 1, 7.337162755943645: 1, 7.311789067176892: 1, 7.295322911577347: 1, 7.270822383083905: 1,
7.266703468710836: 1, 7.149803110289343: 1, 7.146451347710431: 1, 7.113358146764231: 1,
7.091745285771835: 1, 7.08854629163655: 1, 7.088234859727977: 1, 7.065549033948573: 1,
7.059088773970895: 1, 7.055846183736933: 1, 7.041182438624592: 1, 7.023927162488542: 1,
7.006138705431957: 1, 6.926228382588332: 1, 6.925116593317424: 1, 6.917939089689684: 1,
6.895728903086082: 1, 6.86350529232473: 1, 6.830146549997769: 1, 6.828168353858422: 1,
6.812785591526605: 1, 6.737989340815086: 1, 6.734363306251 69: 1, 6.701209934913642: 1,
6.672999258461511: 1, 6.569133574660739: 1, 6.411894435929298: 1, 6.334889194968754: 1,
6.212284940850632: 1})

In [140]:

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

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
```

```
# predict(X) Predict class labels for samples in X.

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1115905121025633
For values of alpha =  0.0001 The log loss is: 1.120359127427825
For values of alpha =  0.001 The log loss is: 1.4025186690592888
For values of alpha =  0.01 The log loss is: 1.8325915709477933
For values of alpha =  0.1 The log loss is: 2.017358795672618
For values of alpha =  1 The log loss is: 2.008483819720005
```




```
For values of best alpha =  1e-05 The train log loss is: 0.6965434306701667
For values of best alpha =  1e-05 The cross validation log loss is: 1.1115905121025633
For values of best alpha =  1e-05 The test log loss is: 1.1450626800427544
```

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

Ans. Yes, it seems like!

In [141]:

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    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 [142]:

```
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")
```

```
3.512 % of word of test data appeared in train data
3.836 % of word of Cross Validation appeared in train data
```

6. Machine Learning Models

In [143]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [283]:

```
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 [284]:

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

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

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

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

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

Stacking the three types of features

In [285]:

```
# merging gene, variance and text features

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

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

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

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

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


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

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

In [286]:

```
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, 3189)
(number of data points * number of features) in test data =  (665, 3189)
(number of data points * number of features) in cross validation data = (532, 3189)
```

In [287]:

```
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)
```

6.1. Base Line Model

6.1.1. Naive Bayes

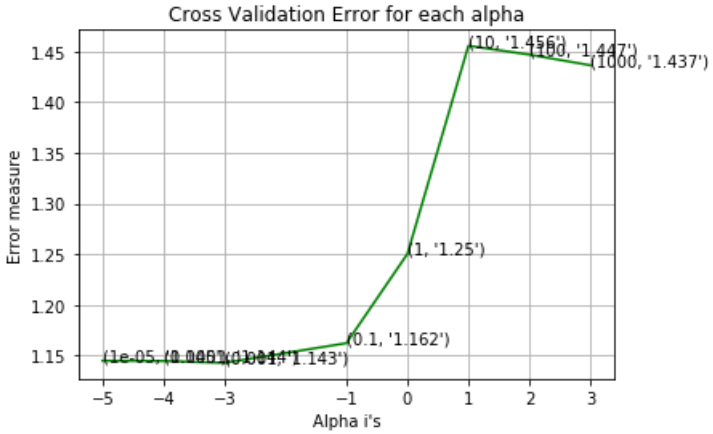
6.1.1.1. Hyper parameter tuning

In [288]:

```
# 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]
```

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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.144647224509575
for alpha = 0.0001
Log Loss : 1.1444134372076307
for alpha = 0.001
Log Loss : 1.1425679206784654
for alpha = 0.1
Log Loss : 1.1620160188159454
for alpha = 1
Log Loss : 1.2502246187385975
for alpha = 10
Log Loss : 1.4558332107036185
for alpha = 100
Log Loss : 1.4470308802695868
for alpha = 1000
Log Loss : 1.4365282971435633
```



```
For values of best alpha =  0.001 The train log loss is: 0.4622541246988854
For values of best alpha =  0.001 The cross validation log loss is: 1.1425679206784654
For values of best alpha =  0.001 The test log loss is: 1.2293718336525843
```

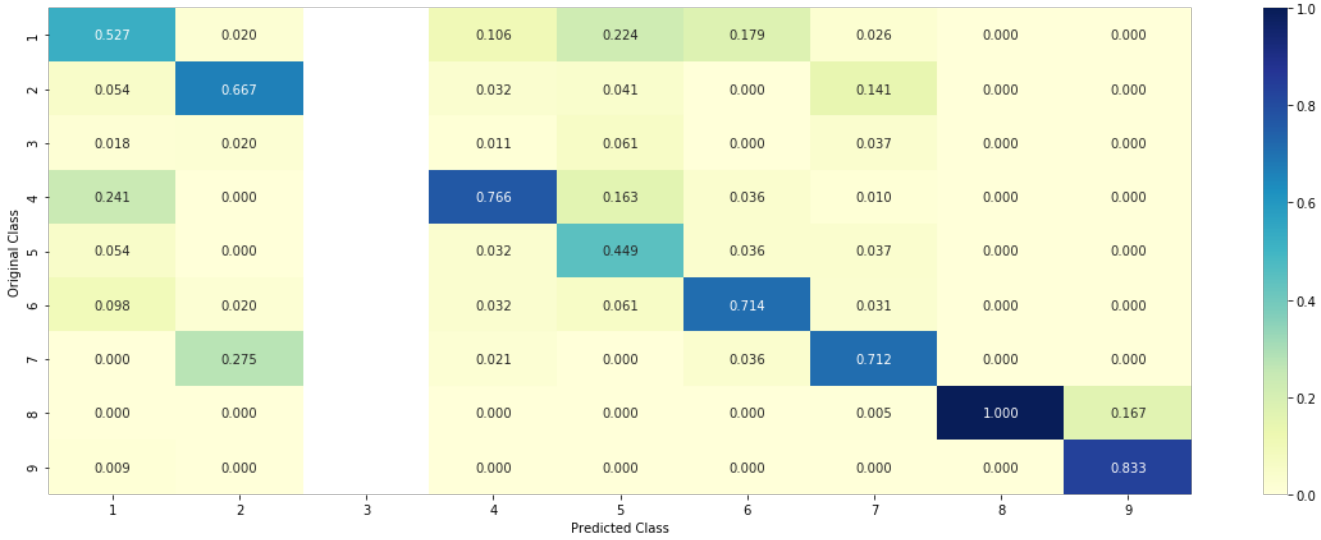
6.1.1.2. Testing the model with best hyper paramters

In [289]:

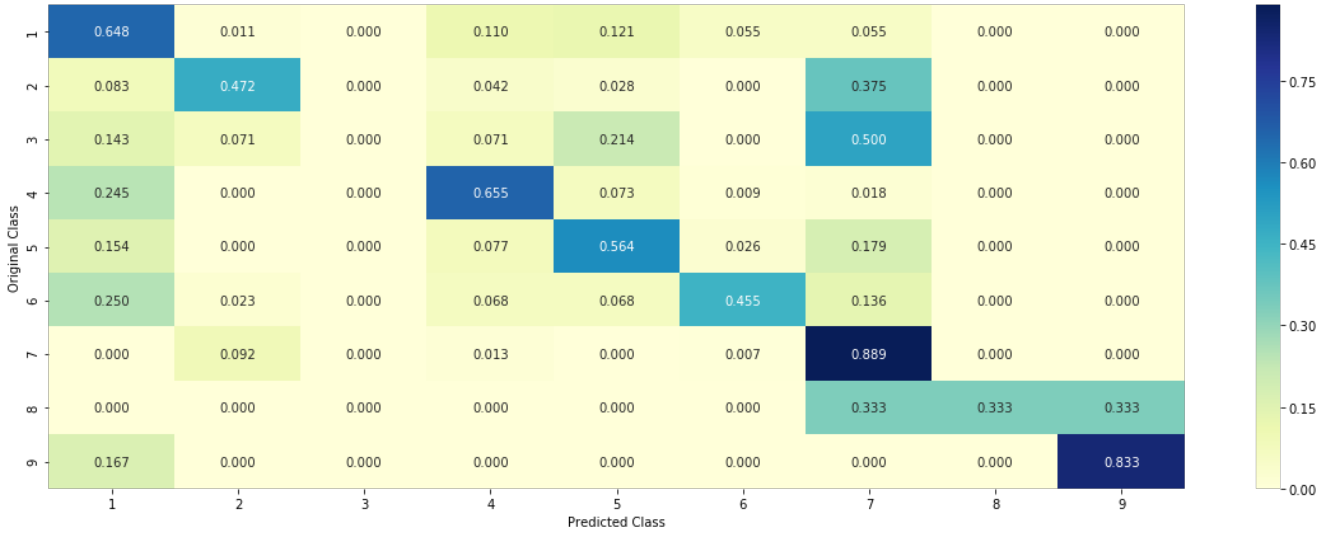
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.1425679206784654
Number of missclassified point : 0.34398496240601506
-------------------- Confusion matrix --------------------
```

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




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



6.1.1.3. Feature Importance, Correctly classified point

In [290]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(abs(-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: [[0.4705 0.0468 0.0137 0.2933 0.0362 0.0339 0.0983 0.0048 0.0026]]
Actual Class : 4
--------------------------------------------------
9 Text feature [one] present in test data point [True]
10 Text feature [function] present in test data point [True]
11 Text feature [protein] present in test data point [True]
12 Text feature [results] present in test data point [True]
13 Text feature [role] present in test data point [True]
```

```
14 Text feature [also] present in test data point [True]
15 Text feature [dna] present in test data point [True]
16 Text feature [loss] present in test data point [True]
17 Text feature [type] present in test data point [True]
18 Text feature [therefore] present in test data point [True]
19 Text feature [however] present in test data point [True]
20 Text feature [two] present in test data point [True]
22 Text feature [specific] present in test data point [True]
23 Text feature [containing] present in test data point [True]
24 Text feature [wild] present in test data point [True]
25 Text feature [table] present in test data point [True]
26 Text feature [whether] present in test data point [True]
27 Text feature [either] present in test data point [True]
28 Text feature [control] present in test data point [True]
29 Text feature [human] present in test data point [True]
30 Text feature [expression] present in test data point [True]
31 Text feature [binding] present in test data point [True]
32 Text feature [determined] present in test data point [True]
33 Text feature [proteins] present in test data point [True]
34 Text feature [discussion] present in test data point [True]
35 Text feature [important] present in test data point [True]
38 Text feature [using] present in test data point [True]
39 Text feature [may] present in test data point [True]
40 Text feature [suggest] present in test data point [True]
41 Text feature [region] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [analysis] present in test data point [True]
44 Text feature [shown] present in test data point [True]
46 Text feature [possible] present in test data point [True]
48 Text feature [used] present in test data point [True]
49 Text feature [gene] present in test data point [True]
51 Text feature [well] present in test data point [True]
52 Text feature [reduced] present in test data point [True]
53 Text feature [furthermore] present in test data point [True]
54 Text feature [including] present in test data point [True]
56 Text feature [critical] present in test data point [True]
58 Text feature [affect] present in test data point [True]
59 Text feature [deletion] present in test data point [True]
60 Text feature [data] present in test data point [True]
61 Text feature [although] present in test data point [True]
63 Text feature [25] present in test data point [True]
64 Text feature [whereas] present in test data point [True]
65 Text feature [within] present in test data point [True]
67 Text feature [three] present in test data point [True]
69 Text feature [following] present in test data point [True]
70 Text feature [directly] present in test data point [True]
72 Text feature [observed] present in test data point [True]
73 Text feature [present] present in test data point [True]
75 Text feature [fig] present in test data point [True]
76 Text feature [cell] present in test data point [True]
77 Text feature [compared] present in test data point [True]
78 Text feature [involved] present in test data point [True]
79 Text feature [indicate] present in test data point [True]
80 Text feature [remaining] present in test data point [True]
81 Text feature [together] present in test data point [True]
82 Text feature [likely] present in test data point [True]
83 Text feature [associated] present in test data point [True]
84 Text feature [genes] present in test data point [True]
85 Text feature [addition] present in test data point [True]
86 Text feature [determine] present in test data point [True]
87 Text feature [described] present in test data point [True]
88 Text feature [studies] present in test data point [True]
89 Text feature [four] present in test data point [True]
90 Text feature [30] present in test data point [True]
91 Text feature [cancer] present in test data point [True]
92 Text feature [mediated] present in test data point [True]
93 Text feature [transcriptional] present in test data point [True]
94 Text feature [cells] present in test data point [True]
95 Text feature [defined] present in test data point [True]
96 Text feature [indicated] present in test data point [True]
97 Text feature [37] present in test data point [True]
Out of the top  100  features  76 are present in query point
```

6.1.1.4. Feature Importance, Incorrectly classified point

In [291]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.2696 0.072  0.0172 0.0866 0.099  0.0797 0.3666 0.006  0.0034]]
Actual Class : 6
--------------------------------------------------
14 Text feature [activation] present in test data point [True]
23 Text feature [cells] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [contrast] present in test data point [True]
28 Text feature [independent] present in test data point [True]
30 Text feature [however] present in test data point [True]
31 Text feature [treatment] present in test data point [True]
35 Text feature [shown] present in test data point [True]
36 Text feature [well] present in test data point [True]
37 Text feature [10] present in test data point [True]
40 Text feature [treated] present in test data point [True]
41 Text feature [addition] present in test data point [True]
43 Text feature [compared] present in test data point [True]
44 Text feature [showed] present in test data point [True]
46 Text feature [mutations] present in test data point [True]
47 Text feature [found] present in test data point [True]
48 Text feature [previously] present in test data point [True]
49 Text feature [recently] present in test data point [True]
54 Text feature [similar] present in test data point [True]
55 Text feature [may] present in test data point [True]
56 Text feature [receptor] present in test data point [True]
58 Text feature [presence] present in test data point [True]
59 Text feature [cell] present in test data point [True]
61 Text feature [although] present in test data point [True]
65 Text feature [increased] present in test data point [True]
67 Text feature [results] present in test data point [True]
69 Text feature [tyrosine] present in test data point [True]
70 Text feature [suggest] present in test data point [True]
71 Text feature [mutation] present in test data point [True]
72 Text feature [described] present in test data point [True]
76 Text feature [mechanism] present in test data point [True]
77 Text feature [reported] present in test data point [True]
79 Text feature [figure] present in test data point [True]
80 Text feature [two] present in test data point [True]
81 Text feature [absence] present in test data point [True]
82 Text feature [examined] present in test data point [True]
83 Text feature [discussion] present in test data point [True]
84 Text feature [obtained] present in test data point [True]
85 Text feature [identified] present in test data point [True]
87 Text feature [interestingly] present in test data point [True]
88 Text feature [observed] present in test data point [True]
89 Text feature [total] present in test data point [True]
92 Text feature [12] present in test data point [True]
93 Text feature [studies] present in test data point [True]
94 Text feature [different] present in test data point [True]
95 Text feature [using] present in test data point [True]
96 Text feature [serum] present in test data point [True]
97 Text feature [15] present in test data point [True]
98 Text feature [clinical] present in test data point [True]
Out of the top  100  features  50 are present in query point
```

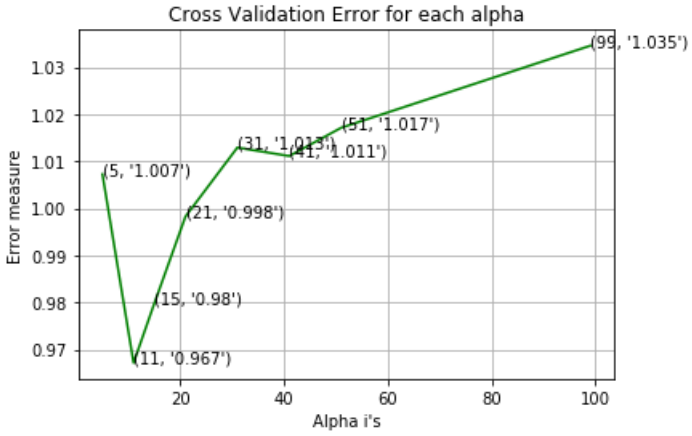
6.2. K Nearest Neighbour Classification

6.2.1. Hyper parameter tuning

In [292]:

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

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


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


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

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


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

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

```
for alpha = 5
Log Loss : 1.0072694236505277
for alpha = 11
Log Loss : 0.9671837344987549
for alpha = 15
Log Loss : 0.9798253850884513
for alpha = 21
Log Loss : 0.9982743699186711
for alpha = 31
Log Loss : 1.012925176322652
for alpha = 41
Log Loss : 1.011125745196786
for alpha = 51
Log Loss : 1.0171314083631673
for alpha = 99
Log Loss : 1.034561896671872
```



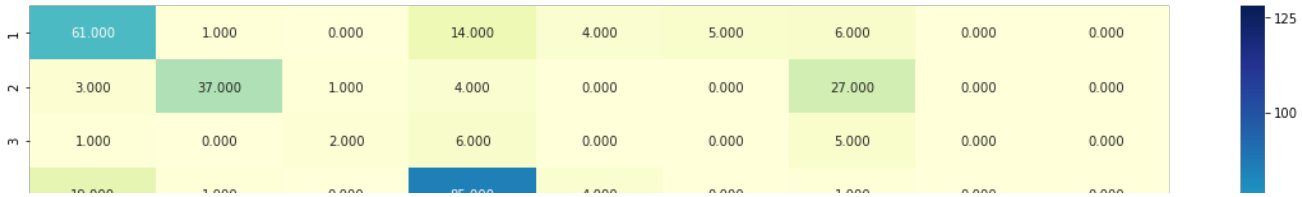

```
For values of best alpha =  11 The train log loss is: 0.6053345698282367
For values of best alpha =  11 The cross validation log loss is: 0.9671837344987549
For values of best alpha =  11 The test log loss is: 1.0860927137251968
```

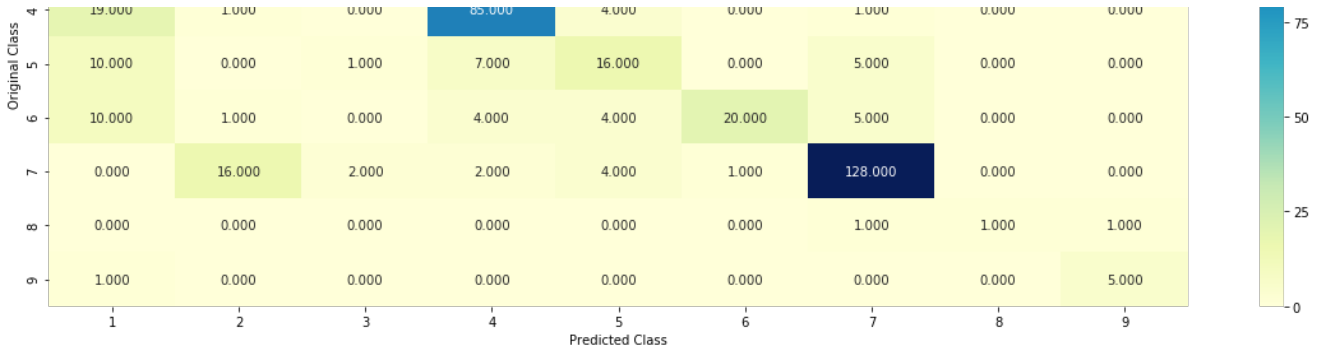
6.2.2. Testing the model with best hyper paramters

In [293]:

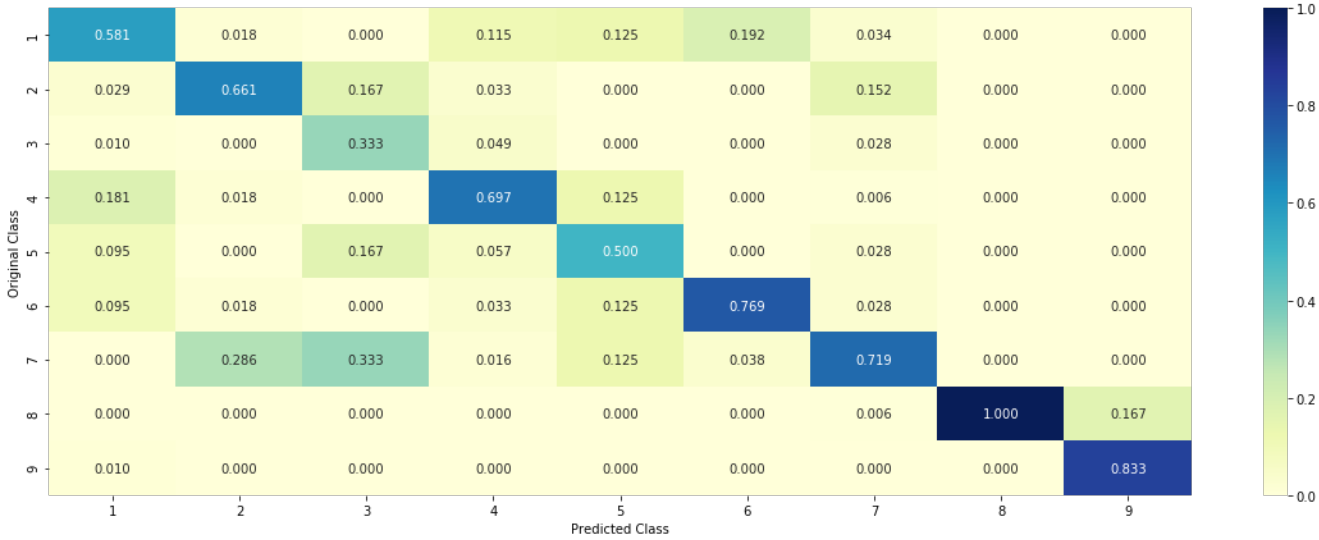
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 0.9671837344987549
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```

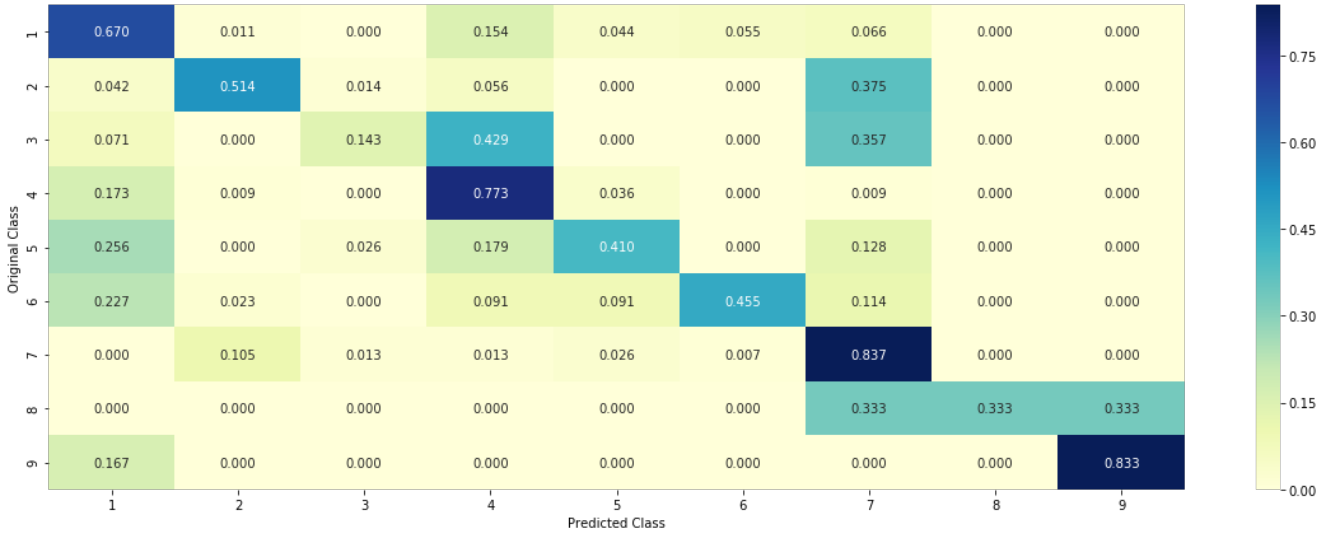




---------- Precision matrix (Columm Sum=1) ----------



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



6.2.3.Sample Query point -1

In [294]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
```

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

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

6.2.4. Sample Query Point-2

In [295]:

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

test_point_index = 100

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

```
Predicted Class : 6
Actual Class : 6
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [1 5 6
6 5 6 6 6 6 6 6]
Fequency of nearest points : Counter({6: 8, 5: 2, 1: 1})
```

6.3. Logistic Regression

6.3.1. With Class balancing

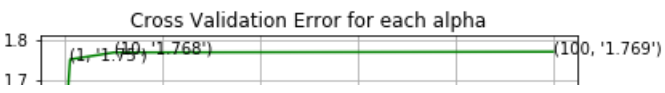
6.3.1.1. Hyper paramter tuning

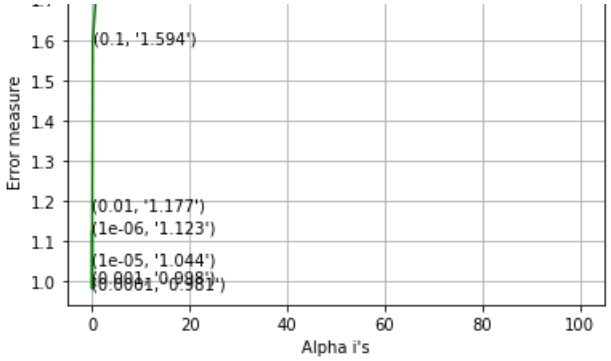
In [296]:

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

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

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


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

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.123374784997356
for alpha = 1e-05
Log Loss : 1.0438370099770464
for alpha = 0.0001
Log Loss : 0.9814287803490364
for alpha = 0.001
Log Loss : 0.9979277563276842
for alpha = 0.01
Log Loss : 1.1771333289490296
for alpha = 0.1
Log Loss : 1.5941957461760887
for alpha = 1
Log Loss : 1.750385765702629
for alpha = 10
Log Loss : 1.7675891127546253
for alpha = 100
Log Loss : 1.7694208999410275
```





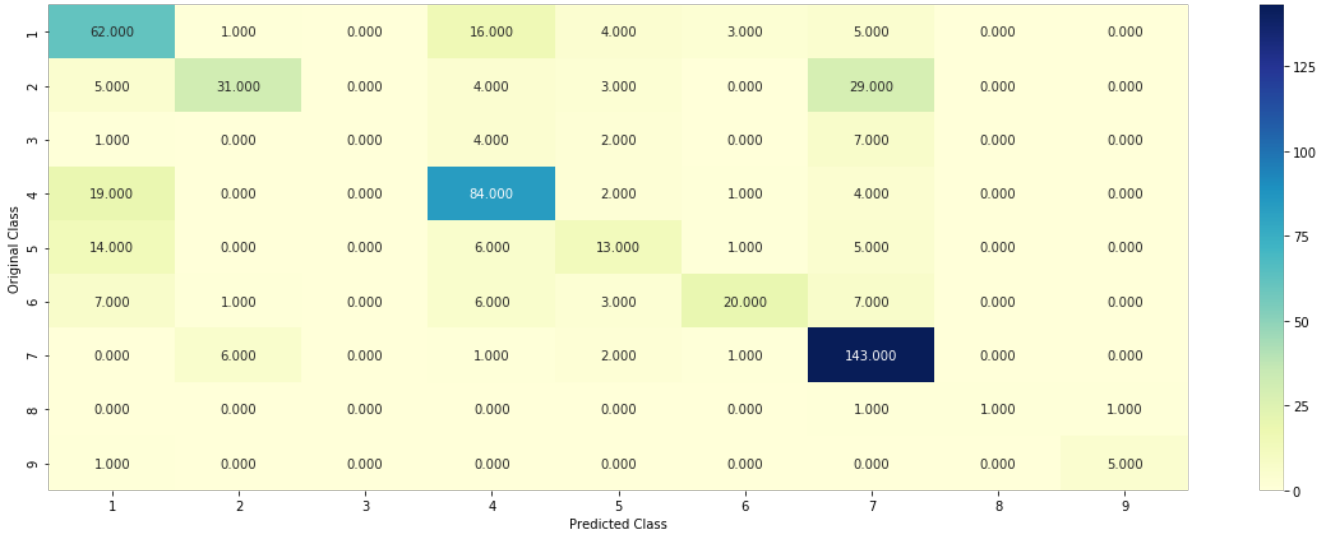

```
For values of best alpha =  0.0001 The train log loss is: 0.4012496372020715
For values of best alpha =  0.0001 The cross validation log loss is: 0.9814287803490364
For values of best alpha =  0.0001 The test log loss is: 1.0338150968853859
```

6.3.1.2. Testing the model with best hyper paramters

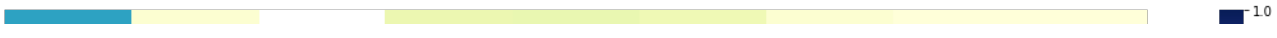
In [297]:

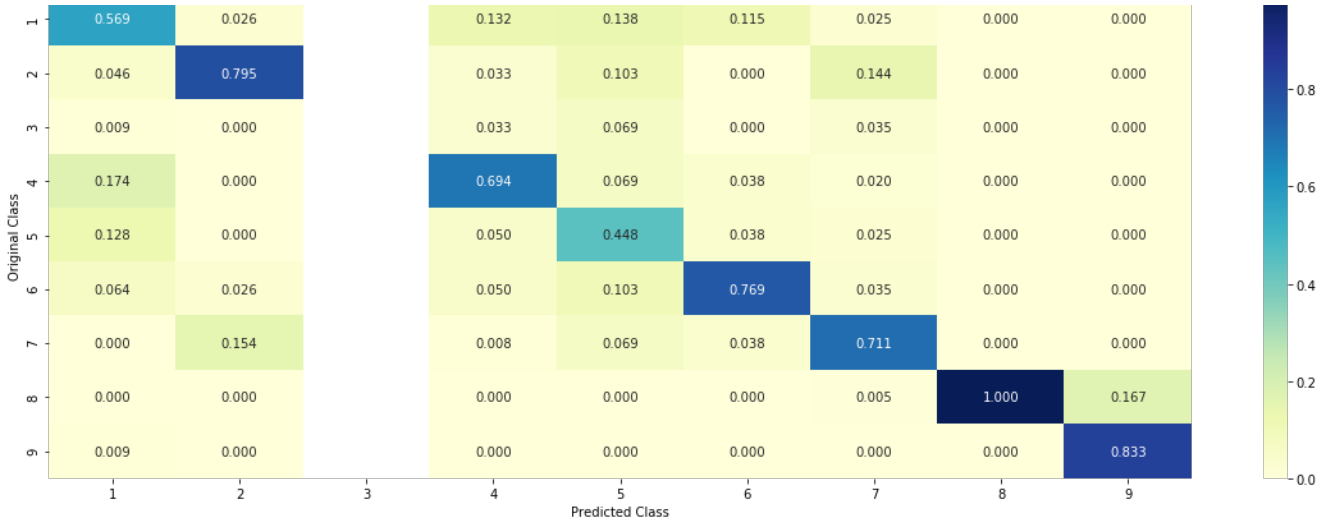
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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

```
Log loss : 0.9814287803490364
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```

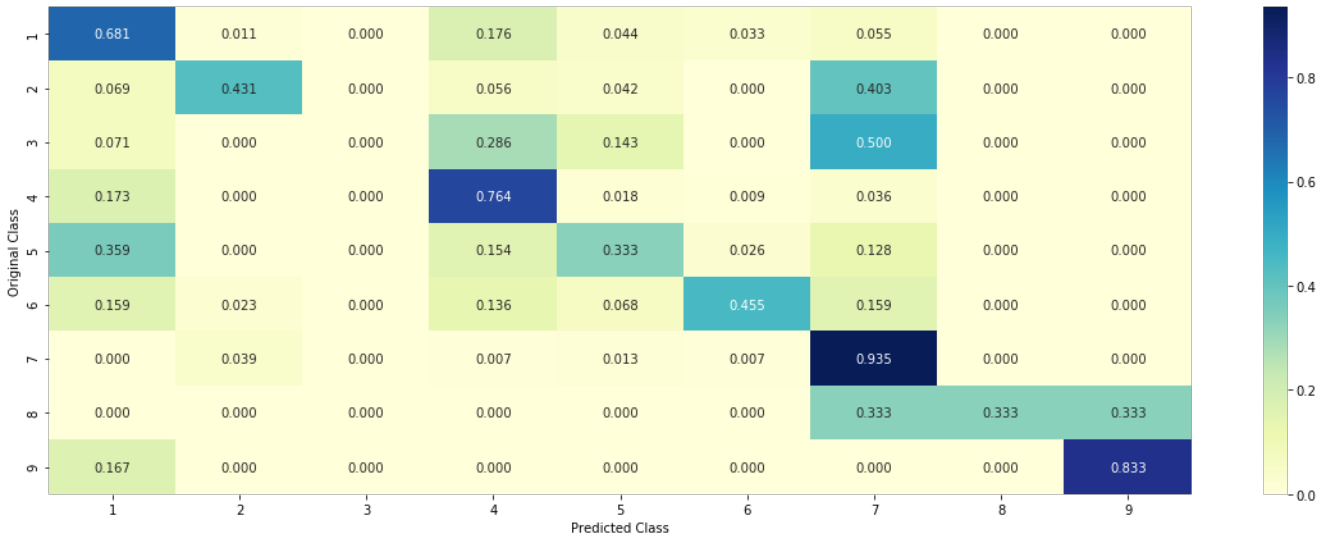



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



6.3.1.3. Feature Importance

In [298]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

6.3.1.3.1. Correctly Classified point

In [299]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-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: [[0.4185 0.0215 0.0167 0.4868 0.0164 0.0144 0.0171 0.0053 0.0032]]
Actual Class : 4
--------------------------------------------------
3 Text feature [positive] present in test data point [True]
7 Text feature [methods] present in test data point [True]
8 Text feature [association] present in test data point [True]
11 Text feature [expression] present in test data point [True]
13 Text feature [25] present in test data point [True]
15 Text feature [corresponding] present in test data point [True]
16 Text feature [alone] present in test data point [True]
17 Text feature [observed] present in test data point [True]
19 Text feature [remaining] present in test data point [True]
20 Text feature [genes] present in test data point [True]
21 Text feature [generated] present in test data point [True]
22 Text feature [28] present in test data point [True]
25 Text feature [45] present in test data point [True]
28 Text feature [various] present in test data point [True]
31 Text feature [cause] present in test data point [True]
32 Text feature [relative] present in test data point [True]
41 Text feature [medium] present in test data point [True]
43 Text feature [50] present in test data point [True]
44 Text feature [conserved] present in test data point [True]
47 Text feature [ii] present in test data point [True]
48 Text feature [residues] present in test data point [True]
51 Text feature [31] present in test data point [True]
60 Text feature [invitrogen] present in test data point [True]
68 Text feature [60] present in test data point [True]
74 Text feature [co] present in test data point [True]
75 Text feature [pathogenic] present in test data point [True]
77 Text feature [44] present in test data point [True]
89 Text feature [number] present in test data point [True]
91 Text feature [similar] present in test data point [True]
92 Text feature [important] present in test data point [True]
95 Text feature [affinity] present in test data point [True]
105 Text feature [times] present in test data point [True]
108 Text feature [finally] present in test data point [True]
110 Text feature [2012] present in test data point [True]
119 Text feature [phosphorylation] present in test data point [True]
136 Text feature [limited] present in test data point [True]
143 Text feature [nucleotide] present in test data point [True]
144 Text feature [70] present in test data point [True]
145 Text feature [could] present in test data point [True]
151 Text feature [40] present in test data point [True]
155 Text feature [majority] present in test data point [True]
161 Text feature [syndrome] present in test data point [True]
163 Text feature [cells] present in test data point [True]
164 Text feature [likely] present in test data point [True]
165 Text feature [fig] present in test data point [True]
168 Text feature [unknown] present in test data point [True]
182 Text feature [three] present in test data point [True]
197 Text feature [residue] present in test data point [True]
221 Text feature [95] present in test data point [True]
243 Text feature [alternative] present in test data point [True]
248 Text feature [including] present in test data point [True]
262 Text feature [reporter] present in test data point [True]
274 Text feature [detected] present in test data point [True]
276 Text feature [published] present in test data point [True]
277 Text feature [human] present in test data point [True]
```

```
279 Text feature [reported] present in test data point [True]
290 Text feature [primary] present in test data point [True]
306 Text feature [colorectal] present in test data point [True]
363 Text feature [individual] present in test data point [True]
373 Text feature [analyzed] present in test data point [True]
425 Text feature [combined] present in test data point [True]
426 Text feature [smad3] present in test data point [True]
439 Text feature [used] present in test data point [True]
442 Text feature [shown] present in test data point [True]
482 Text feature [transcription] present in test data point [True]
485 Text feature [classified] present in test data point [True]
495 Text feature [role] present in test data point [True]
Out of the top  500  features  67 are present in query point
```

6.3.1.3.2. Incorrectly Classified point

In [300]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1929 0.1012 0.0197 0.1071 0.1764 0.2162 0.1727 0.0084 0.0054]]
Actual Class : 6
--------------------------------------------------
1 Text feature [general] present in test data point [True]
3 Text feature [2005] present in test data point [True]
5 Text feature [measured] present in test data point [True]
10 Text feature [determined] present in test data point [True]
13 Text feature [10] present in test data point [True]
27 Text feature [sequencing] present in test data point [True]
42 Text feature [buffer] present in test data point [True]
55 Text feature [domains] present in test data point [True]
64 Text feature [image] present in test data point [True]
73 Text feature [shown] present in test data point [True]
92 Text feature [normal] present in test data point [True]
96 Text feature [repeats] present in test data point [True]
169 Text feature [discussion] present in test data point [True]
191 Text feature [46] present in test data point [True]
193 Text feature [using] present in test data point [True]
246 Text feature [tumours] present in test data point [True]
285 Text feature [31] present in test data point [True]
289 Text feature [leading] present in test data point [True]
336 Text feature [transactivation] present in test data point [True]
347 Text feature [statistical] present in test data point [True]
468 Text feature [22] present in test data point [True]
486 Text feature [analysis] present in test data point [True]
Out of the top  500  features  22 are present in query point
```

6.3.2. Without Class balancing

6.3.2.1. Hyper paramter tuning

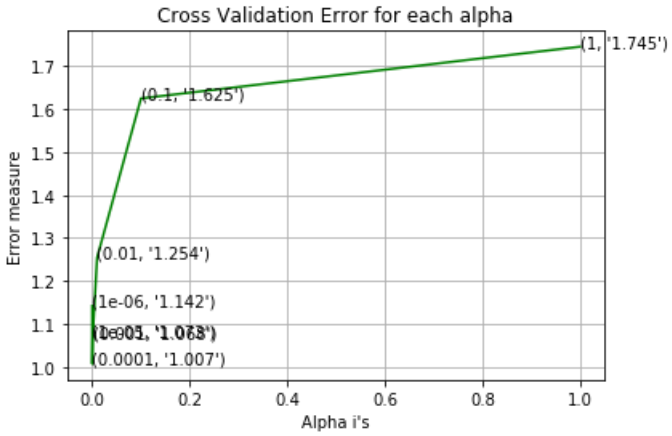
In [301]:

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

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

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1419291653513974
for alpha = 1e-05
Log Loss : 1.0726621494559188
for alpha = 0.0001
Log Loss : 1.007169533516834
for alpha = 0.001
```

```
Log Loss : 1.0679114903368117
for alpha = 0.01
Log Loss : 1.2544125539159294
for alpha = 0.1
Log Loss : 1.6247967453226444
for alpha = 1
Log Loss : 1.7450590593150963
```



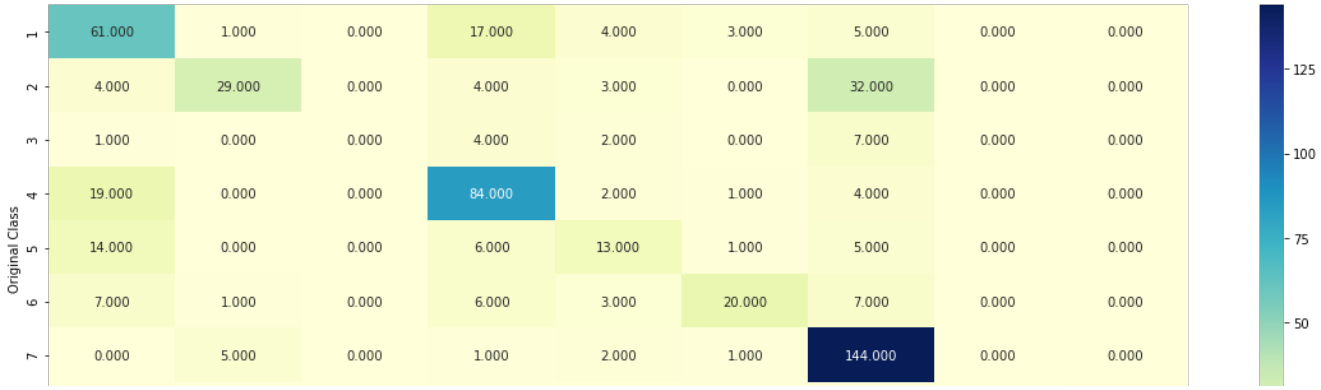

```
For values of best alpha =  0.0001 The train log loss is: 0.3885419921751507
For values of best alpha =  0.0001 The cross validation log loss is: 1.007169533516834
For values of best alpha =  0.0001 The test log loss is: 1.0568723028024094
```

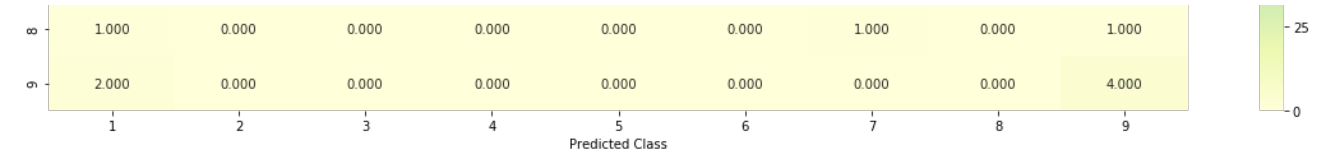
6.3.2.2. Testing model with best hyper parameters

In [302]:

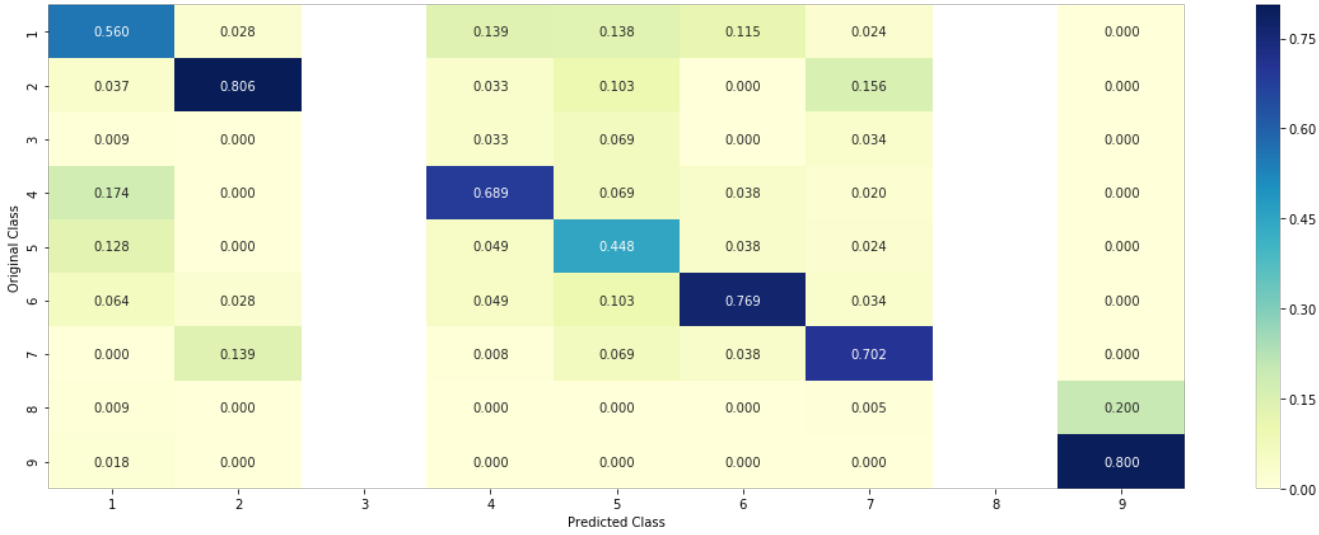
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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 : 1.007169533516834
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```

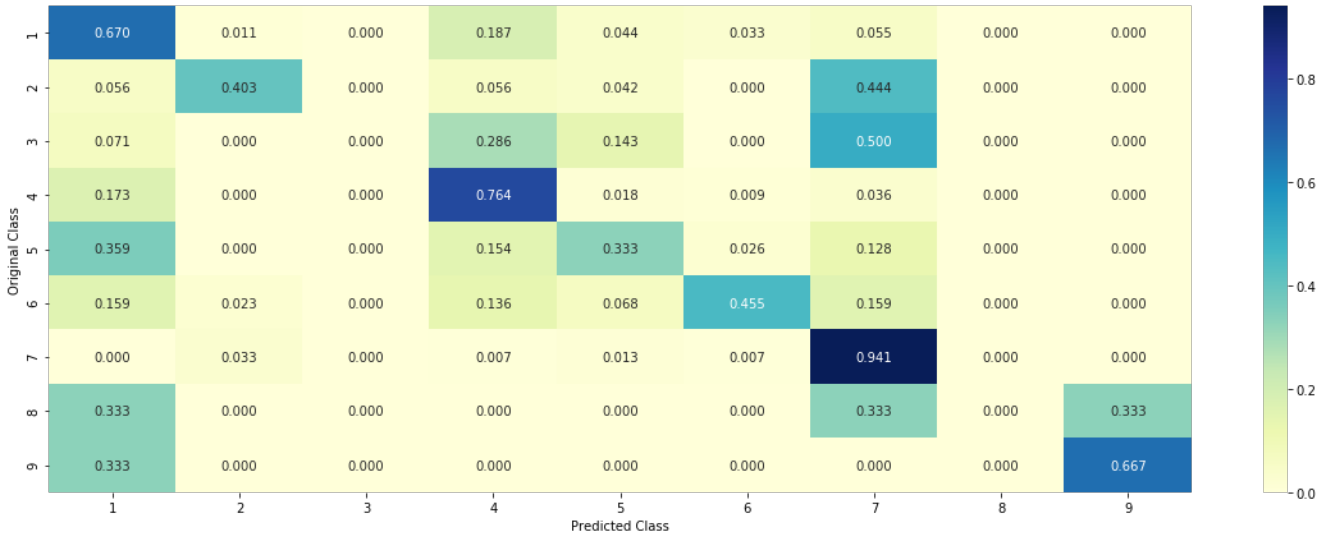





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



6.3.2.3. Feature Importance, Correctly Classified point

In [303]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-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: [[0.4245 0.0221 0.0169 0.4831 0.0161 0.0143 0.016  0.0047 0.0023]]
Actual Class : 4
--------------------------------------------------
1 Text feature [28] present in test data point [True]
3 Text feature [three] present in test data point [True]
4 Text feature [corresponding] present in test data point [True]
5 Text feature [ii] present in test data point [True]
6 Text feature [alternative] present in test data point [True]
7 Text feature [unknown] present in test data point [True]
8 Text feature [cultured] present in test data point [True]
11 Text feature [70] present in test data point [True]
13 Text feature [expression] present in test data point [True]
15 Text feature [cause] present in test data point [True]
21 Text feature [genes] present in test data point [True]
30 Text feature [various] present in test data point [True]
31 Text feature [remaining] present in test data point [True]
32 Text feature [pathogenic] present in test data point [True]
38 Text feature [human] present in test data point [True]
41 Text feature [25] present in test data point [True]
44 Text feature [generated] present in test data point [True]
46 Text feature [times] present in test data point [True]
58 Text feature [50] present in test data point [True]
66 Text feature [association] present in test data point [True]
72 Text feature [number] present in test data point [True]
76 Text feature [majority] present in test data point [True]
77 Text feature [relative] present in test data point [True]
82 Text feature [activated] present in test data point [True]
108 Text feature [medium] present in test data point [True]
112 Text feature [invitrogen] present in test data point [True]
118 Text feature [95] present in test data point [True]
125 Text feature [residues] present in test data point [True]
126 Text feature [methods] present in test data point [True]
130 Text feature [45] present in test data point [True]
133 Text feature [residue] present in test data point [True]
173 Text feature [individual] present in test data point [True]
181 Text feature [limited] present in test data point [True]
185 Text feature [domain] present in test data point [True]
197 Text feature [31] present in test data point [True]
212 Text feature [classified] present in test data point [True]
213 Text feature [alone] present in test data point [True]
228 Text feature [core] present in test data point [True]
230 Text feature [results] present in test data point [True]
231 Text feature [affinity] present in test data point [True]
236 Text feature [60] present in test data point [True]
237 Text feature [could] present in test data point [True]
239 Text feature [important] present in test data point [True]
288 Text feature [primary] present in test data point [True]
297 Text feature [2012] present in test data point [True]
299 Text feature [combined] present in test data point [True]
309 Text feature [nucleotide] present in test data point [True]
324 Text feature [following] present in test data point [True]
331 Text feature [associated] present in test data point [True]
332 Text feature [47] present in test data point [True]
343 Text feature [locus] present in test data point [True]
350 Text feature [co] present in test data point [True]
362 Text feature [fig] present in test data point [True]
366 Text feature [use] present in test data point [True]
371 Text feature [15] present in test data point [True]
376 Text feature [conserved] present in test data point [True]
380 Text feature [44] present in test data point [True]
394 Text feature [finally] present in test data point [True]
454 Text feature [manner] present in test data point [True]
460 Text feature [similar] present in test data point [True]
494 Text feature [role] present in test data point [True]
495 Text feature [including] present in test data point [True]
Out of the top  500  features  62 are present in query point
```

6.3.2.4. Feature Importance, Inorrectly Classified point

In [304]:

```
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 :", 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(abs(-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: [[0.2078 0.1078 0.0195 0.101  0.1841 0.2003 0.1676 0.0078 0.004 ]]
Actual Class : 6
--------------------------------------------------
1 Text feature [relative] present in test data point [True]
2 Text feature [second] present in test data point [True]
4 Text feature [30] present in test data point [True]
5 Text feature [range] present in test data point [True]
6 Text feature [ratio] present in test data point [True]
10 Text feature [transfection] present in test data point [True]
17 Text feature [mediated] present in test data point [True]
29 Text feature [figure] present in test data point [True]
32 Text feature [common] present in test data point [True]
34 Text feature [features] present in test data point [True]
41 Text feature [studies] present in test data point [True]
74 Text feature [obtained] present in test data point [True]
99 Text feature [secondary] present in test data point [True]
103 Text feature [regulated] present in test data point [True]
129 Text feature [spectrum] present in test data point [True]
167 Text feature [prostate] present in test data point [True]
180 Text feature [database] present in test data point [True]
216 Text feature [sites] present in test data point [True]
228 Text feature [activities] present in test data point [True]
285 Text feature [sample] present in test data point [True]
287 Text feature [generation] present in test data point [True]
297 Text feature [consistent] present in test data point [True]
305 Text feature [ar] present in test data point [True]
306 Text feature [resulted] present in test data point [True]
318 Text feature [western] present in test data point [True]
326 Text feature [major] present in test data point [True]
329 Text feature [2007] present in test data point [True]
345 Text feature [pcr] present in test data point [True]
347 Text feature [using] present in test data point [True]
364 Text feature [25] present in test data point [True]
416 Text feature [tumours] present in test data point [True]
456 Text feature [test] present in test data point [True]
464 Text feature [ml] present in test data point [True]
484 Text feature [caused] present in test data point [True]
485 Text feature [affected] present in test data point [True]
Out of the top  500  features  35 are present in query point
```

6.4. Linear Support Vector Machines

6.4.1. Hyper paramter tuning

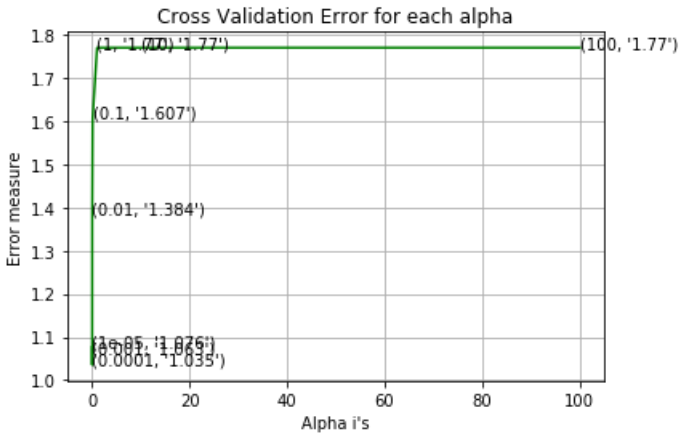
In [305]:

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

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

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

```
# --------------------------------



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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.076256165158973
for C = 0.0001
Log Loss : 1.0353250346043332
for C = 0.001
Log Loss : 1.0630393046928746
for C = 0.01
Log Loss : 1.3838341857800263
for C = 0.1
Log Loss : 1.606687249124266
for C = 1
Log Loss : 1.7697071464826843
for C = 10
Log Loss : 1.7697062430967585
```

for C = 100
Log Loss : 1.769707226680088



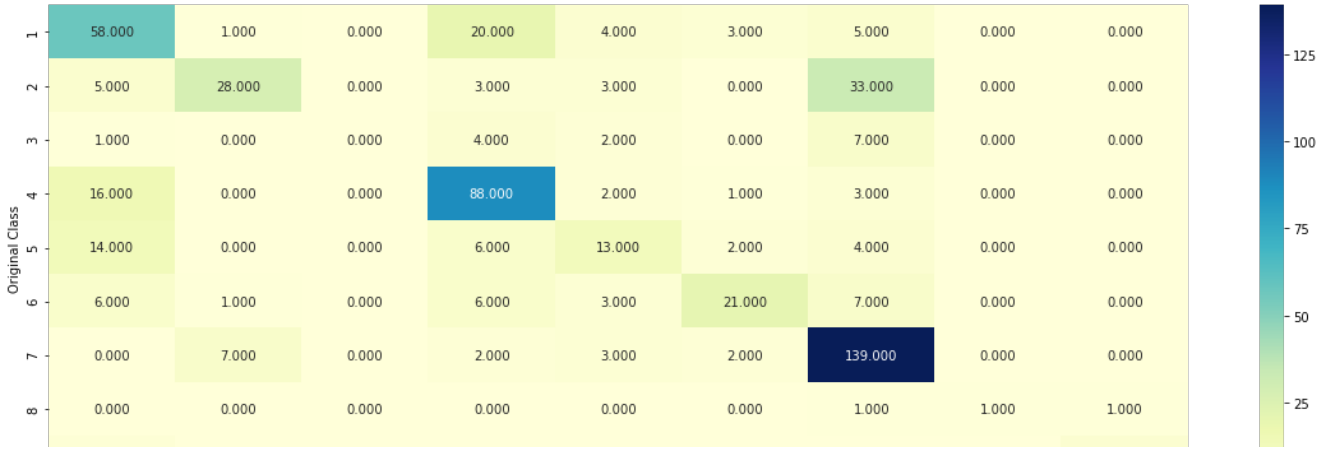
For values of best alpha = 0.0001 The train log loss is: 0.3176879924392105
For values of best alpha = 0.0001 The cross validation log loss is: 1.0353250346043332
For values of best alpha = 0.0001 The test log loss is: 1.07444445723241

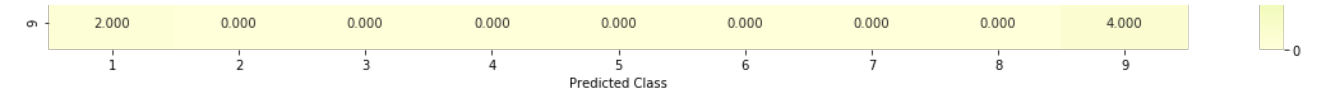
6.4.2. Testing model with best hyper parameters

In [306]:

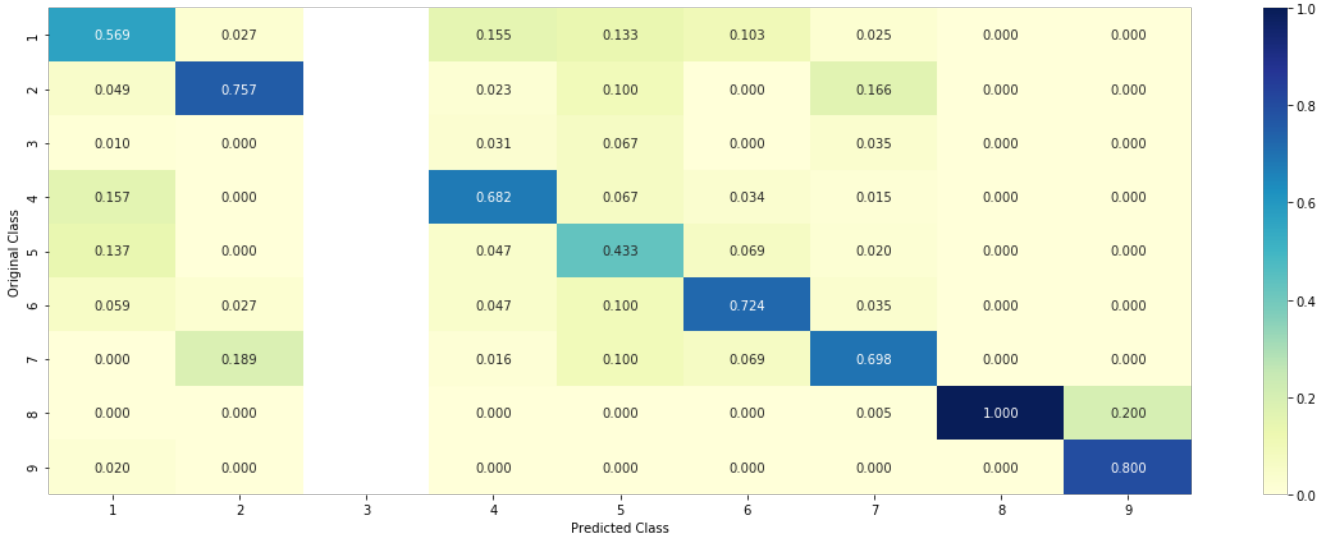
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

Log loss : 1.0353250346043332
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------

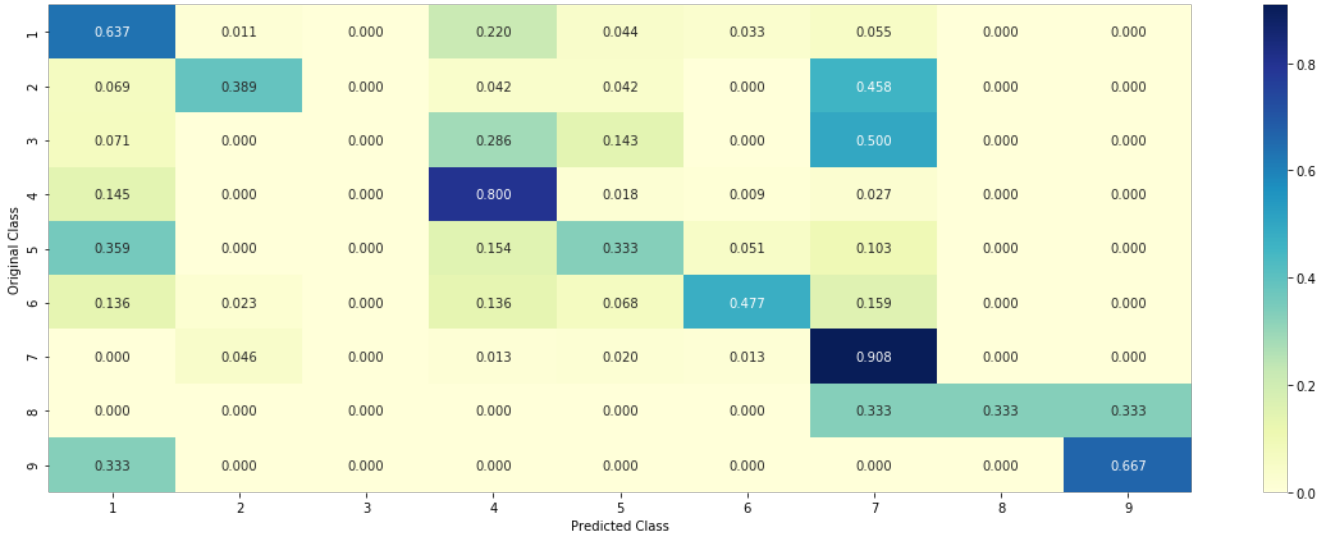





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



6.3.3. Feature Importance

6.3.3.1. For Correctly classified point

In [307]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-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: [[0.3796 0.0343 0.0203 0.4676 0.0234 0.0336 0.0339 0.004  0.0032]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

6.3.3.2. For Incorrectly classified point

In [308]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1755 0.1276 0.0427 0.1063 0.1926 0.2137 0.129  0.0066 0.0059]]
Actual Class : 6
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

6.5 Random Forest Classifier

6.5.1. Hyper paramter tuning (With One hot Encoding)

In [309]:

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

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

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

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


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

```
# 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))
```

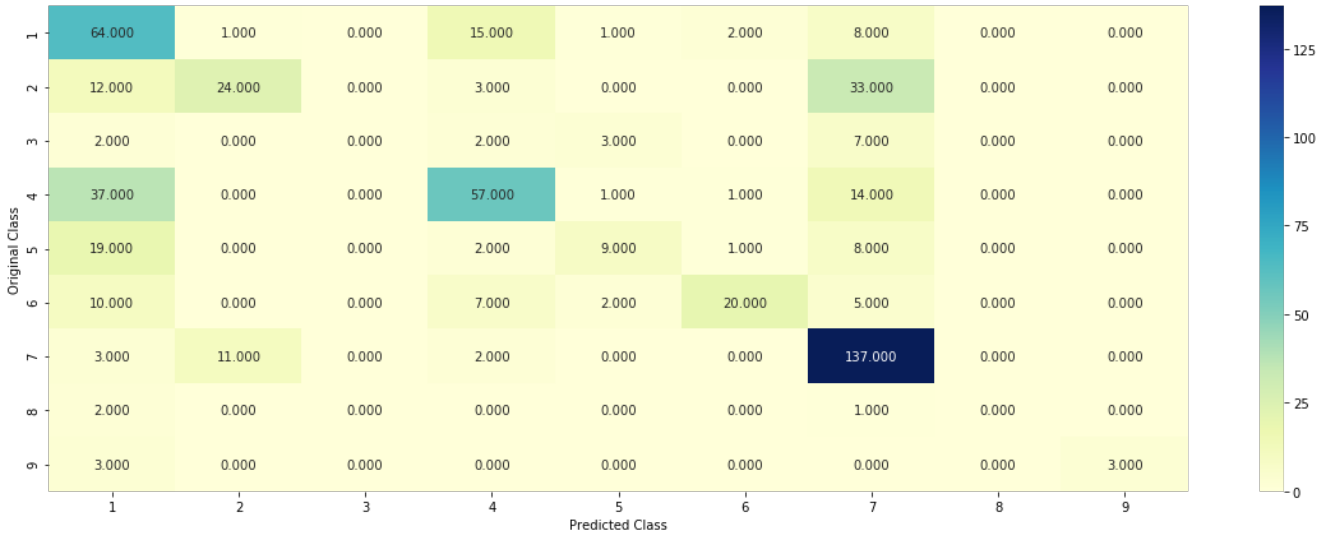
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2025945681963461
for n_estimators = 100 and max depth =  10
Log Loss : 1.221206415559088
for n_estimators = 200 and max depth =  5
Log Loss : 1.1946377521731968
for n_estimators = 200 and max depth =  10
Log Loss : 1.2133218920165902
for n_estimators = 500 and max depth =  5
Log Loss : 1.1870809295424987
for n_estimators = 500 and max depth =  10
Log Loss : 1.209199902713746
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1821679007012513
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2099502227091266
for n_estimators = 2000 and max depth =  5
Log Loss : 1.179384319610551
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2089577764212964
For values of best estimator =  2000 The train log loss is: 0.8654564657868419
For values of best estimator =  2000 The cross validation log loss is: 1.179384319610551
For values of best estimator =  2000 The test log loss is: 1.2072718835095686
```

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

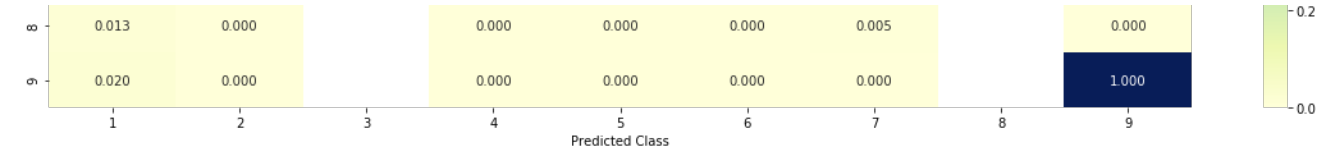
In [310]:

```
# --------------------------------
# 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/
# --------------------------------

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)
```

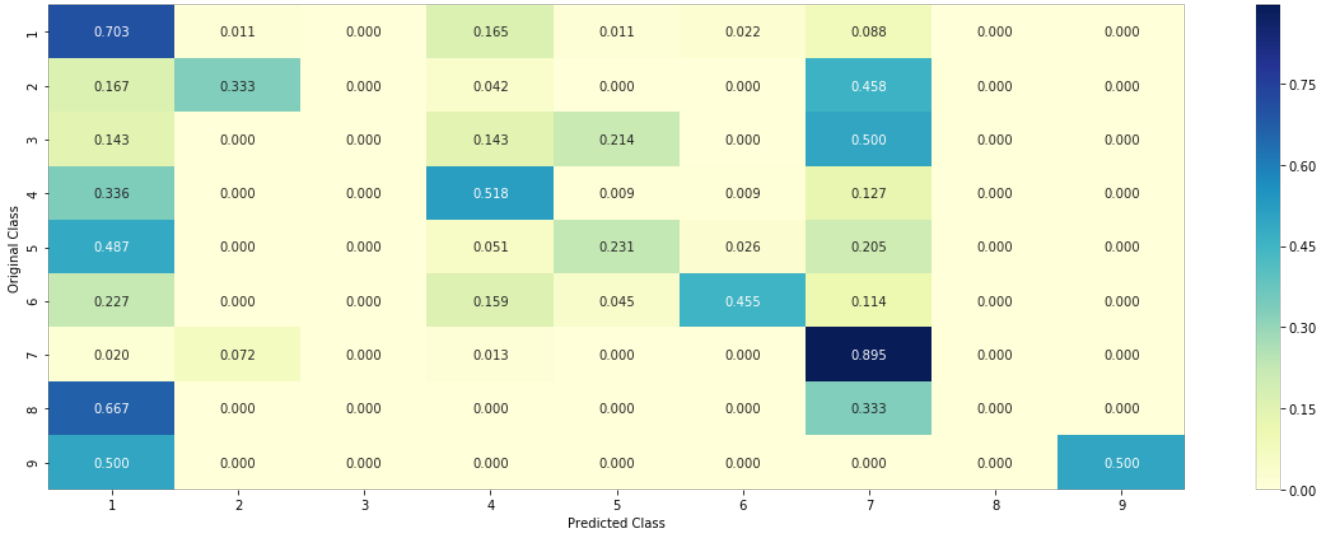
```
Log loss : 1.179384319610551
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```




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




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



6.5.3. Feature Importance

6.5.3.1. Correctly Classified point

In [311]:

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

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3583 0.03   0.0149 0.4327 0.0525 0.0463 0.0521 0.0063 0.0068]]
Actual Class : 4
--------------------------------------------------
3 Text feature [activation] present in test data point [True]
4 Text feature [phosphorylation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [activated] present in test data point [True]
8 Text feature [suppressor] present in test data point [True]
9 Text feature [function] present in test data point [True]
11 Text feature [loss] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
15 Text feature [missense] present in test data point [True]
18 Text feature [protein] present in test data point [True]
21 Text feature [signaling] present in test data point [True]
22 Text feature [stability] present in test data point [True]
27 Text feature [pathogenic] present in test data point [True]
28 Text feature [functional] present in test data point [True]
```

```
28 Text feature [functional] present in test data point [True]
33 Text feature [constitutively] present in test data point [True]
35 Text feature [receptor] present in test data point [True]
36 Text feature [growth] present in test data point [True]
39 Text feature [cells] present in test data point [True]
40 Text feature [treated] present in test data point [True]
44 Text feature [expression] present in test data point [True]
51 Text feature [classified] present in test data point [True]
55 Text feature [advanced] present in test data point [True]
56 Text feature [proteins] present in test data point [True]
57 Text feature [patients] present in test data point [True]
60 Text feature [cell] present in test data point [True]
63 Text feature [inactivation] present in test data point [True]
64 Text feature [phosphorylated] present in test data point [True]
65 Text feature [sensitivity] present in test data point [True]
71 Text feature [predicted] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
76 Text feature [dna] present in test data point [True]
77 Text feature [lines] present in test data point [True]
86 Text feature [harboring] present in test data point [True]
88 Text feature [patient] present in test data point [True]
96 Text feature [nuclear] present in test data point [True]
Out of the top  100  features  35 are present in query point
```

6.5.3.2. Inorrectly Classified point

In [312]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3525 0.0338 0.0196 0.3918 0.0745 0.0609 0.0536 0.0064 0.0067]]
Actuall Class : 6
--------------------------------------------------
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [activation] present in test data point [True]
9 Text feature [function] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [loss] present in test data point [True]
15 Text feature [missense] present in test data point [True]
18 Text feature [protein] present in test data point [True]
22 Text feature [stability] present in test data point [True]
27 Text feature [pathogenic] present in test data point [True]
28 Text feature [functional] present in test data point [True]
35 Text feature [receptor] present in test data point [True]
39 Text feature [cells] present in test data point [True]
40 Text feature [treated] present in test data point [True]
44 Text feature [expression] present in test data point [True]
47 Text feature [resistance] present in test data point [True]
51 Text feature [classified] present in test data point [True]
56 Text feature [proteins] present in test data point [True]
57 Text feature [patients] present in test data point [True]
60 Text feature [cell] present in test data point [True]
65 Text feature [sensitivity] present in test data point [True]
71 Text feature [predicted] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
76 Text feature [dna] present in test data point [True]
77 Text feature [lines] present in test data point [True]
82 Text feature [expressing] present in test data point [True]
84 Text feature [response] present in test data point [True]
85 Text feature [ligand] present in test data point [True]
88 Text feature [patient] present in test data point [True]
89 Text feature [variant] present in test data point [True]
94 Text feature [history] present in test data point [True]
96 Text feature [nuclear] present in test data point [True]
```

```
Out of the top  100  features  31 are present in query point
```

6.5.3. Hyper paramter tuning (With Response Coding)

In [313]:

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

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

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

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


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

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

best_alpha = np.argmin(cv_log_error_array)
```

```
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 : 2.0586152486273193
for n_estimators = 10 and max depth =  3
Log Loss : 1.6331089709318751
for n_estimators = 10 and max depth =  5
Log Loss : 1.421496745484747
for n_estimators = 10 and max depth =  10
Log Loss : 1.7421003668254873
for n_estimators = 50 and max depth =  2
Log Loss : 1.6544052885632121
for n_estimators = 50 and max depth =  3
Log Loss : 1.3635526672160734
for n_estimators = 50 and max depth =  5
Log Loss : 1.3605576898907548
for n_estimators = 50 and max depth =  10
Log Loss : 1.6683239493878115
for n_estimators = 100 and max depth =  2
Log Loss : 1.5010646775401784
for n_estimators = 100 and max depth =  3
Log Loss : 1.4498900801138432
for n_estimators = 100 and max depth =  5
Log Loss : 1.3452969552924314
for n_estimators = 100 and max depth =  10
Log Loss : 1.731017923744125
for n_estimators = 200 and max depth =  2
Log Loss : 1.5605923469898648
for n_estimators = 200 and max depth =  3
Log Loss : 1.4329831261070476
for n_estimators = 200 and max depth =  5
Log Loss : 1.3696704682864844
for n_estimators = 200 and max depth =  10
Log Loss : 1.7263035966437579
for n_estimators = 500 and max depth =  2
Log Loss : 1.6028087397528639
for n_estimators = 500 and max depth =  3
Log Loss : 1.487278322718902
for n_estimators = 500 and max depth =  5
Log Loss : 1.3565133959101345
for n_estimators = 500 and max depth =  10
Log Loss : 1.7504476659829868
for n_estimators = 1000 and max depth =  2
Log Loss : 1.557198381950113
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4696992745390047
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3120308705486212
for n_estimators = 1000 and max depth =  10
Log Loss : 1.738703586338356
For values of best alpha =  1000 The train log loss is: 0.06076387412940683
For values of best alpha =  1000 The cross validation log loss is: 1.312030870548621
For values of best alpha =  1000 The test log loss is: 1.399037220285708
```

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

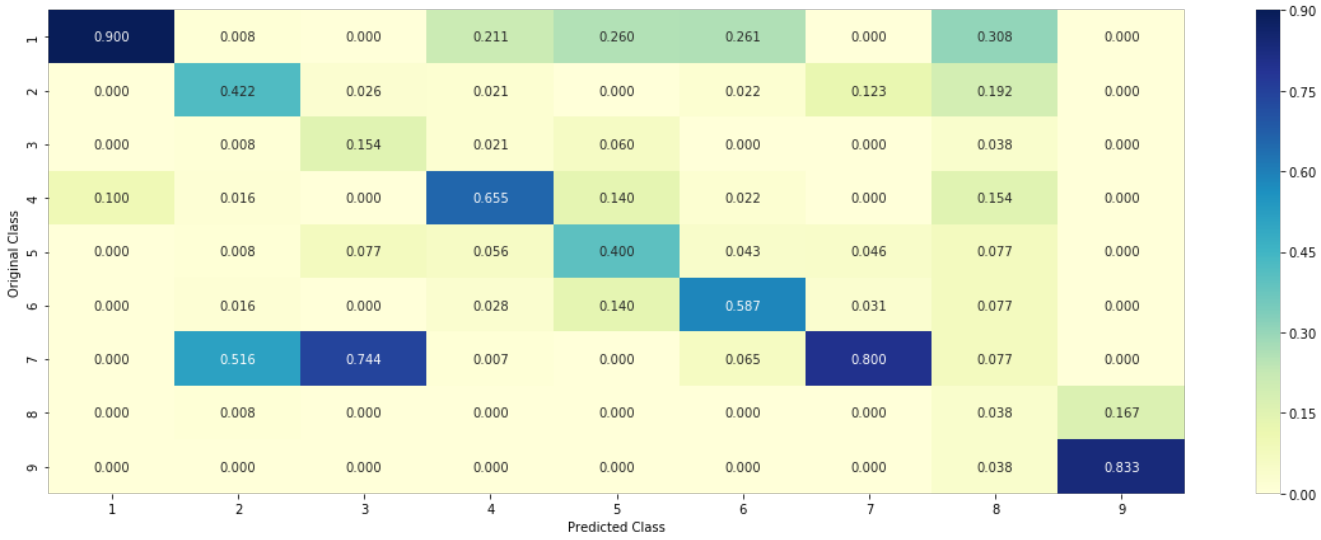
In [314]:

```
# -------------------------------
# default parameters
```

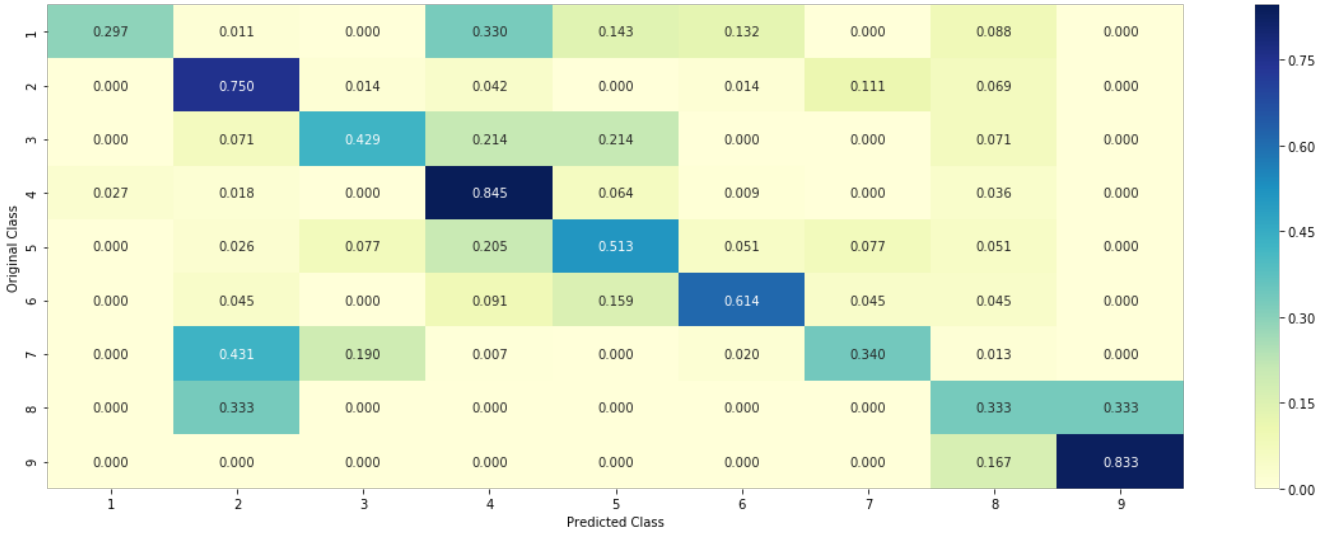
```
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, 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/
# --------------------------------

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

```
Log loss : 1.312030870548621
Number of mis-classified points : 0.4642857142857143
-------------------- Confusion matrix --------------------
```


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



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



6.5.5. Feature Importance

6.5.5.1. Correctly Classified point

In [315]:

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


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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2    0.0252 0.1241 0.429  0.0653 0.0685 0.0132 0.0417 0.033 ]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

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

6.5.5.2. Incorrectly Classified point

In [316]:

```
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 : 6
Predicted Class Probabilities: [[0.0543 0.0456 0.0807 0.0585 0.2122 0.4208 0.0304 0.0625 0.0349]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

6.7 Stack the models

6.7.1 testing with hyper parameter tuning

In [317]:

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

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

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


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

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


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

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

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

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


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

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


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

```
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

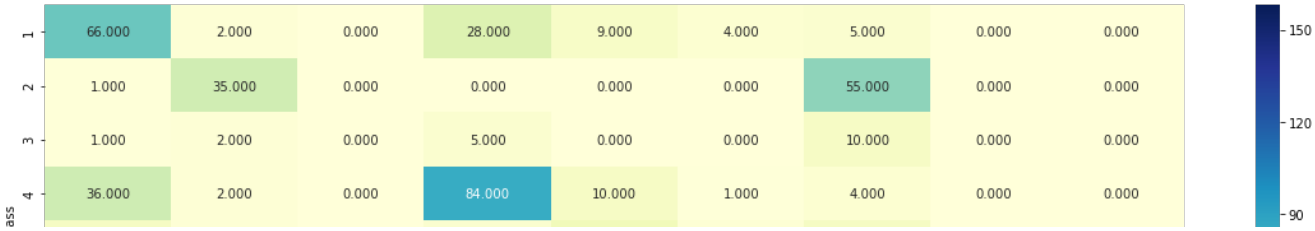
```
Logistic Regression :  Log Loss: 1.00
Support vector machines : Log Loss: 1.77
Naive Bayes : Log Loss: 1.14
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.816
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.707
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.279
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.137
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.454
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.911
```

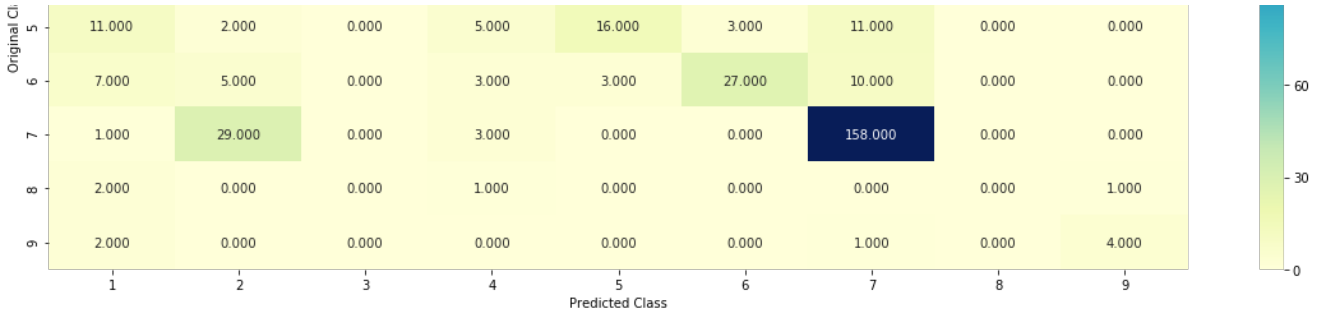
6.7.2 testing the model with the best hyper parameters

In [318]:

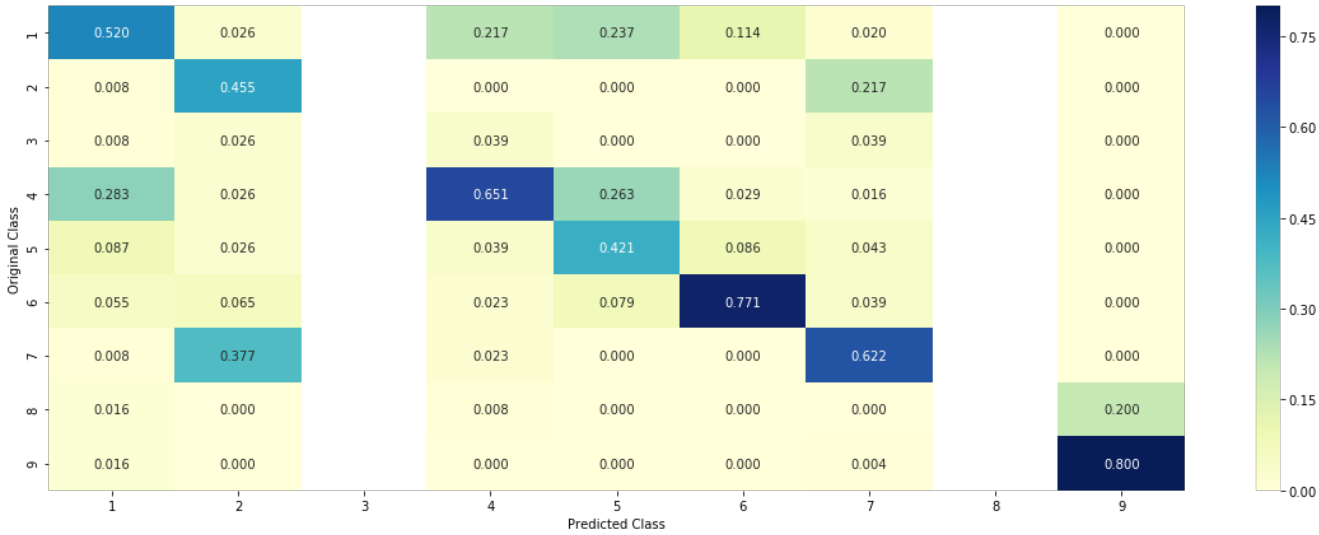
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.34736268072026644
Log loss (CV) on the stacking classifier : 1.13659796719238
Log loss (test) on the stacking classifier : 1.2849463531291183
Number of missclassified point : 0.41353383458646614
-------------------- Confusion matrix --------------------
```

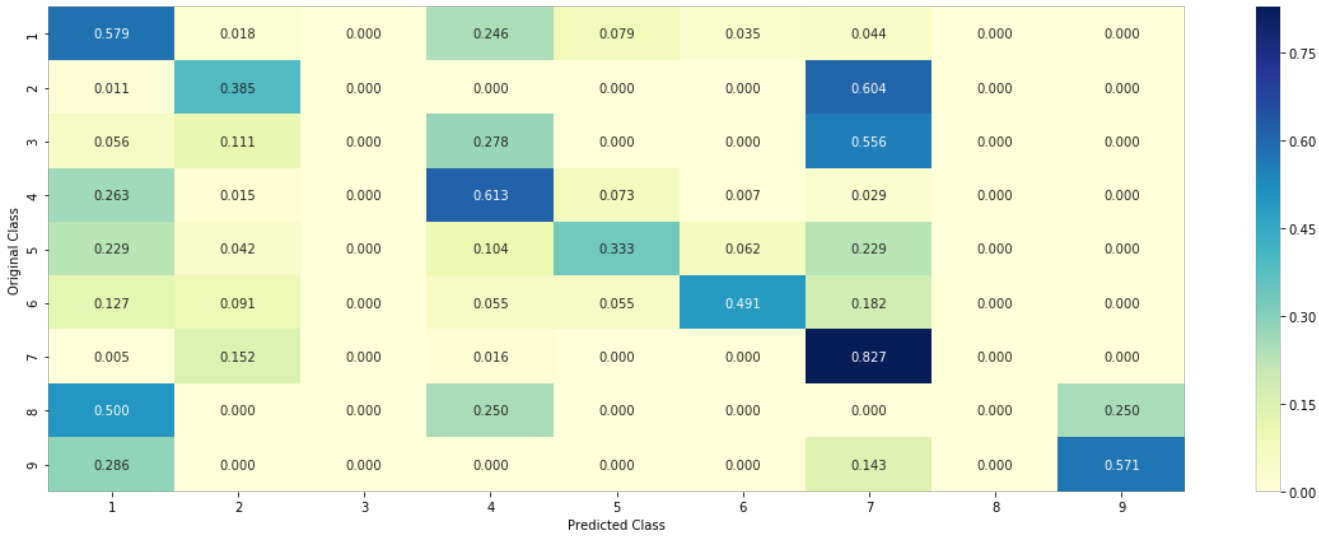




---------- Precision matrix (Columm Sum=1) ----------



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



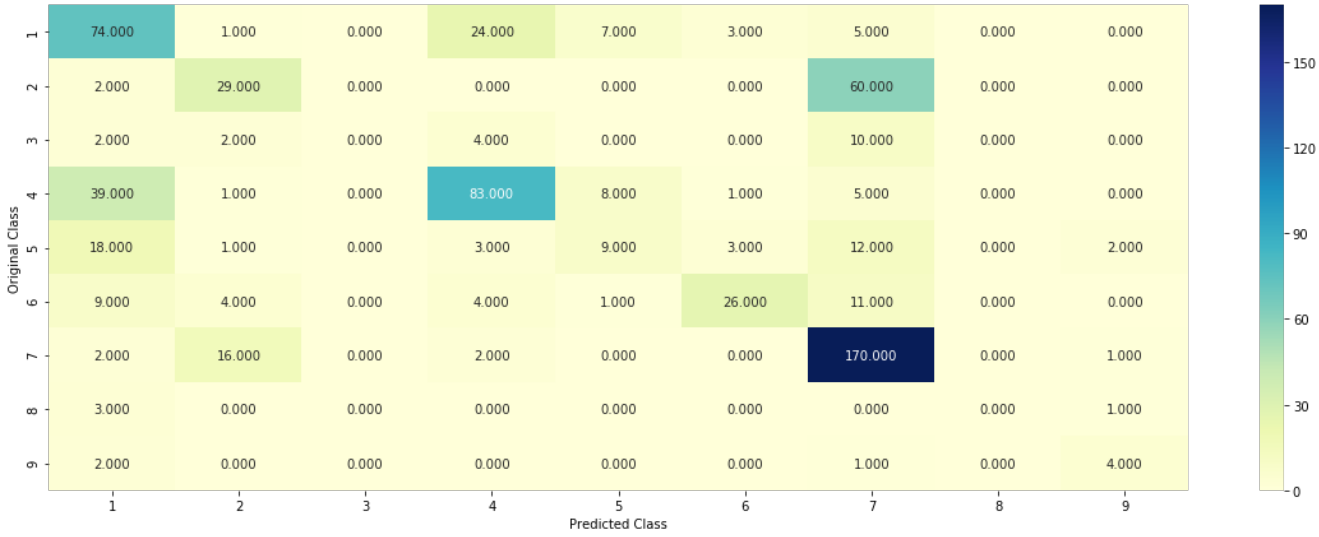
6.7.3 Maximum Voting classifier

In [319]:

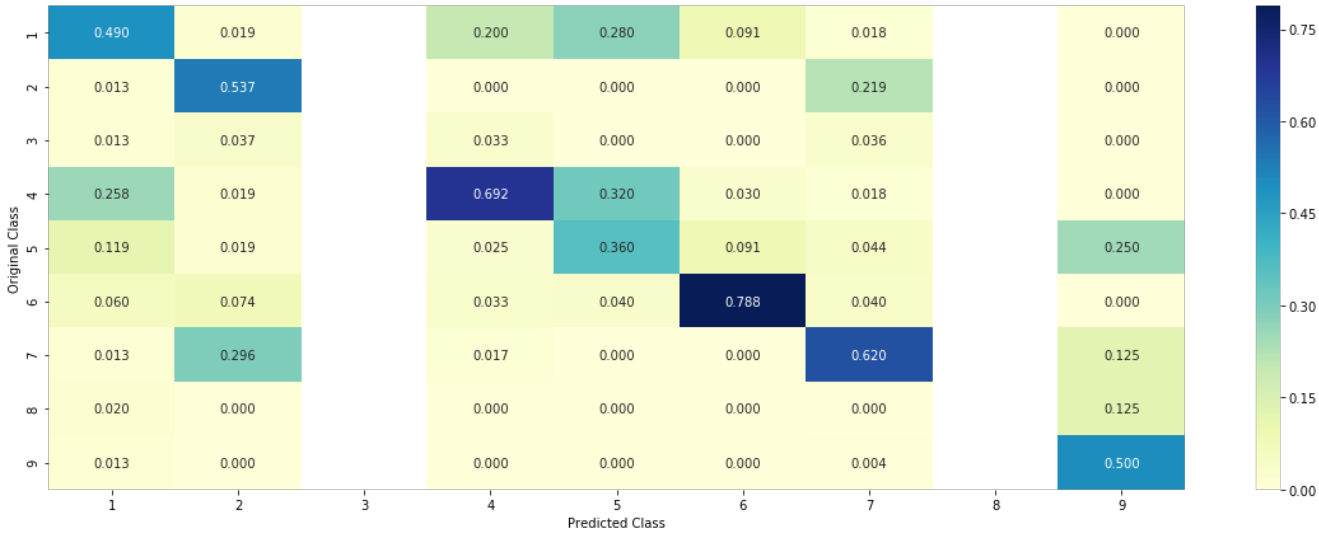
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log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

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

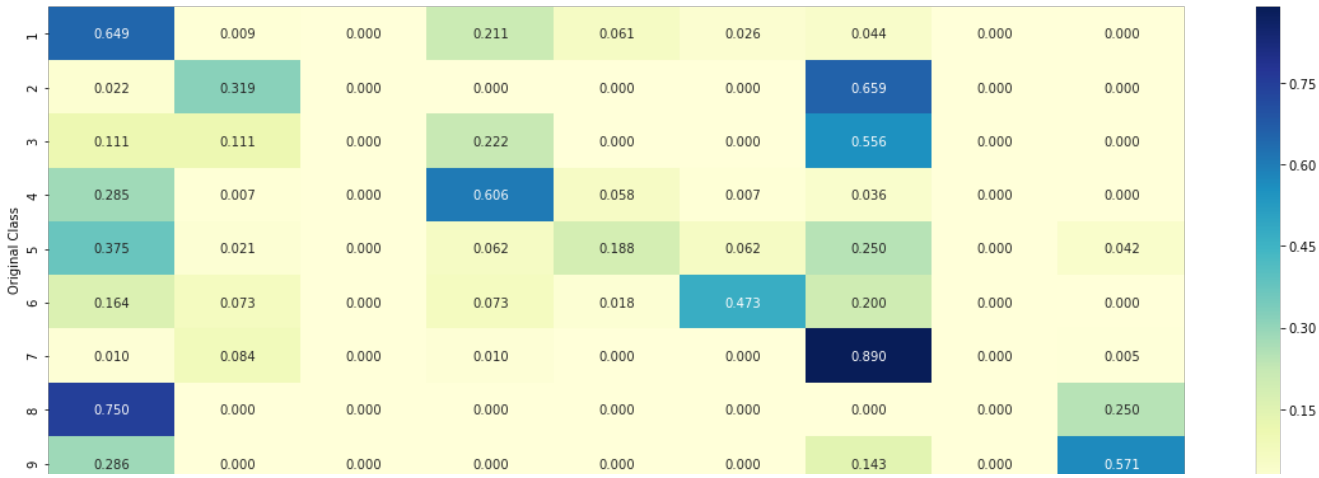
```
Log loss (train) on the VotingClassifier : 0.7985231526435436
Log loss (CV) on the VotingClassifier : 1.1564436087392544
Log loss (test) on the VotingClassifier : 1.2208718844848039
Number of missclassified point : 0.40601503759398494
-------------------- Confusion matrix --------------------
```

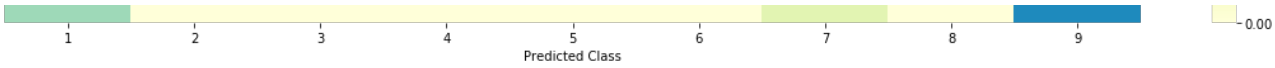



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





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

In [320]:

```
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 featu
res) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 778200
```

In [321]:

```
# 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 [322]:

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

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

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

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

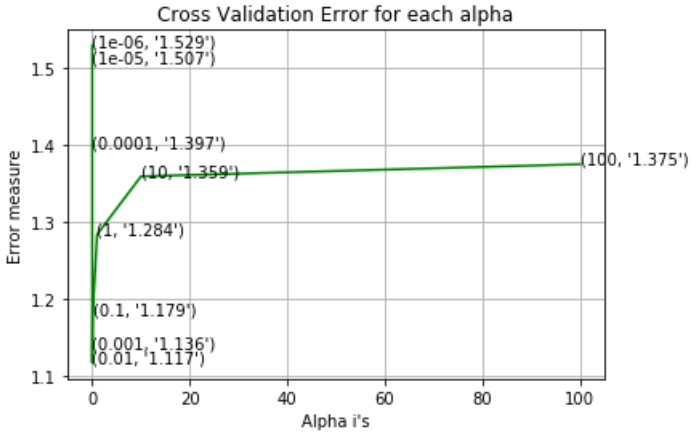
In [323]:

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

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, 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 : 1.5292171959834087
for alpha = 1e-05
Log Loss : 1.5067970468985632
for alpha = 0.0001
Log Loss : 1.3968074242308925
for alpha = 0.001
Log Loss : 1.1363435835549212
for alpha = 0.01
```

```
for alpha = 0.01
Log Loss : 1.116795429493152
for alpha = 0.1
Log Loss : 1.178891008745388
for alpha = 1
Log Loss : 1.2837599928829617
for alpha = 10
Log Loss : 1.3593148220030336
for alpha = 100
Log Loss : 1.375081814283882
```

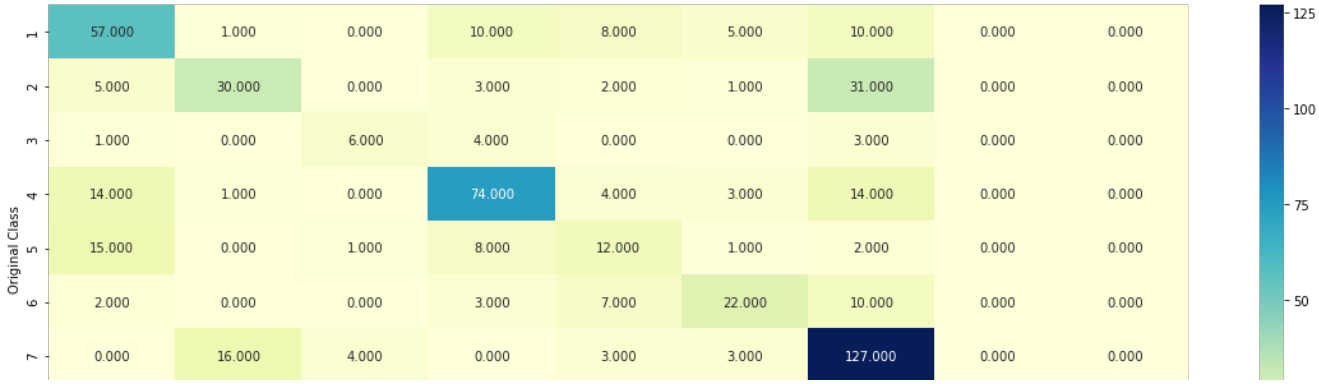


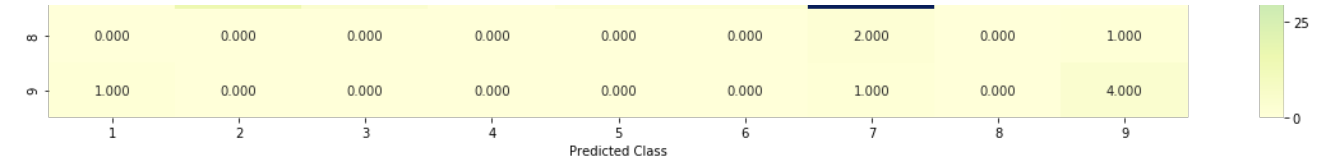

```
For values of best alpha =  0.01 The train log loss is: 0.6904044986493857
For values of best alpha =  0.01 The cross validation log loss is: 1.116795429493152
For values of best alpha =  0.01 The test log loss is: 1.1692159341942823
```

In [324]:

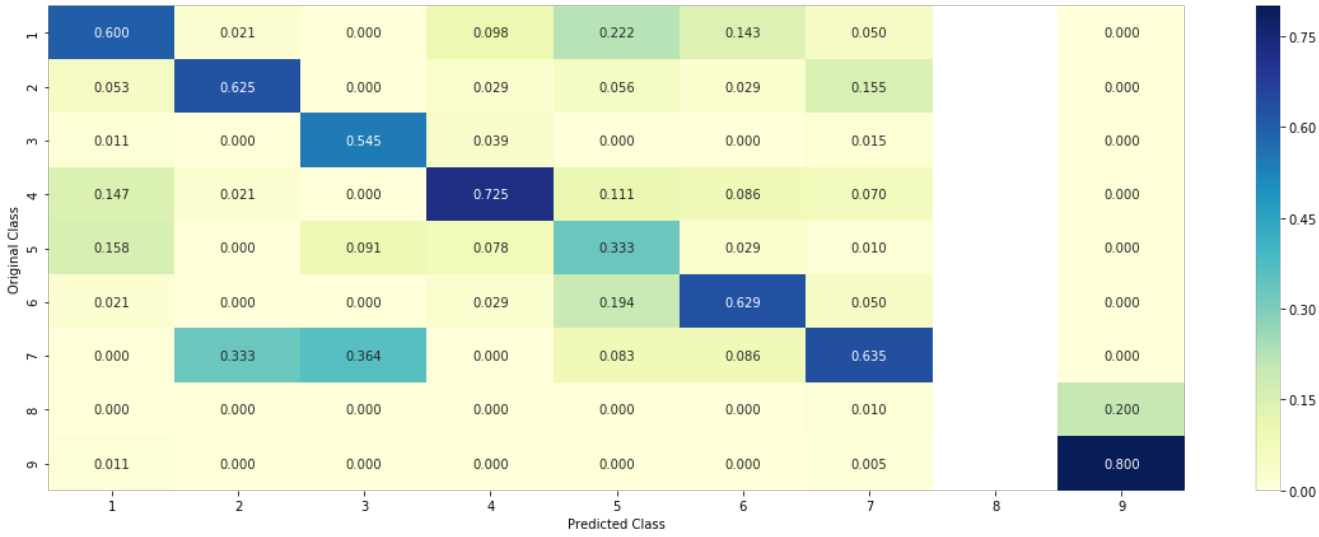
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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

```
Log loss : 1.116795429493152
Number of mis-classified points : 0.37593984962406013
--------------------- Confusion matrix ----------------------
```

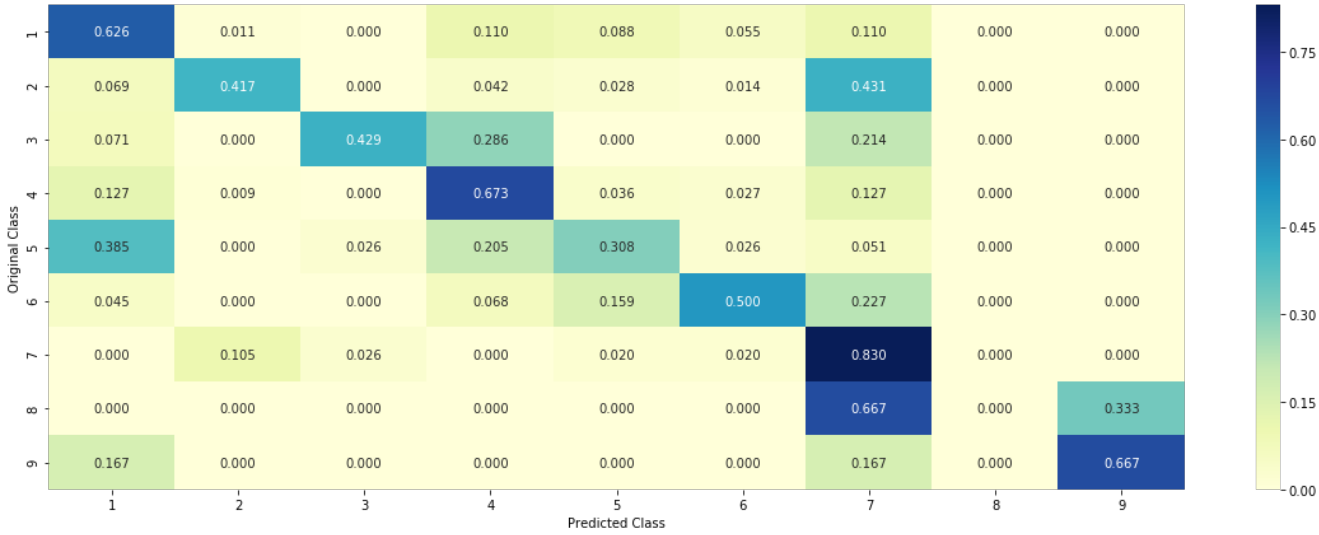




---------- Precision matrix (Columm Sum=1) ----------



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



In [325]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
```

```
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

In [326]:

```
# 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 = CountVectorizer(min_df=3  ,ngram_range=(1, 2))

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

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

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

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

In [327]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1374 0.0989 0.016  0.5058 0.0431 0.0301 0.1559 0.0062 0.0066]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

In [328]:

```
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 : 6
Predicted Class Probabilities: [[0.1916 0.0465 0.0087 0.0272 0.3064 0.3608 0.0439 0.0107 0.0044]]
Actual Class : 6
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

6.9 Feature engineering techniques to reduce the CV and test log-loss to a value less than 1.0

In [329]:

```
train_df.loc[:, 'TextCount']  = train_df["TEXT"].apply(lambda x: len(x.split()))
cv_df.loc[:, 'TextCount']  = cv_df["TEXT"].apply(lambda x: len(x.split()))
test_df.loc[:, 'TextCount']  = test_df["TEXT"].apply(lambda x: len(x.split()))
```

In [330]:

```
train_df.head()
```

Out[330]:

	ID	Gene	Variation	Class	TEXT	Text_count	TextCount
571	571	SMAD3	D408E	4	transforming growth factor tgf activates trans...	5232	5232
2731	2731	BRAF	MKRN1-BRAF_Fusion	2	pilocytic astrocytomas pa common pediatric bra...	8623	8623
1727	1727	APC	A290T	1	familial adenomatous polyposis autosomal domin...	3623	3623
2498	2498	BRCA1	R496C	5	significant proportion inherited breast cancer...	7344	7344
1448	1448	SPOP	F102C	4	largest e3 ligase subfamily cul3 binds btb dom...	16838	16838

In [331]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features = 10000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 10000
```

In [332]:

```
# 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
```

```
# 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 [333]:

```
import scipy
train_text_count= scipy.sparse.csr_matrix(train_df["Text_count"])
train_text_count = normalize(train_text_count, axis = 0)

cv_text_count= scipy.sparse.csr_matrix(cv_df["Text_count"])
cv_text_count = normalize(cv_text_count, axis = 0)

test_text_count= scipy.sparse.csr_matrix(test_df["Text_count"])
test_text_count = normalize(test_text_count, axis = 0)

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,train_text_count.T )).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding,
test_text_count.T)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding, cv_text_count.
T)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [334]:

```
train_text_count= np.array(train_df["Text_count"])
train_text_count = normalize(train_text_count[:, None], axis = 0)

cv_text_count= np.array(cv_df["Text_count"])
cv_text_count = normalize(cv_text_count[:, None], axis = 0)

test_text_count= np.array(test_df["Text_count"])
test_text_count = normalize(test_text_count[:, None], axis = 0)

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,train_text_count))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding,
test_text_count))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding,
cv_text_feature_responseCoding,cv_text_count))
```

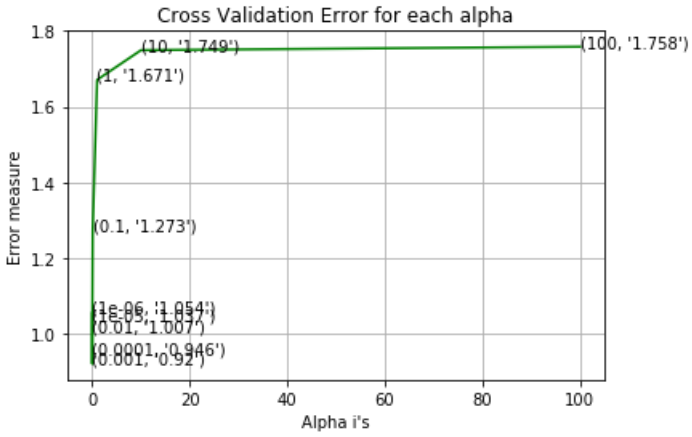
In [335]:

```
# 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)
```

```
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, 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 : 1.0544064218880855
for alpha = 1e-05
Log Loss : 1.0371616159761359
for alpha = 0.0001
Log Loss : 0.9463515923907313
for alpha = 0.001
Log Loss : 0.9195918733283946
```

```
for alpha = 0.01
Log Loss : 1.0069036186757097
for alpha = 0.1
Log Loss : 1.2727555695842818
for alpha = 1
Log Loss : 1.6707275695020518
for alpha = 10
Log Loss : 1.7491090302091155
for alpha = 100
Log Loss : 1.758250398881556
```

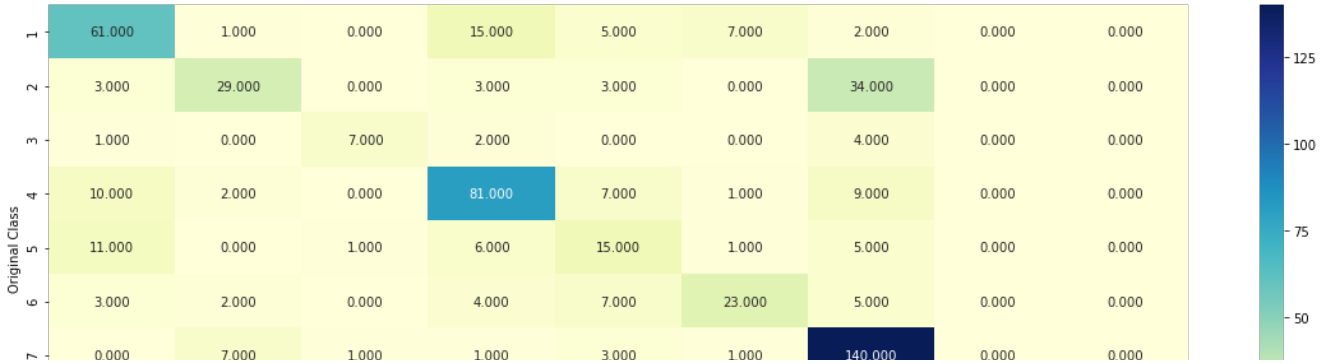


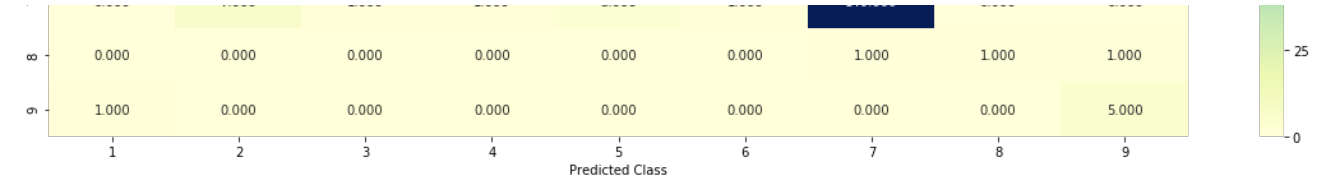

```
For values of best alpha =  0.001 The train log loss is: 0.5314114990736387
For values of best alpha =  0.001 The cross validation log loss is: 0.9195918733283946
For values of best alpha =  0.001 The test log loss is: 0.9967081292034086
```

In [336]:

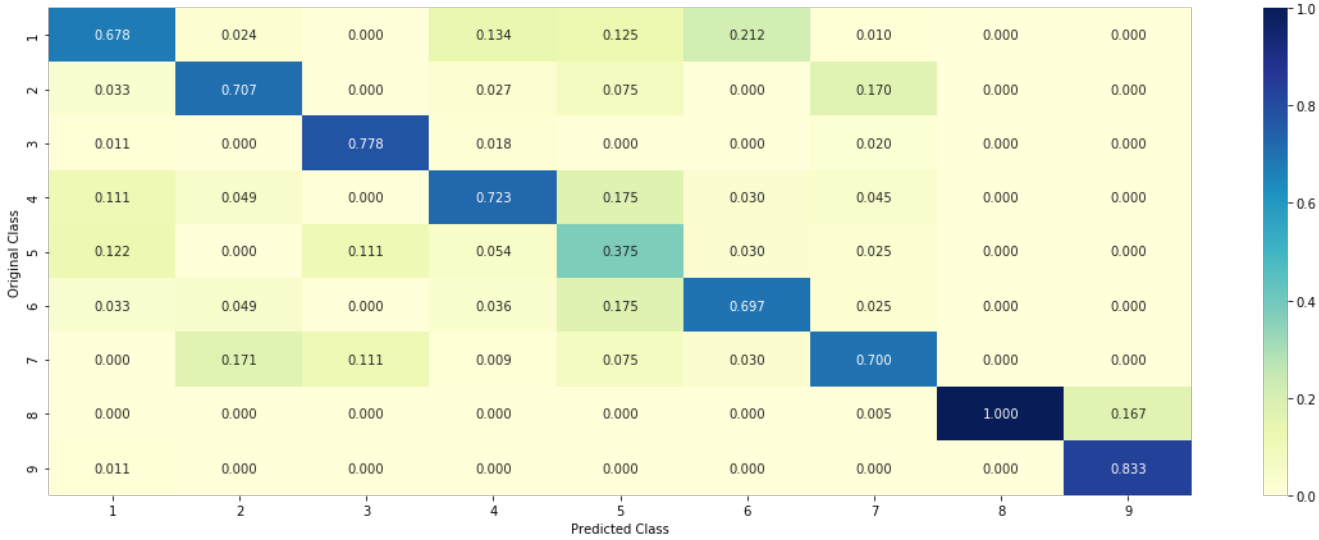
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# 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 : 0.9195918733283946
Number of mis-classified points : 0.31954887218045114
-------------------- Confusion matrix --------------------
```

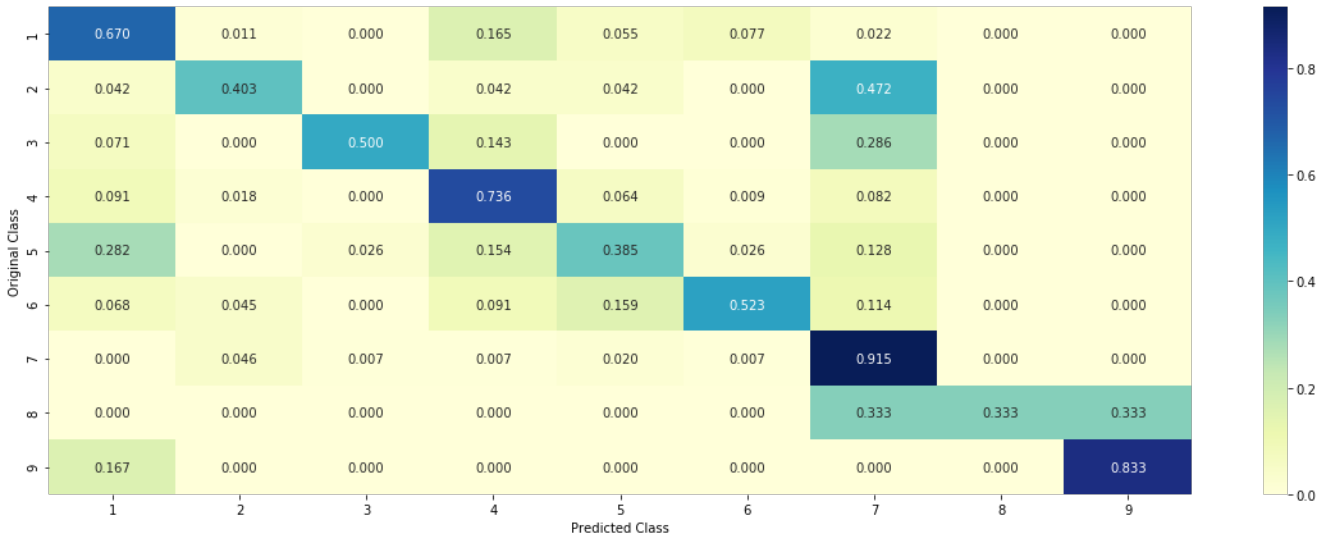





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



In [339]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features = 10000)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
```

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

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

In [340]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2415 0.053  0.0041 0.602  0.0167 0.0028 0.066  0.0059 0.0081]]
Actual Class : 4
--------------------------------------------------
32 Text feature [suppressor] present in test data point [True]
61 Text feature [symbols] present in test data point [True]
72 Text feature [microscopy] present in test data point [True]
74 Text feature [thermo] present in test data point [True]
77 Text feature [homozygous] present in test data point [True]
99 Text feature [tagged] present in test data point [True]
109 Text feature [v4] present in test data point [True]
137 Text feature [leukocyte] present in test data point [True]
139 Text feature [aurora] present in test data point [True]
144 Text feature [germline] present in test data point [True]
175 Text feature [scientific] present in test data point [True]
178 Text feature [missense] present in test data point [True]
185 Text feature [fibroblasts] present in test data point [True]
197 Text feature [hematopoiesis] present in test data point [True]
216 Text feature [families] present in test data point [True]
223 Text feature [localization] present in test data point [True]
239 Text feature [creates] present in test data point [True]
272 Text feature [heterozygotes] present in test data point [True]
312 Text feature [knockout] present in test data point [True]
337 Text feature [cloned] present in test data point [True]
340 Text feature [protein] present in test data point [True]
348 Text feature [loss] present in test data point [True]
356 Text feature [boundaries] present in test data point [True]
363 Text feature [defect] present in test data point [True]
377 Text feature [stability] present in test data point [True]
378 Text feature [truncated] present in test data point [True]
391 Text feature [647] present in test data point [True]
410 Text feature [cytoplasm] present in test data point [True]
419 Text feature [spectral] present in test data point [True]
430 Text feature [premature] present in test data point [True]
435 Text feature [frameshift] present in test data point [True]
450 Text feature [heterozygous] present in test data point [True]
```

```
476 Text feature [junctions] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

In [341]:

```
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 : 6
Predicted Class Probabilities: [[0.1279 0.018  0.0081 0.0329 0.3507 0.4291 0.0264 0.0044 0.0025]]
Actual Class : 6
--------------------------------------------------
146 Text feature [339] present in test data point [True]
188 Text feature [blue] present in test data point [True]
195 Text feature [pgex] present in test data point [True]
198 Text feature [wildtype] present in test data point [True]
269 Text feature [resistance] present in test data point [True]
276 Text feature [380] present in test data point [True]
277 Text feature [polar] present in test data point [True]
293 Text feature [characteristic] present in test data point [True]
305 Text feature [excitation] present in test data point [True]
322 Text feature [substitutions] present in test data point [True]
330 Text feature [tube] present in test data point [True]
338 Text feature [helix] present in test data point [True]
345 Text feature [helices] present in test data point [True]
358 Text feature [pmol] present in test data point [True]
403 Text feature [wavelength] present in test data point [True]
409 Text feature [library] present in test data point [True]
410 Text feature [nm] present in test data point [True]
420 Text feature [340] present in test data point [True]
442 Text feature [showing] present in test data point [True]
447 Text feature [nmol] present in test data point [True]
454 Text feature [conformation] present in test data point [True]
456 Text feature [142] present in test data point [True]
472 Text feature [emission] present in test data point [True]
477 Text feature [flp] present in test data point [True]
478 Text feature [sex] present in test data point [True]
479 Text feature [g248v] present in test data point [True]
480 Text feature [a159t] present in test data point [True]
481 Text feature [ins58l] present in test data point [True]
484 Text feature [individuals] present in test data point [True]
487 Text feature [species] present in test data point [True]
493 Text feature [af1] present in test data point [True]
498 Text feature [direct] present in test data point [True]
Out of the top  500  features  32 are present in query point
```

Conclusion

Using tfidf vectorizer & top 1000 words based of tf-idf values

In [343]:

```
from prettytable import PrettyTable
x = PrettyTable()

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

x.add_row(["Naive Bayes","0.462","1.142","1.229","34.39"])
x.add_row(["KNN","0.605","0.967","1.086","33.27"])
x.add_row(["LR With Class balancing","0.4012","0.9814","1.0338","32.51"])
x.add_row(["LR Without Class balancing","0.3885","1.0071","1.0568","33.27"])
```

```
x.add_row(["Linear SVM","0.3176","1.0353","1.0744","33.83"])
x.add_row(["RF With One hot Encoding","0.8654","1.1793","1.207","40.97"])
x.add_row(["RF With Response Coding","0.0607","1.3120","1.3990","46.42"])
x.add_row(["Stack Models:LR+NB+SVM","0.3473","1.1365","1.2849","41.35"])
x.add_row(["Maximum Voting classifier","0.7985","1.1564","1.2208","40.60"])

print(x)
```

```
+----------------------------+----------------+-------------+---------------+----------------------
|           Model            | Train Log Loss | CV Log Loss | Test Log Loss | % Misclassified Point
s |
+----------------------------+----------------+-------------+---------------+----------------------
|        Naive Bayes         |     0.462      |    1.142    |     1.229     |          34.39
|
|            KNN             |     0.605      |    0.967    |     1.086     |          33.27
|
|  LR With Class balancing   |     0.4012     |    0.9814   |     1.0338    |          32.51
|
| LR Without Class balancing |     0.3885     |    1.0071   |     1.0568    |          33.27
|
|         Linear SVM         |     0.3176     |    1.0353   |     1.0744    |          33.83
|
|  RF With One hot Encoding  |     0.8654     |    1.1793   |     1.207     |          40.97
|
|  RF With Response Coding   |     0.0607     |    1.3120   |     1.3990    |          46.42
|
|   Stack Models:LR+NB+SVM   |     0.3473     |    1.1365   |     1.2849    |          41.35
|
| Maximum Voting classifier  |     0.7985     |    1.1564   |     1.2208    |          40.60
|
+----------------------------+----------------+-------------+---------------+----------------------
```

With Logistic regression (including both unigrams and bigrams)

In [344]:

```
x = PrettyTable()

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

x.add_row(["LR","0.6904","1.1167","1.169","37.59"])

print(x)
```

```
+-------+----------------+-------------+---------------+-------------------------+
| Model | Train Log Loss | CV Log Loss | Test Log Loss | % Misclassified Points |
+-------+----------------+-------------+---------------+-------------------------+
|   LR  |     0.6904     |    1.1167   |     1.169     |          37.59          |
+-------+----------------+-------------+---------------+-------------------------+
```

With Feature Engineering

In [346]:

```
x = PrettyTable()

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

x.add_row(["LR with Feature Engg.","0.5314","0.9195","0.9967","31.95"])

print(x)
```

```
+-----------------------+----------------+-------------+---------------+-------------------------+
|         Model         | Train Log Loss | CV Log Loss | Test Log Loss | % Misclassified Points |
+-----------------------+----------------+-------------+---------------+-------------------------+
| LR with Feature Engg. |     0.5314     |    0.9195   |     0.9967    |          31.95          |
+-----------------------+----------------+-------------+---------------+-------------------------+
```