Personalized cancer diagnosis

OBJECTIVE :- Instead of using all the words in the dataset , use only top 1000 words based of tf-idf values

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 [2]:

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


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 [3]:

```
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[3]:

	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 [4]:

```
# 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[4]:

	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 [5]:

```
# 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 [6]:

```
#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 : 188.69429436629434 seconds
```

In [7]:

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

Out[7]:

	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 [8]:

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

Out[8]:

	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 [9]:

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

In [10]:

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

Out[10]:

	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 [11]:

```
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 [12]:

```
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])
```

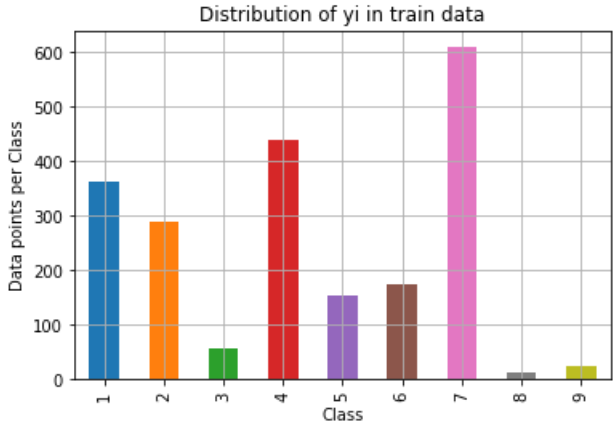
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

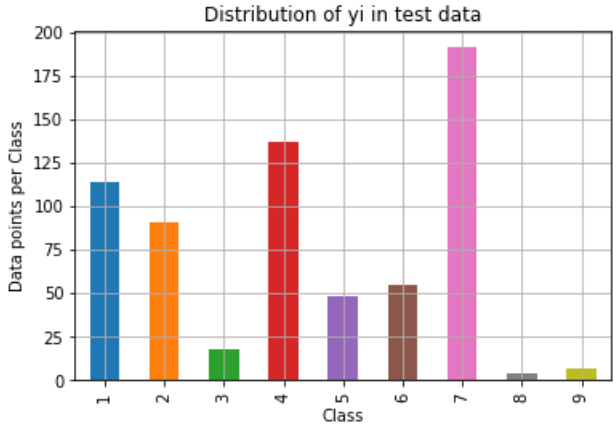
In [13]:

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

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

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


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

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

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

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

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

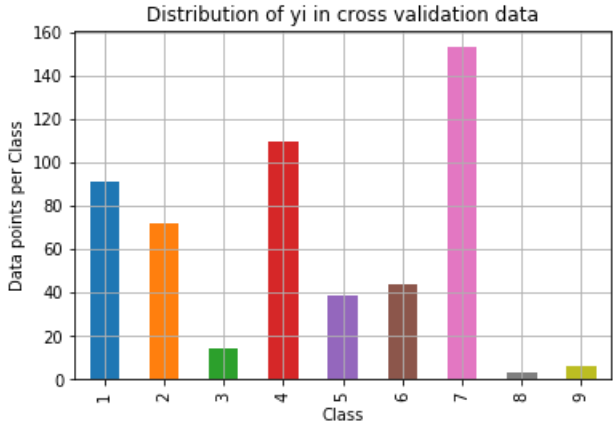



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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




```
Number of data points in class 7 : 191 ( 28.722 %)
```

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




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

3.2 Prediction using a 'Random' Model

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

In [14]:

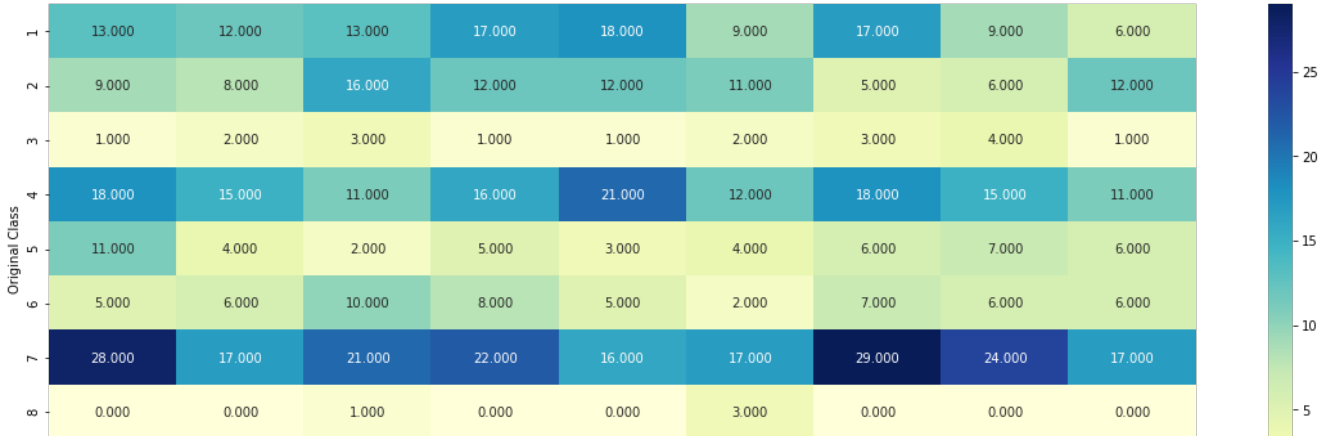
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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]
```

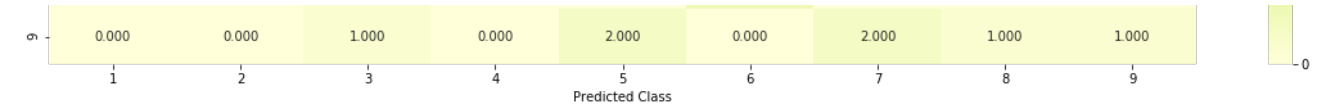
```
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    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 [15]:

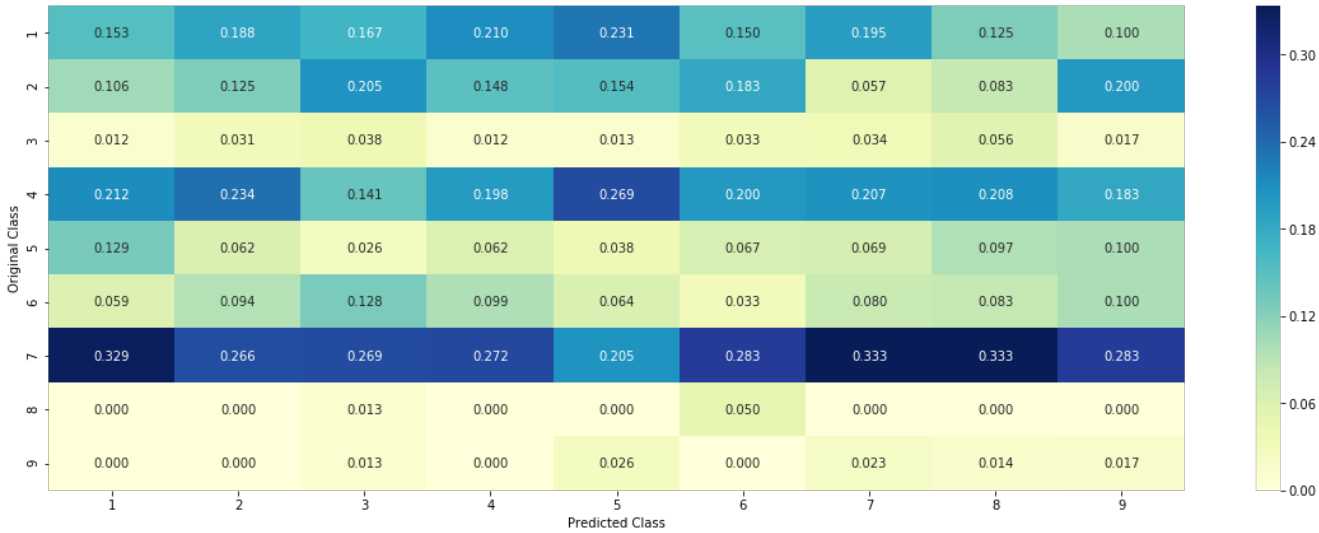
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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 2.5253735680755676
Log loss on Test Data using Random Model 2.442673224476882
-------------------- Confusion matrix --------------------
```

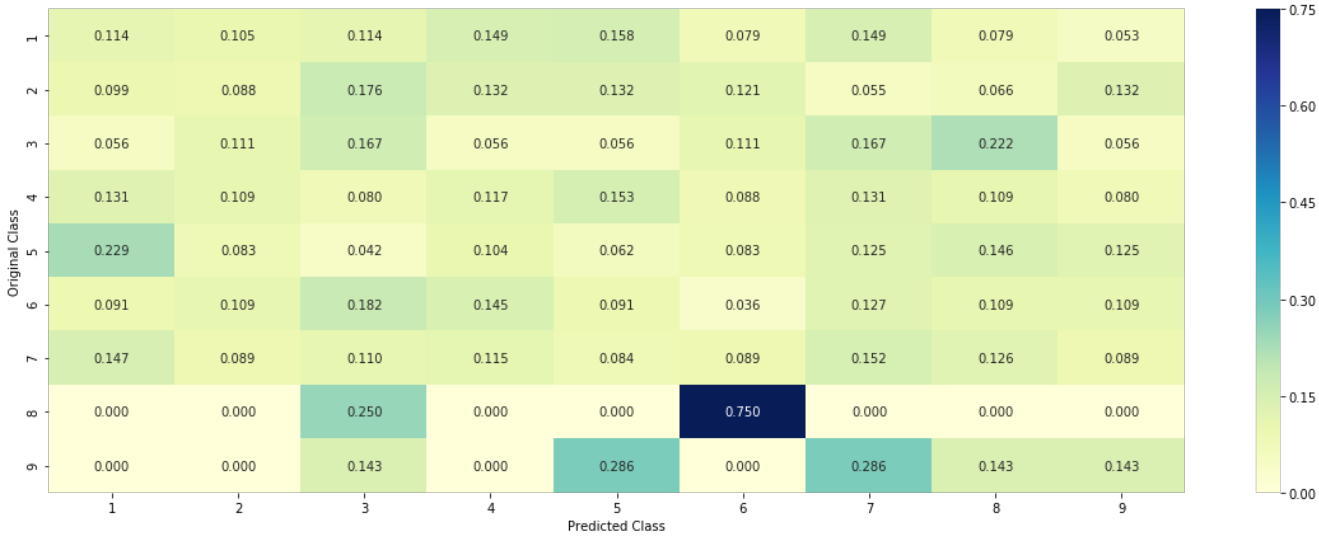





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



3.3 Univariate Analysis

In [16]:

```
# 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, 0.07547169811320754, 0.06289308176100629, 0.06918238993710691, 0.062893081
761006289, 0.06289308176100629],
    #       '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 [17]:

```
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 : 234
BRCA1     166
TP53       97
EGFR       83
BRCA2      80
PTEN       77
KIT        69
BRAF       65
ALK        43
ERBB2      43
PIK3CA     40
Name: Gene, dtype: int64
```

In [18]:

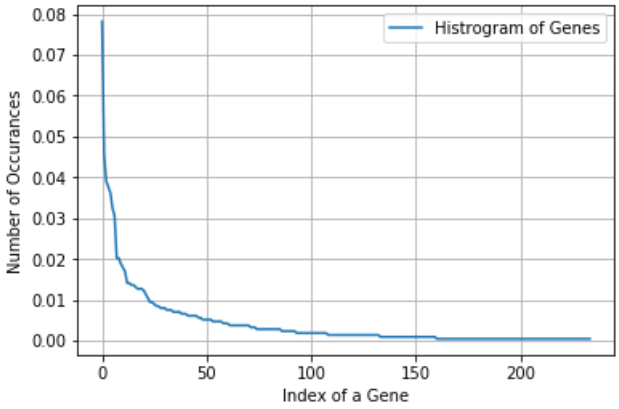
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, an
d they are distibuted as follows",)
```

```
Ans: There are 234 different categories of genes in the train data, and they are distibuted as fol
lows
```

In [19]:

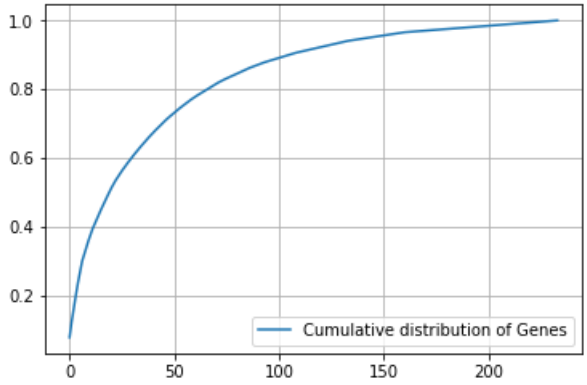
```
s = sum(unique_genes.values);
```

```
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 [20]:

```
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 [21]:

```
#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 [22]:

```
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 [23]:

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

In [24]:

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

Out[24]:

```
1350      AKT1
1136       MET
317       ROS1
2003    MAP2K1
1264    PIK3R1
Name: Gene, dtype: object
```

In [25]:

```
gene_vectorizer.get_feature_names()
```

Out[25]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
```

```
'cdk12',
'cdk4',
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
```

```
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'met',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk3',
'nup93',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
```

```
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'rybp',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'yap1']
```

In [26]:

```
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, 234)
```

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

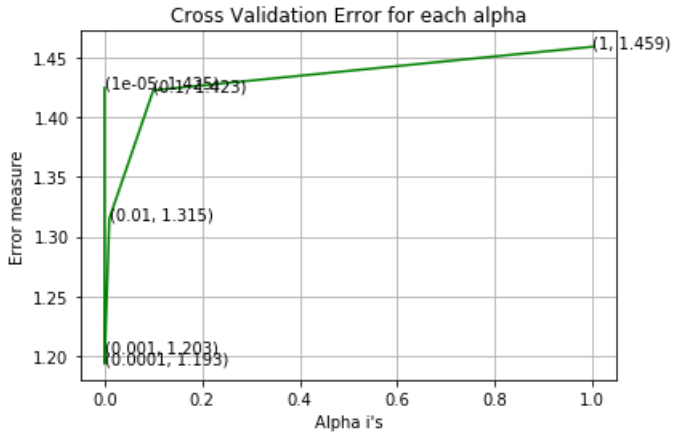
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [27]:

```
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: 1.4247930173820744
For values of alpha =  0.0001 The log loss is: 1.1933400016689775
For values of alpha =  0.001 The log loss is: 1.2034981313107145
For values of alpha =  0.01 The log loss is: 1.3152232062582576
For values of alpha =  0.1 The log loss is: 1.4227287227782988
For values of alpha =  1 The log loss is: 1.459254102352852
```




```
For values of best alpha =  0.0001 The train log loss is: 1.051726402608188
For values of best alpha =  0.0001 The cross validation log loss is: 1.1933400016689775
For values of best alpha =  0.0001 The test log loss is: 1.240686003806901
```

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 [28]:

```
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  234  genes in train dataset?
Ans
1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 513 out of  532 : 96.42857142857143
```

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 [29]:

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

```
Number of Unique Variations : 1931
Truncating_Mutations    61
Amplification           46
Deletion                43
Fusions                 19
Overexpression           5
G12V                     4
E17K                     3
T58I                     3
Q61R                     2
R841K                    2
Name: Variation, dtype: int64
```

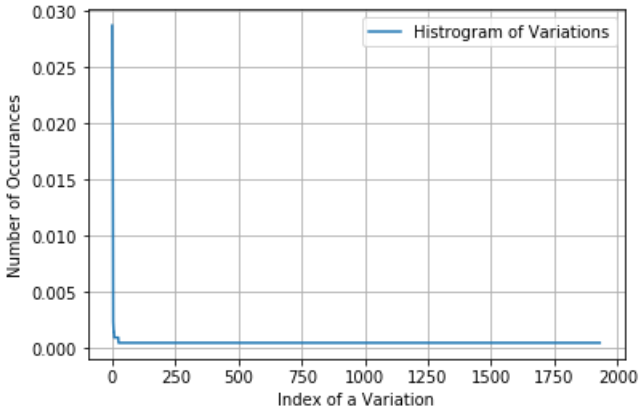
In [30]:

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

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

In [31]:

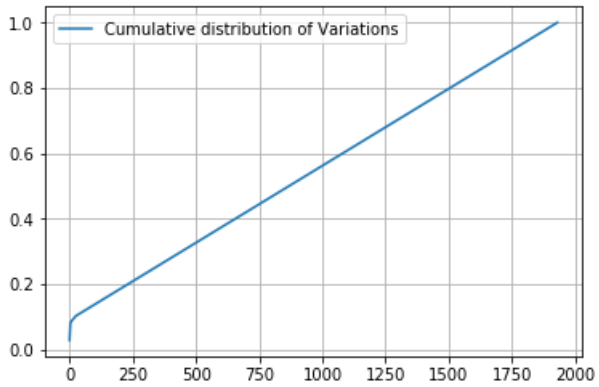
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [32]:

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

```
[0.0287194  0.05037665 0.07062147 ... 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 [33]:

```
# 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 [34]:

```
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 [35]:

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

In [36]:

```
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, 1963)
```

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

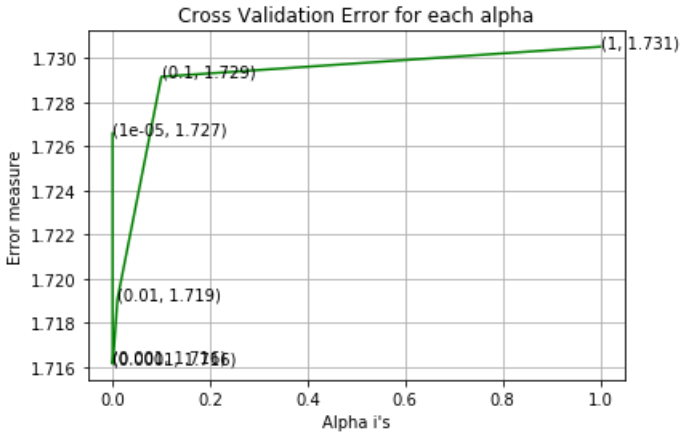
Let's build a model just like the earlier!

In [37]:

```
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: 1.7265942766950513
For values of alpha =  0.0001 The log loss is: 1.7161338416146095
For values of alpha =  0.001 The log loss is: 1.7162398331693225
For values of alpha =  0.01 The log loss is: 1.7190776069720586
For values of alpha =  0.1 The log loss is: 1.7291678794186136
For values of alpha =  1 The log loss is: 1.7305273828736634
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7623494055072086
For values of best alpha =  0.0001 The cross validation log loss is: 1.7161338416146095
For values of best alpha =  0.0001 The test log loss is: 1.7056243042857773
```

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 [38]:

```
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  1931  genes in test and cross validation data
sets?
Ans
1. In test data 78 out of 665 : 11.729323308270677
2. In cross validation data 50 out of  532 : 9.398496240601503
```

3.2.3 Univariate Analysis on Text Feature

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

In [39]:

```
# 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 [40]:

```
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 [41]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(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 [42]:

```
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 [43]:

```
#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 [44]:

```
# 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 [45]:

```
# 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 [46]:

```
#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 [47]:

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

```
Counter({257.39978542736685: 1, 177.28243666894508: 1, 136.46441976200416: 1, 129.65084950918492:
1, 128.61560565538494: 1, 118.42671449446124: 1, 118.29917607283221: 1, 118.07557206431657: 1, 112
.61649546151116: 1, 108.55119804748861: 1, 108.253654520695: 1, 91.85422120160074: 1,
89.16984204111024: 1, 80.94426301029567: 1, 79.16714907498253: 1, 78.70312444258063: 1,
78.25320374272445: 1, 78.16502693552832: 1, 78.12723396141564: 1, 76.63589120840464: 1,
73.2313730843616: 1, 72.981895571711176: 1, 71.3918906843217: 1, 69.56407149178872: 1,
68.86713271746049: 1, 68.26368471041732: 1, 66.50259034891076: 1, 66.33014569817787: 1,
64.45382491763955: 1, 64.24431571550953: 1, 64.16394370534566: 1, 63.261067845348094: 1,
61.54553674473806: 1, 60.79632947295299: 1, 60.21616510017099: 1, 57.42086316876914: 1,
57.04368334073718: 1, 56.12972855582854: 1, 56.120701331945234: 1, 52.9754111463178 75: 1,
51.39893777203324: 1, 50.72523514019548: 1, 48.99266527861622: 1, 48.04315570227214: 1,
48.02788324728651: 1, 47.98097411075397: 1, 47.36915431340152: 1, 46.58771587688782: 1,
45.77076309596089: 1, 45.550323395064446: 1, 45.07519445966123: 1, 44.61654578757668: 1,
44.43077693259986: 1, 43.398638133960155: 1, 43.30017781302526: 1, 43.20378982158019: 1,
43.102690348089276: 1, 42.72164365743028: 1, 42.42482770700917: 1, 42.40805336168335: 1, 42.0921999
3321098: 1, 41.514313709722266: 1, 41.275320713056466: 1, 41.014822314886885: 1,
40.99433111068719: 1, 40.76727808004724: 1, 40.49492246512379: 1, 40.45819924485336: 1,
39.69247596326572: 1, 39.34564465488437: 1, 39.26242015153517: 1, 39.023094574167935: 1,
37.804380239813476: 1, 37.41667571321772: 1, 37.35077405768358: 1, 36.953798574116945: 1,
36.941828985013885: 1, 36.51021629561486: 1, 36.13882885856671: 1, 36.09777529811125: 1, 35.9495278
8385019: 1, 35.86753454771733: 1, 35.372921603155845: 1, 34.790698982643484: 1, 34.3816011664831:
1, 34.38153893061753: 1, 34.05083672287259: 1, 33.90464453233115: 1, 33.87363875334398: 1,
33.85019856501517: 1, 33.58294206590704: 1, 33.3544496549618: 1, 33.34084906520516: 1,
33.32733640600062: 1, 32.9686436686472: 1, 32.615151751680266: 1, 32.61311216985466: 1,
32.34990232653006: 1, 32.12444920796583: 1, 32.08646520992784: 1, 32.047794775596024: 1,
```

32.34990232653006: 1, 32.12444920796583: 1, 32.08646520992784: 1, 32.04779475596024: 1,
32.02273803182875: 1, 31.874737561764 88: 1, 31.771273386074135: 1, 31.770930138006882: 1,
31.51531541616611: 1, 31.360887133301283: 1, 31.223993306267875: 1, 31.19022061115567: 1,
31.141701084092638: 1, 31.072068721557624: 1, 30.865096072816456: 1, 30.796958098109602: 1,
30.519255374030 86: 1, 30.34877532318762: 1, 30.31547046681412: 1, 30.30937178492225: 1,
30.025759192566053: 1, 29.966474949505127: 1, 29.922320393223572: 1, 29.848308644396475: 1,
29.845731669644312: 1, 29.77561701127028: 1, 29.761608339146715: 1, 29.706347380545 99: 1,
29.585131095595955: 1, 29.458066420518 58: 1, 29.063041514787614: 1, 28.973910882741016: 1,
28.65592555844408: 1, 28.631070270768 97: 1, 28.608270671527404: 1, 28.416501399487 13: 1,
28.01168723067217: 1, 27.906418166451903: 1, 27.715606818999458: 1, 27.553129279558 42: 1,
27.4667994008805: 1, 27.375946876661818: 1, 27.15844697235551: 1, 27.137112895858 72: 1,
27.136813167057404: 1, 27.106112198550285: 1, 26.933447664897713: 1, 26.868205428617 89: 1,
26.793400570562415: 1, 26.53648618168 32: 1, 26.48157274475971: 1, 26.435535158942898: 1,
26.194077639430287: 1, 25.837191536138 58: 1, 25.584831024627693: 1, 25.456581191605096: 1,
25.40983600851401: 1, 25.375937458048917: 1, 25.299901098482117: 1, 25.270159663106035: 1,
25.134184287106375: 1, 25.100990928436882: 1, 25.082144533638175: 1, 25.051106609804574: 1,
24.944123828677473: 1, 24.935377714263 68: 1, 24.93355455596227: 1, 24.768820151297433: 1,
24.473074886710936: 1, 24.421482146074034: 1, 24.402149205713876: 1, 24.365850444425952: 1,
24.339783908276182: 1, 24.259785095456287: 1, 24.25224744184153: 1, 24.104303942790 63: 1,
24.046860398349466: 1, 23.959622269647387: 1, 23.937856214118 24: 1, 23.876363113988614: 1,
23.762996450582893: 1, 23.672231142539 69: 1, 23.589402986606004: 1, 23.41140380923 46: 1,
23.149440541495558: 1, 23.088196202275 55: 1, 23.03005648250931: 1, 23.017021998551215: 1,
23.01145629886706: 1, 22.972614536781098: 1, 22.920167039838173: 1, 22.889127686386782: 1,
22.855330565625437: 1, 22.771932110748875: 1, 22.766031567182 87: 1, 22.74794576259184: 1,
22.62160357047764: 1, 22.563093049716958: 1, 22.556883859364223: 1, 22.533940304072622: 1,
22.528483801886534: 1, 22.524213751385417: 1, 22.503140931497 47: 1, 22.466166773038903: 1,
22.40836826798458: 1, 22.382309579414212: 1, 22.352588051879835: 1, 22.301125793020802: 1,
22.132008079853833: 1, 22.11052597928 83: 1, 22.101072871895212: 1, 22.05522587311962: 1,
22.052395852191705: 1, 22.024301779111006: 1, 21.977346040483 81: 1, 21.799781362200974: 1,
21.68205126744081: 1, 21.667419520205637: 1, 21.633144851920427: 1, 21.629275035682 92: 1,
21.58284839559532: 1, 21.510504413782 76: 1, 21.501574120189254: 1, 21.477162121738232: 1,
21.458143212581923: 1, 21.380339018922 18: 1, 21.34315471474 73: 1, 21.323582132776007: 1,
21.313873267751656: 1, 21.130324928997346: 1, 21.099606832395327: 1, 21.096884931584874: 1,
21.03466398626841: 1, 21.031955795700874: 1, 20.998321450649 76: 1, 20.99095480487879: 1,
20.792665630675106: 1, 20.776026517995515: 1, 20.736865813434047: 1, 20.732749604372643: 1,
20.63616782347723: 1, 20.6355102649887: 1, 20.586902583099228: 1, 20.533896094125 97: 1,
20.40067789782703: 1, 20.387470308679365: 1, 20.289847101795523: 1, 20.271852676568347: 1,
20.20147755799322: 1, 20.165118794238904: 1, 20.154166923502995: 1, 20.130369181981383: 1,
20.102643817451344: 1, 20.079771520104 45: 1, 19.722163429307404: 1, 19.663943149165675: 1,
19.654207426304378: 1, 19.648952118564505: 1, 19.568540388760393: 1, 19.542650374331 49: 1,
19.519233077273395: 1, 19.514858190996936: 1, 19.489869768468502: 1, 19.384849470843157: 1,
19.384403317796043: 1, 19.297812812587555: 1, 19.282566740765994: 1, 19.259402434184 42: 1,
19.254015893704675: 1, 19.243973419245354: 1, 19.233632924981 06: 1, 19.228862351694143: 1,
19.22674431942513: 1, 19.202785732025 83: 1, 19.12139629785588: 1, 19.114449164130352: 1,
19.03435657074854: 1, 18.929383102729883: 1, 18.841105278221097: 1, 18.757369238218352: 1,
18.751485953831367: 1, 18.730044289517146: 1, 18.706119563337268: 1, 18.627542701672535: 1,
18.545394772292962: 1, 18.521326585667044: 1, 18.496225296456593: 1, 18.466342986039 8: 1,
18.448597301507075: 1, 18.369234064305 06: 1, 18.361891204788375: 1, 18.257817313365774: 1,
18.234170135292803: 1, 18.192433367819245: 1, 18.150182607717 05: 1, 18.144074623688756: 1,
18.136767937533545: 1, 18.130746890476196: 1, 18.102631177628 03: 1, 18.094012102751464: 1,
18.03620413579843: 1, 18.0342024603885: 1, 18.026801429028588: 1, 18.000316356538 17: 1,
17.99387180793755: 1, 17.953082335416564: 1, 17.951687142385254: 1, 17.933805437286 79: 1,
17.907893882232926: 1, 17.907139695882 26: 1, 17.895860497526378: 1, 17.883116871645466: 1,
17.88221217183938: 1, 17.851355532210 65: 1, 17.825775814854502: 1, 17.759996892192095: 1,
17.731768564791174: 1, 17.709601945236376: 1, 17.655743622039 04: 1, 17.648263513757044: 1,
17.594967741349283: 1, 17.552606438346928: 1, 17.542456036171 93: 1, 17.530478917709033: 1,
17.52000037734302: 1, 17.473705686175002: 1, 17.464072913306726: 1, 17.449231847330534: 1,
17.404280164012 89: 1, 17.386402183029727: 1, 17.383847463748637: 1, 17.382343614821 08: 1,
17.35149225629359: 1, 17.350756522853917: 1, 17.311090687279 19: 1, 17.306566884040002: 1,
17.29685644685483: 1, 17.225449180857 52: 1, 17.16877891985953: 1, 17.153559156545676: 1,
17.146497964523082: 1, 17.142639677550434: 1, 17.130311797469535: 1, 17.116639656003 03: 1,
17.116137521262 09: 1, 17.075538599388093: 1, 17.029276842842496: 1, 16.989996528186026: 1,
16.976437144728482: 1, 16.946097695778 27: 1, 16.93046405431235: 1, 16.920650797831616: 1,
16.916809458109817: 1, 16.916242928727772: 1, 16.915889540594502: 1, 16.913234668261456: 1,
16.911096696765284: 1, 16.820859794670 25: 1, 16.768623467893978: 1, 16.728174284806055: 1,
16.69589870592976: 1, 16.651942393810824: 1, 16.449293752428314: 1, 16.447277141072107: 1,
16.44092160168928: 1, 16.432121244112 64: 1, 16.420410293186865: 1, 16.346331562927038: 1,
16.275167711965835: 1, 16.264090209814 18: 1, 16.19572906832537: 1, 16.18991256572086: 1, 16.181658(
65088573: 1, 16.172116584634544: 1, 16.11400128425111: 1, 16.10052228374155: 1,
16.066063230461815: 1, 16.064960711781 24: 1, 16.05813277594264: 1, 16.017021957668444: 1,
15.970972114146832: 1, 15.925098123176 05: 1, 15.893128194196496: 1, 15.880388376601696: 1,
15.815601330124636: 1, 15.809318408124483: 1, 15.808131323688732: 1, 15.807964838577233: 1,
15.80356000545457: 1, 15.783820782101007: 1, 15.721073217720683: 1, 15.697212913079444: 1,
15.691769365647856: 1, 15.682315846828331: 1, 15.653457850807 59: 1, 15.652284113063994: 1,
15.650143951393993: 1, 15.638730838430947: 1, 15.533225043996977: 1, 15.503561160938554: 1,
15.491508508285158: 1, 15.439971157008962: 1, 15.435313139308 12: 1, 15.358377499110807: 1,
15.348933509701741: 1, 15.331225793851466: 1, 15.317394750205898: 1, 15.314455577160 81: 1,

15.273650956888591: 1, 15.262335986443128: 1, 15.242859226787749: 1, 15.190243461522691: 1,
15.19008994662936: 1, 15.11276714263435: 1, 15.1106393194711195: 1, 15.104425823731015: 1,
15.041767279878334: 1, 15.036960018470891: 1, 15.00430638837809: 1, 14.9992791782130l9: 1,
14.98441054715678: 1, 14.983231050321958: 1, 14.977471921435727: 1, 14.976761204943886: 1,
14.956214362674245: 1, 14.92459900274894: 1, 14.908007664183769: 1, 14.859420898769658: 1,
14.840213577664695: 1, 14.82644097204719: 1, 14.800798011183149: 1, 14.79863679323554: 1,
14.785657757878198: 1, 14.784635827501816: 1, 14.730182125809849: 1, 14.699919629956769: 1,
14.659966445095147: 1, 14.656220764431689: 1, 14.653628459575957: 1, 14.647212662314823: 1,
14.569468760179362: 1, 14.556590413672975: 1, 14.547857199354937: 1, 14.534507516656973: 1,
14.511486096159997: 1, 14.473614374209665: 1, 14.47269227235314: 1, 14.438172977365832: 1,
14.437321092789952: 1, 14.43149884032066: 1, 14.376539925463808: 1, 14.351273478086833: 1,
14.315399281033125: 1, 14.277520666577836: 1, 14.266699595189843: 1, 14.265120115149902: 1,
14.19302316296535: 1, 14.179001102404783: 1, 14.141270772667875: 1, 14.098361529148168: 1,
14.09020824990558: 1, 14.081446385400525: 1, 14.057709802688544: 1, 14.048994853834843: 1,
14.03867191675792: 1, 14.029854108812467: 1, 13.998861499980402: 1, 13.962740340754245: 1,
13.960707343182113: 1, 13.943854638085881: 1, 13.91711322493816: 1, 13.910743422601733: 1,
13.905046954180508: 1, 13.898495931570702: 1, 13.894526612159606: 1, 13.888911462473894: 1,
13.833466136148926: 1, 13.819474624106233: 1, 13.797456099487565: 1, 13.797051212904773: 1,
13.76570689468518: 1, 13.75392010428402: 1, 13.738256108126784: 1, 13.722477577648753: 1,
13.704768201445424: 1, 13.697056697280802: 1, 13.670401626426417: 1, 13.668115125197572: 1,
13.653462688737035: 1, 13.599263114005094: 1, 13.594324880102205: 1, 13.571239518258018: 1,
13.557284136772102: 1, 13.555691093244425: 1, 13.475060900766836: 1, 13.42837443583501: 1,
13.406658310660864: 1, 13.393022231166753: 1, 13.386977793702052: 1, 13.385383725827053: 1,
13.36508846387053: 1, 13.363544527906928: 1, 13.316748730030247: 1, 13.2687424764653: 1,
13.2685589110783: 1, 13.256117968072939: 1, 13.222665695677472: 1, 13.219541955552124: 1,
13.215469461489974: 1, 13.195774896600998: 1, 13.191629361009968: 1, 13.180877044335944: 1,
13.17853604778122: 1, 13.173633355883823: 1, 13.161649246160275: 1, 13.113344450267668: 1,
13.108869674276525: 1, 13.099176574625414: 1, 13.061989752652886: 1, 13.030211558814134: 1,
13.025801568954996: 1, 13.002248133959593: 1, 12.989646459174871: 1, 12.942051875137151: 1,
12.901080879252769: 1, 12.872718326836399: 1, 12.835779997682053: 1, 12.834696810421075: 1,
12.801104883105722: 1, 12.783592091907886: 1, 12.778251332186766: 1, 12.746175776994962: 1,
12.718461577476827: 1, 12.694765325022619: 1, 12.689318160431363: 1, 12.663514715670576: 1,
12.657993615708968: 1, 12.653203141612186: 1, 12.63353265236407: 1, 12.603050884615799: 1,
12.588912407831899: 1, 12.57444817649283: 1, 12.5721238467593: 1, 12.541942923893131: 1,
12.504904462775977: 1, 12.49000236963212: 1, 12.466404243436887: 1, 12.454667998529136: 1,
12.43706902042105: 1, 12.428887739427468: 1, 12.39664860408087: 1, 12.388264402852034: 1,
12.377556581301288: 1, 12.372787002066548: 1, 12.352110382629073: 1, 12.335904539071137: 1,
12.321355606322669: 1, 12.287338916693805: 1, 12.286220627409405: 1, 12.282661891536165: 1,
12.255821329977547: 1, 12.214084982535901: 1, 12.207081958077168: 1, 12.191055024227568: 1,
12.140137266994223: 1, 12.136153423172338: 1, 12.111708975639054: 1, 12.096996210477753: 1,
12.095778931649795: 1, 12.092573635457224: 1, 12.091571474919164: 1, 12.083247883059425: 1,
12.06791386568696: 1, 12.058822854230009: 1, 12.035732404038745: 1, 12.030308064070l2: 1,
12.019551562543771: 1, 12.003210734748711: 1, 11.95445700847629: 1, 11.930866694729108: 1,
11.916612441120055: 1, 11.90858473802334: 1, 11.907805902227114: 1, 11.883060432555691: 1,
11.85963627501184: 1, 11.850582491510252: 1, 11.84529627754525: 1, 11.834686349507372: 1,
11.816347962063213: 1, 11.801047993842012: 1, 11.775251084017835: 1, 11.766167486560969: 1,
11.674811809631677: 1, 11.628248668281202: 1, 11.593294888262706: 1, 11.58896403893774: 1,
11.577395742670891: 1, 11.573945291571084: 1, 11.53440007137343: 1, 11.507784804788562: 1,
11.506779468513317: 1, 11.505370326879575: 1, 11.495612162724253: 1, 11.4778322916485: 1,
11.467133211200572: 1, 11.459123529235148: 1, 11.458848059279106: 1, 11.397392629651206: 1,
11.397123160730496: 1, 11.378408079287539: 1, 11.33774496210947: 1, 11.330144504195083: 1,
11.32670919344832: 1, 11.312417472437561: 1, 11.30829656227097: 1, 11.282589877689453: 1,
11.275726147645809: 1, 11.264073527126344: 1, 11.25511667839131: 1, 11.242633474197024: 1,
11.237451768808953: 1, 11.23586428851079: 1, 11.233087604440602: 1, 11.23007244590591: 1,
11.220573437022516: 1, 11.219836641098285: 1, 11.205044080756394: 1, 11.176823594690482: 1,
11.126392487837139: 1, 11.122193299941069: 1, 11.116620573880112: 1, 11.112045683517483: 1,
11.097662835173761: 1, 11.085494635298096: 1, 11.080516254715128: 1, 11.077226219404025: 1,
11.067079267247795: 1, 11.05714445194O8: 1, 11.0186606612139l6: 1, 11.0128858200219l83: 1,
10.976671280804249: 1, 10.945185296766l3: 1, 10.931793885343799: 1, 10.920331929763977: 1,
10.905325959813723: 1, 10.898772610261089: 1, 10.897591737002776: 1, 10.872799382490127: 1,
10.868464780628635: 1, 10.853226750850055: 1, 10.853211727948553: 1, 10.852681266084142: 1,
10.81413401664713: 1, 10.810996101353l5: 1, 10.792653006428178: 1, 10.791519728388662: 1,
10.781893045743148: 1, 10.771341079090249: 1, 10.745284042913493: 1, 10.743633455665l2: 1,
10.725232907728l9: 1, 10.704421099672178: 1, 10.703997498287347: 1, 10.654062646381l7: 1,
10.651583514433l7: 1, 10.650875148639997: 1, 10.648331973814004: 1, 10.632687013673l: 1,
10.625801766918356: 1, 10.583066159016903: 1, 10.577325679902845: 1, 10.561825609879115: 1,
10.55172867597l7: 1, 10.543667297556063: 1, 10.540925008386685: 1, 10.532367673627l3: 1,
10.52004175319267: 1, 10.508652991341343: 1, 10.491172345188899: 1, 10.487510164930093: 1,
10.484384970607438: 1, 10.478310049722735: 1, 10.473148255622808: 1, 10.454462338480685: 1,
10.4404489872326: 1, 10.439721526240199: 1, 10.436674671253842: 1, 10.4230383848382931: 1,
10.418443194630713: 1, 10.398398961378106: 1, 10.35524651615204: 1, 10.354543991032669: 1,
10.353249013375793: 1, 10.329817605558581: 1, 10.316285398622583: 1, 10.311910271366786: 1,
10.294746667706946: 1, 10.278937111632553: 1, 10.272602355562388: 1, 10.267638226817535: 1,
10.255271023467259: 1, 10.244772471237036: 1, 10.232507034610O8: 1, 10.221386747268351: 1,
10.214224540252724: 1, 10.211963434578976: 1, 10.211630847763671: 1, 10.210044172408516: 1,
10.197796264246849: 1, 10.194820327222466: 1, 10.178627102432552: 1, 10.176265937092673: 1,
10.160341791638571: 1, 10.149305795631086: 1, 10.146421630800006: 1, 10.140863319318756: 1,

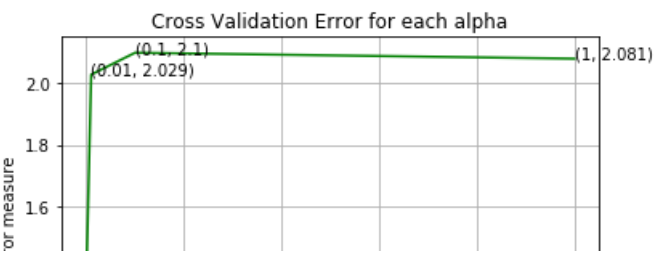
10.124569273286589: 1, 10.10528750059841 5: 1, 10.080599069757536: 1, 10.05022649258923 6: 1,
10.049386239952382: 1, 10.03230619531243: 1, 10.025423796132294: 1, 10.020840124143026: 1,
10.01964369030781: 1, 9.974117232960584: 1, 9.97127761990364: 1, 9.965483600662397: 1,
9.956570541864254: 1, 9.954230633985686: 1, 9.947221376983894: 1, 9.929312992678692: 1,
9.91439360734802: 1, 9.906498616186354: 1, 9.891184608597246: 1, 9.874017995165485: 1,
9.867103106220606: 1, 9.863620147526102: 1, 9.861462023226082: 1, 9.8597232234154 98: 1,
9.85439839655548: 1, 9.8418641654749: 1, 9.841663537674433: 1, 9.82138228236178: 1,
9.812140929297717: 1, 9.79496154062169: 1, 9.79390082524055: 1, 9.789006189728614: 1,
9.784181685859219: 1, 9.783566233098345: 1, 9.757007314285472: 1, 9.754455022435494: 1,
9.748734368210604: 1, 9.725366648409453: 1, 9.72463247029339: 1, 9.698921590515647: 1,
9.684127932492148: 1, 9.642788562144808: 1, 9.639570441169987: 1, 9.610003590119 82: 1,
9.606096785151 36: 1, 9.585513269681842: 1, 9.584456632945189: 1, 9.566989353470753: 1,
9.562764560933 88: 1, 9.55245766435323: 1, 9.540800217893565: 1, 9.540775824088453: 1,
9.537970044978968: 1, 9.529248528199064: 1, 9.524973156950841: 1, 9.513100684598301: 1,
9.500001757462195: 1, 9.493601703711006: 1, 9.484683638208386: 1, 9.480954932479 29: 1,
9.480591359045635: 1, 9.477757805349285: 1, 9.455068556020 06: 1, 9.431845495963408: 1,
9.427746220123 72: 1, 9.422261832796 42: 1, 9.417369611372424: 1, 9.416604411693827: 1,
9.405003823331764: 1, 9.398293782557 17: 1, 9.392003461865434: 1, 9.385492470787877: 1,
9.380725676024 55: 1, 9.368581339110559: 1, 9.332602004775401: 1, 9.327439693320638: 1,
9.325711701069176: 1, 9.310391944997695: 1, 9.307764877219 17: 1, 9.295883433734 36: 1,
9.291425611120367: 1, 9.288218772400 22: 1, 9.279787191539318: 1, 9.271423527372916: 1,
9.266109371202386: 1, 9.259271074775565: 1, 9.258988048939813: 1, 9.230343199660382: 1,
9.197128303694576: 1, 9.189909785438292: 1, 9.187962168729094: 1, 9.183878330773743: 1,
9.169432259114842: 1, 9.163450756775845: 1, 9.138718182241092: 1, 9.138179427946183: 1,
9.136161458799277: 1, 9.111640083275672: 1, 9.108139524887635: 1, 9.105992272312136: 1,
9.100893144959 28: 1, 9.076914318809656: 1, 9.071788206554727: 1, 9.060416689053 51: 1,
9.056083720748 82: 1, 9.043665100045109: 1, 9.021526875970354: 1, 9.011705159123458: 1,
9.006674604134538: 1, 8.999894846835502: 1, 8.971339618720348: 1, 8.966737083753024: 1,
8.960006613700921: 1, 8.955689712891928: 1, 8.947136443313786: 1, 8.934105899494 58: 1,
8.926716951639088: 1, 8.924704176174 63: 1, 8.921581447647556: 1, 8.916700652128227: 1,
8.909303181288944: 1, 8.907778301512089: 1, 8.907136573169 58: 1, 8.884728263667089: 1,
8.882926760751724: 1, 8.879718351073198: 1, 8.868200102630464: 1, 8.865283897809359: 1,
8.853493192361523: 1, 8.852547598889 83: 1, 8.847666273835484: 1, 8.846119652356654: 1,
8.845522822165059: 1, 8.812180222035852: 1, 8.801021380202554: 1, 8.774908980622127: 1,
8.774456918296 67: 1, 8.761935353018245: 1, 8.761887746945948: 1, 8.760284449247422: 1,
8.753547039085 59: 1, 8.735809257022696: 1, 8.733790094602382: 1, 8.705181426002124: 1,
8.687508389483051: 1, 8.662803554623494: 1, 8.656440943646 86: 1, 8.652840715561268: 1,
8.628091226067458: 1, 8.623649705682155: 1, 8.615854329966849: 1, 8.611453172810 98: 1,
8.605331035483298: 1, 8.594832878682107: 1, 8.583207392970401: 1, 8.580165385241875: 1,
8.571977824343055: 1, 8.561603255063982: 1, 8.560423923019 47: 1, 8.557075462224697: 1,
8.556247402751874: 1, 8.555220592593264: 1, 8.537478827922403: 1, 8.537296653278556: 1,
8.503834512624401: 1, 8.499250485889695: 1, 8.473220194236282: 1, 8.468560176900233: 1,
8.451434506540123: 1, 8.448852849980987: 1, 8.443100773499223: 1, 8.436314202217915: 1,
8.396255402500197: 1, 8.387216141698724: 1, 8.381132274234561: 1, 8.360354328881 89: 1,
8.357907967103188: 1, 8.355707521388904: 1, 8.347123699861625: 1, 8.342250781083617: 1,
8.323800768253866: 1, 8.294395260469786: 1, 8.289281843099667: 1, 8.259075336933 6: 1,
8.245524384005142: 1, 8.226979666817 45: 1, 8.224814500273036: 1, 8.223264891940401: 1,
8.190997921061626: 1, 8.180103506326308: 1, 8.156426818284062: 1, 8.130259264094404: 1,
8.115378576552798: 1, 8.110263668139643: 1, 8.105258384319068: 1, 8.088910845818676: 1,
8.081632757766869: 1, 8.081401261914797: 1, 8.060159639041201: 1, 8.053571310974924: 1,
8.0430197198761: 1, 8.029877193352034: 1, 8.021573772340988: 1, 8.018112343959144: 1, 8.01788703534
34: 1, 8.016627563820807: 1, 8.00867956243037: 1, 8.006400810227396: 1, 7.973133425144697: 1, 7.950
425623495578: 1, 7.946347383901843: 1, 7.938056444539051: 1, 7.9322965426946155: 1,
7.92669628588417: 1, 7.916272486663908: 1, 7.898642044467431: 1, 7.861775979561316: 1,
7.85671130626384: 1, 7.849923843706098: 1, 7.837911389579805: 1, 7.833820705749519: 1,
7.811299759080497: 1, 7.810215222376364: 1, 7.809119665812085: 1, 7.807540475358174: 1,
7.800436033214853: 1, 7.798640382616556: 1, 7.775909788347329: 1, 7.734816664998208: 1,
7.685425159837274: 1, 7.677030555565127: 1, 7.669249669720266: 1, 7.666877120057322: 1,
7.655502180350904: 1, 7.654473988933779: 1, 7.653355500501547: 1, 7.644648815983388: 1,
7.640427547043895: 1, 7.611630913203177: 1, 7.602037648030 5965: 1, 7.601892826926262: 1,
7.5910803510923435: 1, 7.589411424209006: 1, 7.579735032068677: 1, 7.567625895637764: 1, 7.54351504
4515271: 1, 7.466881209734193: 1, 7.466778088436382: 1, 7.425189788146608: 1, 7.414037457431298: 1,
7.389216290537969: 1, 7.386532509836551: 1, 7.376814964071 63: 1, 7.362317385901542: 1,
7.337710317527593: 1, 7.3154831072672755: 1, 7.300674994414332: 1, 7.2654960616656705: 1,
7.233929487105558: 1, 7.232481283002 58: 1, 7.231268739356 53: 1, 7.182788440425236: 1,
7.169914478723915: 1, 7.160163207392553: 1, 7.143678966771719: 1, 7.132707722463 61: 1,
7.122232722215794: 1, 7.117631295388989: 1, 7.062228128381529: 1, 7.040984545269686: 1,
7.011996043035295: 1, 6.974907572260788: 1, 6.958368935589127: 1, 6.951678649364942: 1,
6.903881388513675: 1, 6.897334399916874: 1, 6.881831625876521: 1, 6.877373443488571: 1,
6.840569676067342: 1, 6.8373926450115405: 1, 6.761736857881402: 1, 6.666018158561651: 1,
6.642243404222107: 1, 6.62155813163023: 1, 6.619859008359633: 1, 6.606575872625798: 1,
6.440787553107125: 1})

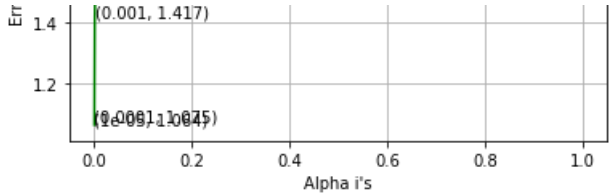
In [48]:

```
# 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: 1.0643188743178482
For values of alpha =  0.0001 The log loss is: 1.0747162902797915
For values of alpha =  0.001 The log loss is: 1.4167558326558294
For values of alpha =  0.01 The log loss is: 2.028843265523044
For values of alpha =  0.1 The log loss is: 2.1002015181864113
For values of alpha =  1 The log loss is: 2.0808354395617634
```






```
For values of best alpha =  1e-05 The train log loss is: 0.7861409443318345
For values of best alpha =  1e-05 The cross validation log loss is: 1.0643188743178482
For values of best alpha =  1e-05 The test log loss is: 1.1258970344905883
```

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

Ans. Yes, it seems like!

In [49]:

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

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

In [50]:

```
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")
```

```
94.2 % of word of test data appeared in train data
93.6 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [51]:

```
#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 [52]:

```
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 [53]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(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 [54]:

```
# 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 [55]:

```
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, 3197)
(number of data points * number of features) in test data =  (665, 3197)
(number of data points * number of features) in cross validation data = (532, 3197)
```

In [56]:

```
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

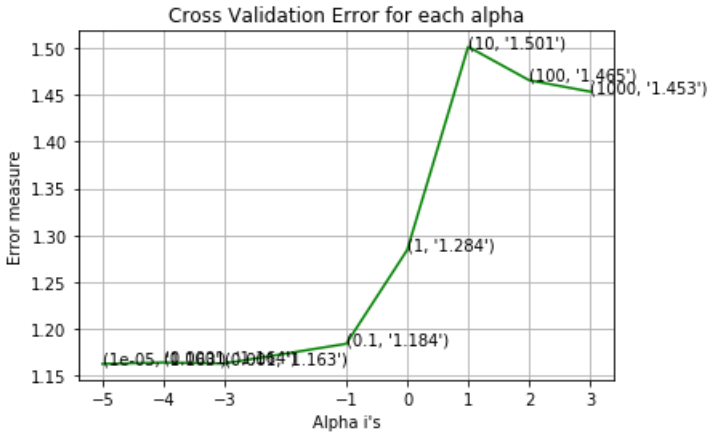
In [57]:

```
# 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))

# Variables that will be used in the end to make comparison table of all models
nb_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=
1e-15)
nb_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
nb_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-
15)
```

```
for alpha = 1e-05
Log Loss : 1.1627203702835112
for alpha = 0.0001
Log Loss : 1.164146044359952
for alpha = 0.001
Log Loss : 1.1630906565423742
for alpha = 0.1
Log Loss : 1.1841434523975574
for alpha = 1
Log Loss : 1.2842463123291985
for alpha = 10
Log Loss : 1.5010785571726184
for alpha = 100
Log Loss : 1.4653383657738797
```

```
for alpha = 1000
Log Loss : 1.453200654563474
```




```
For values of best alpha =  1e-05 The train log loss is: 0.5357886635941876
For values of best alpha =  1e-05 The cross validation log loss is: 1.1627203702835112
For values of best alpha =  1e-05 The test log loss is: 1.213810087257443
```

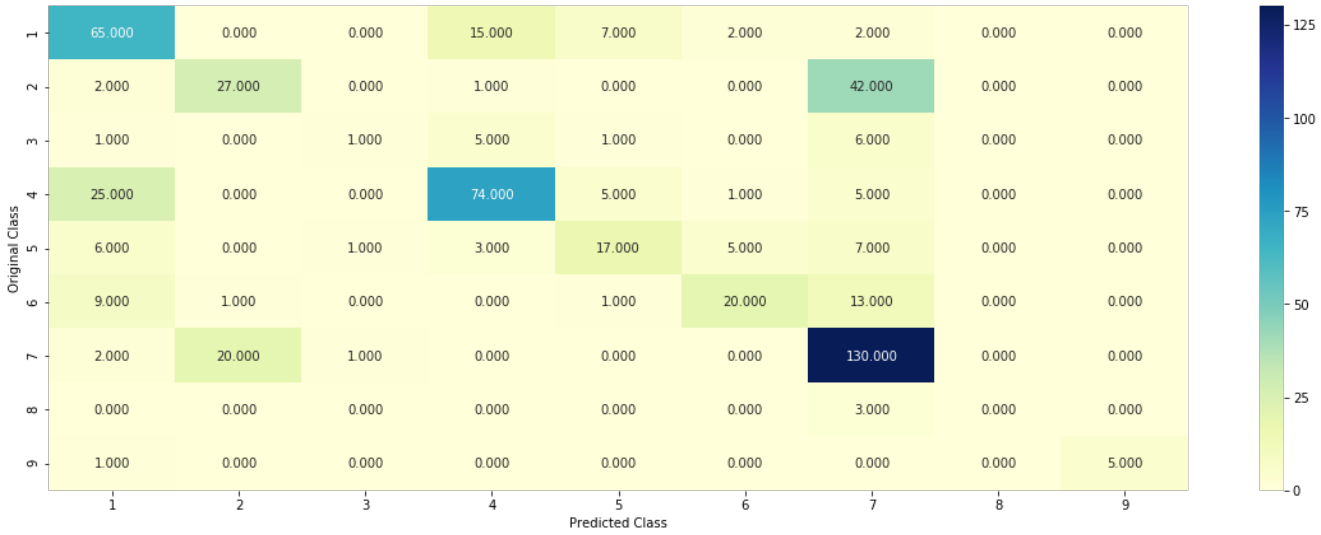
4.1.1.2. Testing the model with best hyper paramters

In [58]:

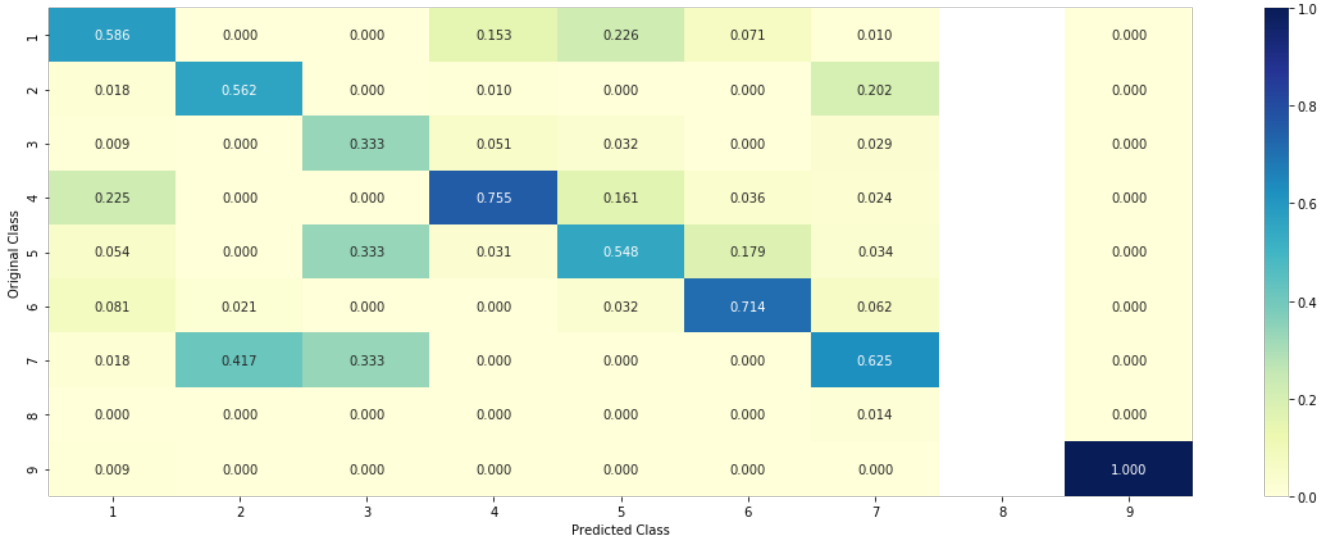
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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()))

# Variables that will be used in the end to make comparison table of models
nb_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

```
Log Loss : 1.1627203702835112
```

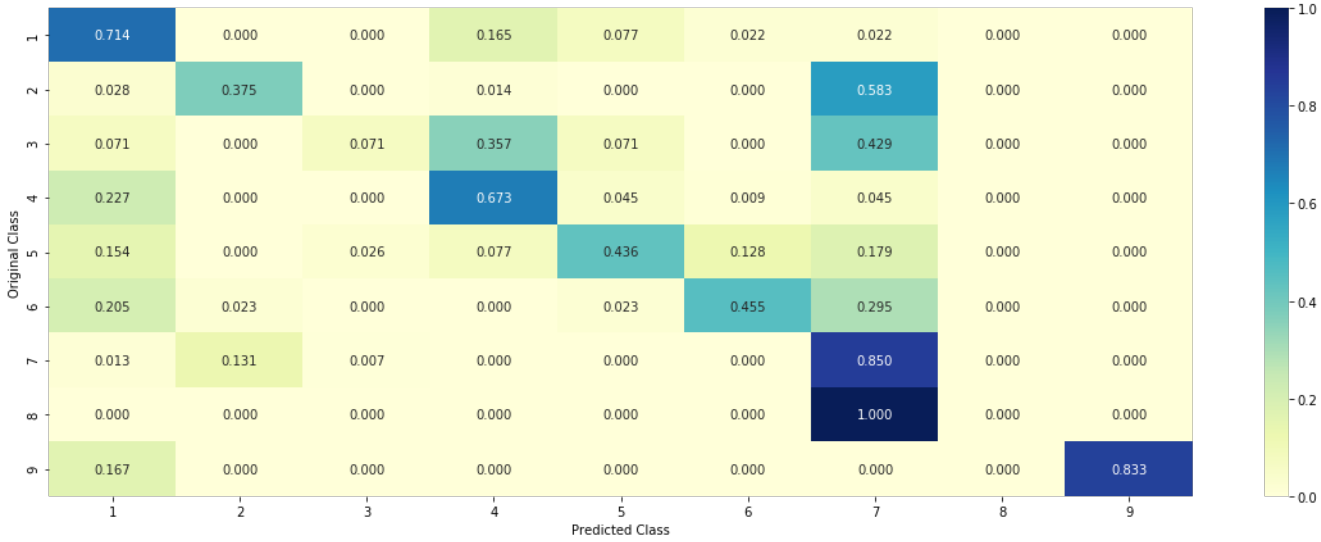
```
Number of missclassified point : 0.36278195488721804
-------------------- Confusion matrix --------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [59]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0645 0.0411 0.0122 0.7246 0.0331 0.0342 0.0856 0.0022 0.0024]]
Actual Class : 4
--------------------------------------------------
9 Text feature [activity] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [proteins] present in test data point [True]
16 Text feature [function] present in test data point [True]
17 Text feature [missense] present in test data point [True]
19 Text feature [acid] present in test data point [True]
23 Text feature [results] present in test data point [True]
25 Text feature [amino] present in test data point [True]
26 Text feature [type] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [whether] present in test data point [True]
29 Text feature [wild] present in test data point [True]
30 Text feature [whereas] present in test data point [True]
31 Text feature [shown] present in test data point [True]
32 Text feature [mutations] present in test data point [True]
33 Text feature [functional] present in test data point [True]
34 Text feature [described] present in test data point [True]
36 Text feature [important] present in test data point [True]
37 Text feature [two] present in test data point [True]
38 Text feature [may] present in test data point [True]
39 Text feature [related] present in test data point [True]
40 Text feature [determined] present in test data point [True]
42 Text feature [thus] present in test data point [True]
43 Text feature [either] present in test data point [True]
44 Text feature [ability] present in test data point [True]
45 Text feature [discussion] present in test data point [True]
46 Text feature [indicate] present in test data point [True]
47 Text feature [previously] present in test data point [True]
49 Text feature [show] present in test data point [True]
50 Text feature [suppressor] present in test data point [True]
52 Text feature [three] present in test data point [True]
53 Text feature [vitro] present in test data point [True]
54 Text feature [although] present in test data point [True]
55 Text feature [bind] present in test data point [True]
58 Text feature [therefore] present in test data point [True]
59 Text feature [similar] present in test data point [True]
60 Text feature [see] present in test data point [True]
61 Text feature [containing] present in test data point [True]
62 Text feature [30] present in test data point [True]
64 Text feature [suggesting] present in test data point [True]
65 Text feature [contribute] present in test data point [True]
66 Text feature [introduction] present in test data point [True]
67 Text feature [associated] present in test data point [True]
68 Text feature [found] present in test data point [True]
69 Text feature [one] present in test data point [True]
70 Text feature [loss] present in test data point [True]
72 Text feature [several] present in test data point [True]
73 Text feature [mutation] present in test data point [True]
74 Text feature [10] present in test data point [True]
75 Text feature [analysis] present in test data point [True]
76 Text feature [however] present in test data point [True]
77 Text feature [substitutions] present in test data point [True]
78 Text feature [lower] present in test data point [True]
79 Text feature [addition] present in test data point [True]
80 Text feature [assay] present in test data point [True]
81 Text feature [reported] present in test data point [True]
82 Text feature [tagged] present in test data point [True]
```

```
83 Text feature [possible] present in test data point [True]
84 Text feature [mutant] present in test data point [True]
88 Text feature [could] present in test data point [True]
89 Text feature [binding] present in test data point [True]
90 Text feature [cells] present in test data point [True]
91 Text feature [affect] present in test data point [True]
92 Text feature [indicated] present in test data point [True]
94 Text feature [mutants] present in test data point [True]
95 Text feature [effects] present in test data point [True]
96 Text feature [figure] present in test data point [True]
97 Text feature [acids] present in test data point [True]
99 Text feature [used] present in test data point [True]
Out of the top  100  features  69 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [60]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0617 0.0435 0.0129 0.0636 0.0352 0.6874 0.0908 0.0023 0.0026]]
Actual Class : 4
--------------------------------------------------
6 Text feature [odds] present in test data point [True]
7 Text feature [brca] present in test data point [True]
8 Text feature [deleterious] present in test data point [True]
9 Text feature [classified] present in test data point [True]
10 Text feature [57] present in test data point [True]
11 Text feature [basis] present in test data point [True]
12 Text feature [history] present in test data point [True]
13 Text feature [combined] present in test data point [True]
14 Text feature [49] present in test data point [True]
15 Text feature [brca2] present in test data point [True]
16 Text feature [43] present in test data point [True]
17 Text feature [family] present in test data point [True]
18 Text feature [26] present in test data point [True]
19 Text feature [52] present in test data point [True]
20 Text feature [sequence] present in test data point [True]
21 Text feature [evidence] present in test data point [True]
22 Text feature [000] present in test data point [True]
23 Text feature [model] present in test data point [True]
24 Text feature [41] present in test data point [True]
25 Text feature [brca1] present in test data point [True]
26 Text feature [variant] present in test data point [True]
27 Text feature [models] present in test data point [True]
28 Text feature [testing] present in test data point [True]
29 Text feature [31] present in test data point [True]
30 Text feature [23] present in test data point [True]
31 Text feature [neutral] present in test data point [True]
32 Text feature [expected] present in test data point [True]
33 Text feature [predicted] present in test data point [True]
34 Text feature [47] present in test data point [True]
35 Text feature [45] present in test data point [True]
36 Text feature [given] present in test data point [True]
37 Text feature [variants] present in test data point [True]
38 Text feature [54] present in test data point [True]
39 Text feature [use] present in test data point [True]
40 Text feature [used] present in test data point [True]
41 Text feature [34] present in test data point [True]
42 Text feature [data] present in test data point [True]
43 Text feature [known] present in test data point [True]
44 Text feature [substitutions] present in test data point [True]
```

```
45 Text feature [36] present in test data point [True]
46 Text feature [35] present in test data point [True]
47 Text feature [56] present in test data point [True]
48 Text feature [70] present in test data point [True]
50 Text feature [significance] present in test data point [True]
51 Text feature [42] present in test data point [True]
52 Text feature [46] present in test data point [True]
53 Text feature [would] present in test data point [True]
54 Text feature [50] present in test data point [True]
55 Text feature [values] present in test data point [True]
56 Text feature [conserved] present in test data point [True]
57 Text feature [likely] present in test data point [True]
58 Text feature [25] present in test data point [True]
59 Text feature [75] present in test data point [True]
60 Text feature [site] present in test data point [True]
61 Text feature [32] present in test data point [True]
62 Text feature [vus] present in test data point [True]
63 Text feature [ovarian] present in test data point [True]
64 Text feature [analysis] present in test data point [True]
65 Text feature [available] present in test data point [True]
66 Text feature [least] present in test data point [True]
67 Text feature [studied] present in test data point [True]
68 Text feature [individuals] present in test data point [True]
69 Text feature [28] present in test data point [True]
70 Text feature [overall] present in test data point [True]
71 Text feature [population] present in test data point [True]
72 Text feature [although] present in test data point [True]
73 Text feature [33] present in test data point [True]
74 Text feature [number] present in test data point [True]
75 Text feature [information] present in test data point [True]
76 Text feature [39] present in test data point [True]
77 Text feature [significant] present in test data point [True]
78 Text feature [proportion] present in test data point [True]
79 Text feature [risk] present in test data point [True]
80 Text feature [developed] present in test data point [True]
81 Text feature [methods] present in test data point [True]
82 Text feature [prior] present in test data point [True]
83 Text feature [approach] present in test data point [True]
84 Text feature [characteristics] present in test data point [True]
85 Text feature [29] present in test data point [True]
86 Text feature [51] present in test data point [True]
87 Text feature [60] present in test data point [True]
88 Text feature [substitution] present in test data point [True]
89 Text feature [genetic] present in test data point [True]
90 Text feature [72] present in test data point [True]
91 Text feature [database] present in test data point [True]
92 Text feature [provide] present in test data point [True]
93 Text feature [12] present in test data point [True]
94 Text feature [studies] present in test data point [True]
95 Text feature [40] present in test data point [True]
96 Text feature [missense] present in test data point [True]
97 Text feature [none] present in test data point [True]
98 Text feature [44] present in test data point [True]
99 Text feature [based] present in test data point [True]
Out of the top  100  features  93 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

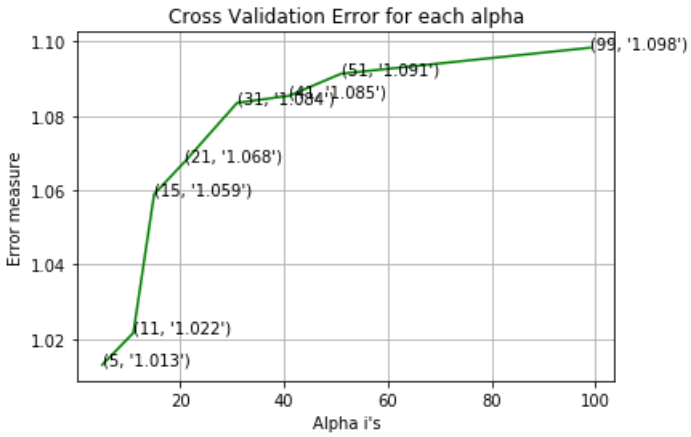
In [61]:

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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#
```

```
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying 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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

# Variables that will be used in the end to make comparison table of all models
knn_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labels=clf.classes_, eps=1e-15)
knn_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.classes_, eps=1e-15)
knn_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=clf.classes_, eps=1e-15)
```

```
for alpha = 5
Log Loss : 1.0130873350555543
for alpha = 11
Log Loss : 1.0216884171620246
for alpha = 15
Log Loss : 1.0589593368066004
for alpha = 21
Log Loss : 1.0678218811518234
```

```
for alpha = 31
Log Loss : 1.0835063332807073
for alpha = 41
Log Loss : 1.0853874126895267
for alpha = 51
Log Loss : 1.0913132167738657
for alpha = 99
Log Loss : 1.0983067197665266
```



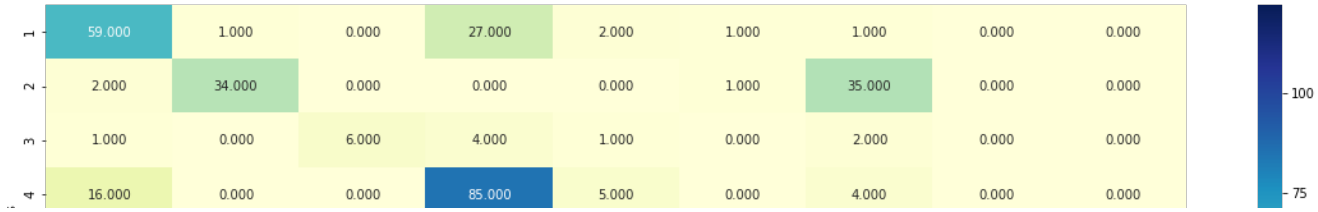

```
For values of best alpha =  5 The train log loss is: 0.5059032744599191
For values of best alpha =  5 The cross validation log loss is: 1.0130873350555543
For values of best alpha =  5 The test log loss is: 1.0780972527681583
```

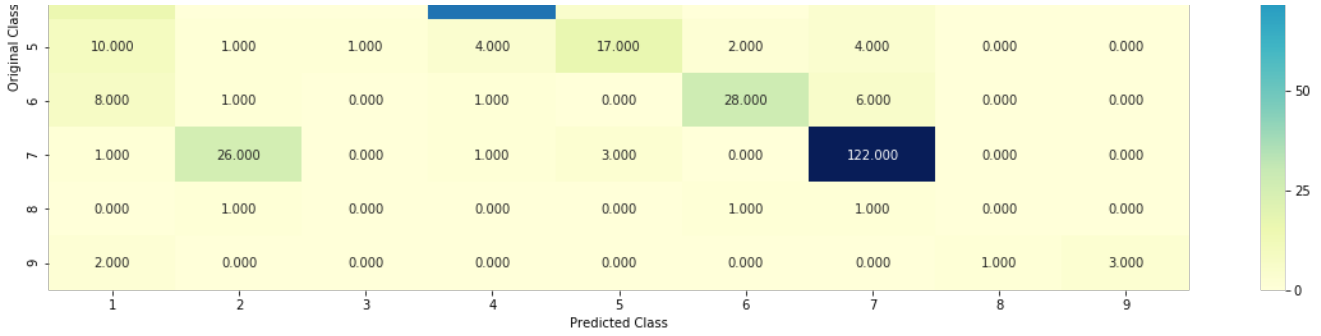
4.2.2. Testing the model with best hyper paramters

In [62]:

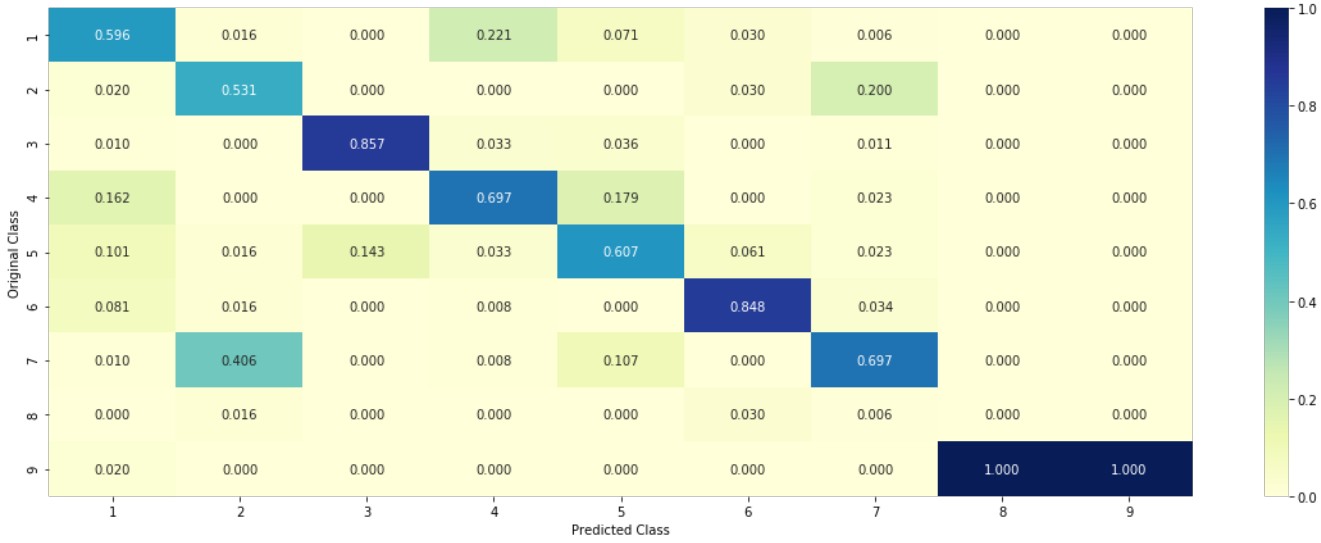
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', 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)

clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

# Variables that will be used in the end to make comparison table of models
knn_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y))/cv_y.shape[0])
*100
```

```
Log loss : 1.0130873350555543
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

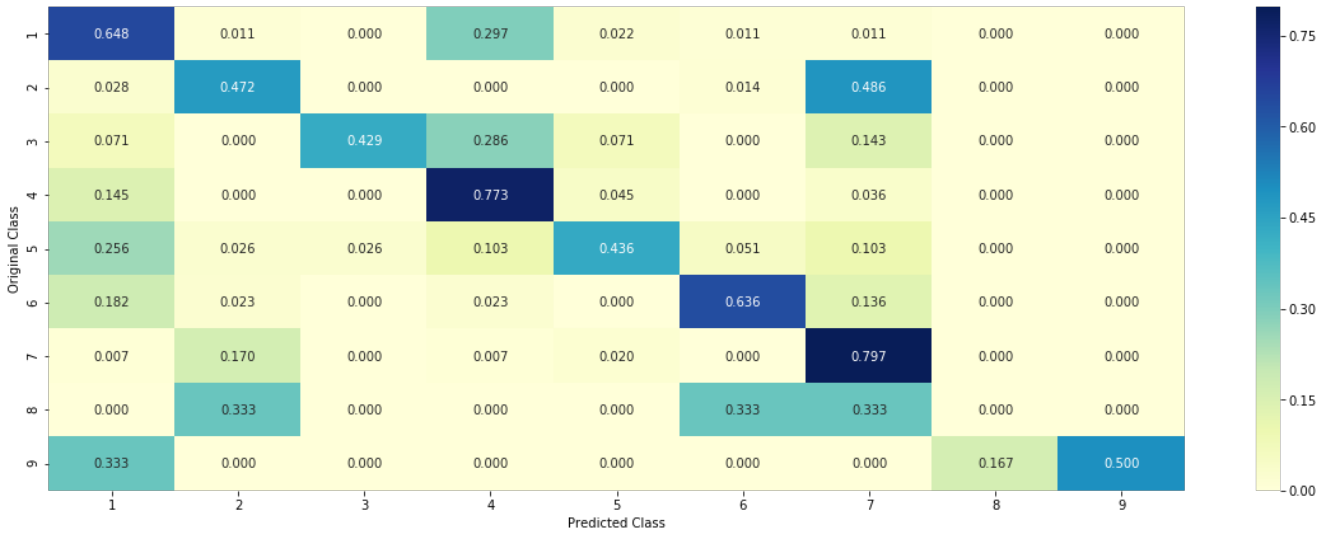




---------- Precision matrix (Columm Sum=1) ----------



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



4.2.3.Sample Query point -1

In [63]:

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

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

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

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

4.2.4. Sample Query Point-2

In [64]:

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

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

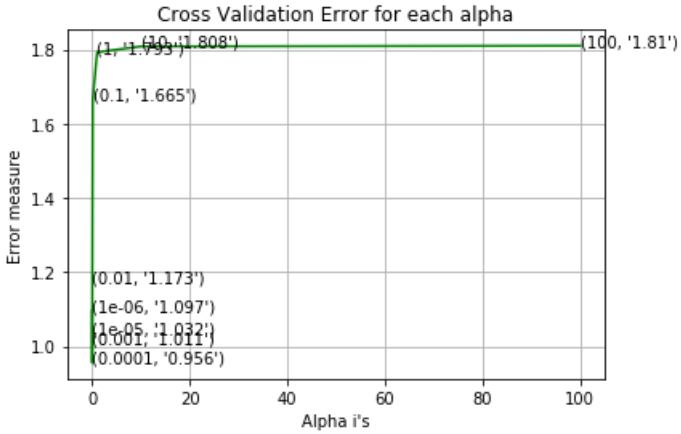
In [65]:

```
# 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
```

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

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

# Variables that will be used in the end to make comparison table of all models
lr_balance_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classe
s_, eps=1e-15)
lr_balance_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1
e-15)
lr_balance_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_,
eps=1e-15)
```

```
for alpha = 1e-06
Log Loss : 1.096792541731062
for alpha = 1e-05
Log Loss : 1.0324571571094006
for alpha = 0.0001
Log Loss : 0.9555910935177091
for alpha = 0.001
Log Loss : 1.010559788851284
for alpha = 0.01
Log Loss : 1.172897759158335
for alpha = 0.1
Log Loss : 1.6652656818324745
for alpha = 1
Log Loss : 1.7927074102598126
for alpha = 10
```

```
Log Loss : 1.8082719642828182
for alpha = 100
Log Loss : 1.810139958908488
```



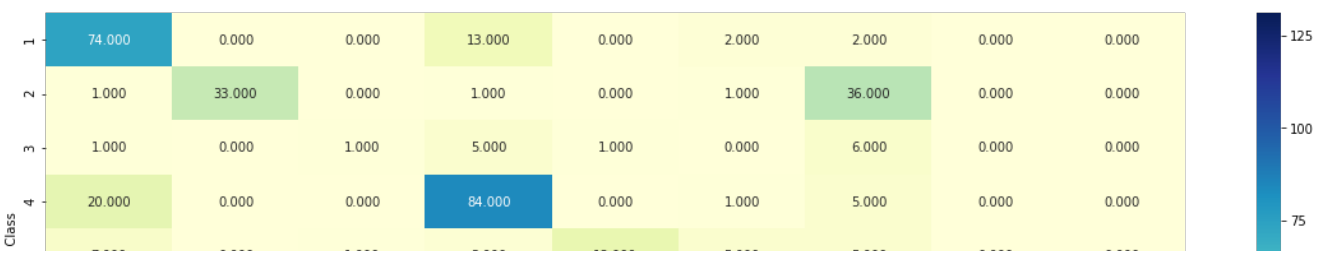

```
For values of best alpha =  0.0001 The train log loss is: 0.45127756128393487
For values of best alpha =  0.0001 The cross validation log loss is: 0.9555910935177091
For values of best alpha =  0.0001 The test log loss is: 1.0141393110847454
```

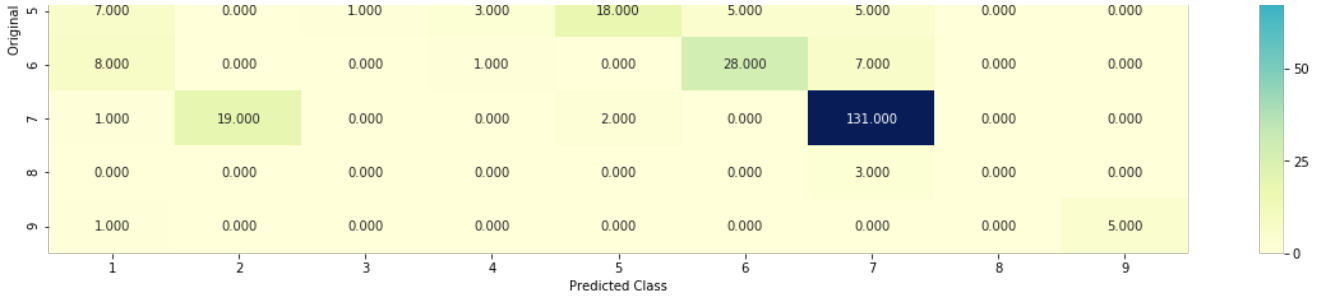
4.3.1.2. Testing the model with best hyper paramters

In [66]:

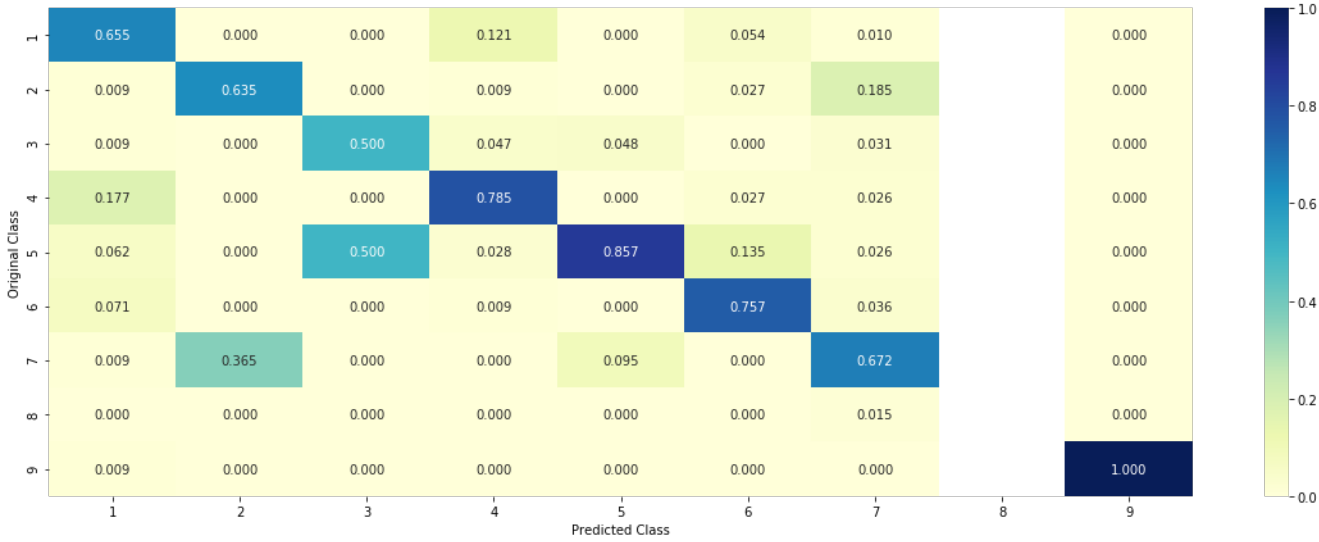
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

# Variables that will be used in the end to make comparison table of models
lr_balance_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

```
Log loss : 0.9555910935177091
Number of mis-classified points : 0.29699248120300753
-------------------- Confusion matrix --------------------
```

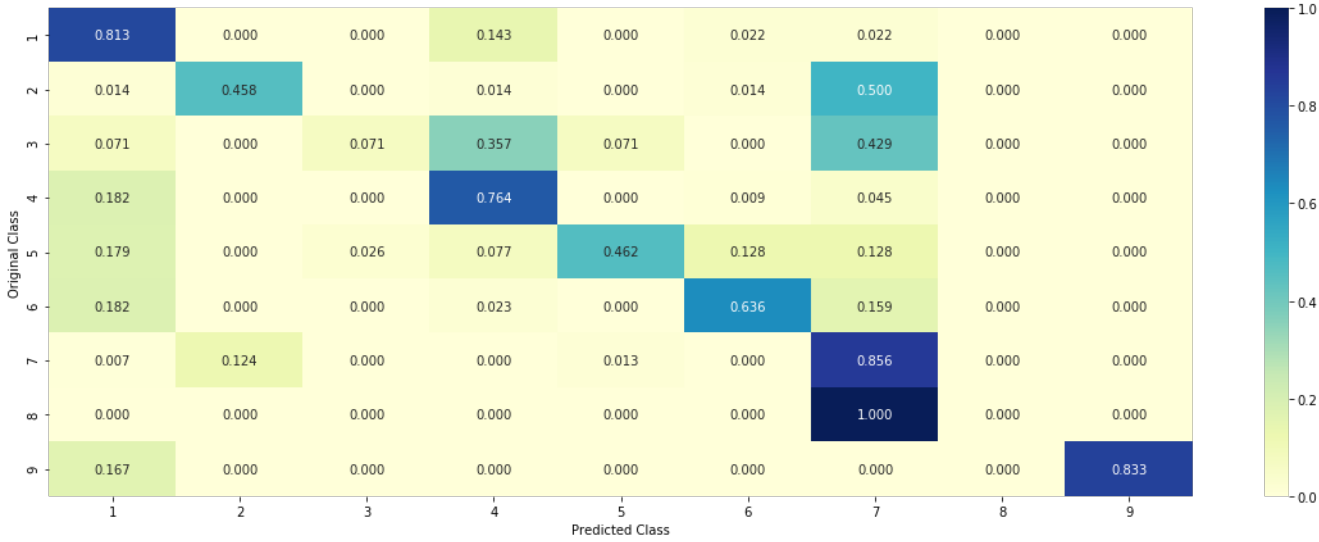




---------- Precision matrix (Columm Sum=1) ----------



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



4.3.1.3. Feature Importance

In [67]:

```
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]
```

```
            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 [68]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[4.860e-02 8.900e-03 2.900e-03 9.163e-01 1.290e-02 3.800e-03 4.700
e-03
  1.100e-03 8.000e-04]]
Actual Class : 4
--------------------------------------------------
20 Text feature [suppressor] present in test data point [True]
70 Text feature [missense] present in test data point [True]
105 Text feature [mm] present in test data point [True]
107 Text feature [unable] present in test data point [True]
129 Text feature [iii] present in test data point [True]
130 Text feature [show] present in test data point [True]
172 Text feature [stability] present in test data point [True]
182 Text feature [groups] present in test data point [True]
239 Text feature [germline] present in test data point [True]
252 Text feature [deletion] present in test data point [True]
268 Text feature [families] present in test data point [True]
270 Text feature [bind] present in test data point [True]
272 Text feature [see] present in test data point [True]
281 Text feature [high] present in test data point [True]
283 Text feature [1998] present in test data point [True]
296 Text feature [express] present in test data point [True]
299 Text feature [tagged] present in test data point [True]
313 Text feature [ca] present in test data point [True]
320 Text feature [protein] present in test data point [True]
329 Text feature [cannot] present in test data point [True]
336 Text feature [represent] present in test data point [True]
337 Text feature [skin] present in test data point [True]
340 Text feature [suggesting] present in test data point [True]
344 Text feature [age] present in test data point [True]
346 Text feature [cause] present in test data point [True]
367 Text feature [bound] present in test data point [True]
368 Text feature [intermediate] present in test data point [True]
372 Text feature [activity] present in test data point [True]
380 Text feature [affected] present in test data point [True]
384 Text feature [indeed] present in test data point [True]
398 Text feature [loss] present in test data point [True]
406 Text feature [alterations] present in test data point [True]
415 Text feature [splice] present in test data point [True]
417 Text feature [consequences] present in test data point [True]
419 Text feature [functionally] present in test data point [True]
425 Text feature [cycle] present in test data point [True]
426 Text feature [relatively] present in test data point [True]
```

```
427 Text feature [functional] present in test data point [True]
430 Text feature [cases] present in test data point [True]
437 Text feature [nuclear] present in test data point [True]
443 Text feature [correlation] present in test data point [True]
453 Text feature [low] present in test data point [True]
458 Text feature [dna] present in test data point [True]
461 Text feature [comparison] present in test data point [True]
466 Text feature [nature] present in test data point [True]
469 Text feature [deletions] present in test data point [True]
472 Text feature [indicate] present in test data point [True]
483 Text feature [function] present in test data point [True]
486 Text feature [phenotype] present in test data point [True]
489 Text feature [degradation] present in test data point [True]
492 Text feature [predicted] present in test data point [True]
493 Text feature [motif] present in test data point [True]
495 Text feature [risk] present in test data point [True]
496 Text feature [contribute] present in test data point [True]
499 Text feature [linked] present in test data point [True]
Out of the top  500  features  55 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [69]:

```
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: [[1.809e-01 7.800e-03 2.000e-03 2.160e-02 1.857e-01 5.962e-01 3.600
e-03
  1.700e-03 5.000e-04]]
Actual Class : 4
--------------------------------------------------
62 Text feature [values] present in test data point [True]
85 Text feature [ci] present in test data point [True]
96 Text feature [brca] present in test data point [True]
99 Text feature [ovarian] present in test data point [True]
113 Text feature [degradation] present in test data point [True]
115 Text feature [substitutions] present in test data point [True]
116 Text feature [dose] present in test data point [True]
120 Text feature [conformation] present in test data point [True]
123 Text feature [ligase] present in test data point [True]
125 Text feature [considered] present in test data point [True]
140 Text feature [site] present in test data point [True]
142 Text feature [showing] present in test data point [True]
147 Text feature [free] present in test data point [True]
152 Text feature [substitution] present in test data point [True]
155 Text feature [49] present in test data point [True]
157 Text feature [cause] present in test data point [True]
159 Text feature [models] present in test data point [True]
165 Text feature [000] present in test data point [True]
166 Text feature [interaction] present in test data point [True]
170 Text feature [deleterious] present in test data point [True]
172 Text feature [occurs] present in test data point [True]
178 Text feature [95] present in test data point [True]
179 Text feature [significant] present in test data point [True]
186 Text feature [confer] present in test data point [True]
187 Text feature [basis] present in test data point [True]
189 Text feature [overall] present in test data point [True]
190 Text feature [expected] present in test data point [True]
203 Text feature [ring] present in test data point [True]
204 Text feature [classified] present in test data point [True]
208 Text feature [odds] present in test data point [True]
212 Text feature [loss] present in test data point [True]
```

```
213 Text feature [57] present in test data point [True]
214 Text feature [isolated] present in test data point [True]
215 Text feature [phase] present in test data point [True]
216 Text feature [associated] present in test data point [True]
217 Text feature [history] present in test data point [True]
218 Text feature [42] present in test data point [True]
219 Text feature [development] present in test data point [True]
220 Text feature [higher] present in test data point [True]
222 Text feature [groups] present in test data point [True]
224 Text feature [structural] present in test data point [True]
228 Text feature [copy] present in test data point [True]
230 Text feature [potential] present in test data point [True]
232 Text feature [particular] present in test data point [True]
233 Text feature [affect] present in test data point [True]
236 Text feature [40] present in test data point [True]
237 Text feature [suppression] present in test data point [True]
238 Text feature [individuals] present in test data point [True]
242 Text feature [studied] present in test data point [True]
244 Text feature [decreased] present in test data point [True]
245 Text feature [characteristics] present in test data point [True]
246 Text feature [status] present in test data point [True]
249 Text feature [nucleotide] present in test data point [True]
250 Text feature [43] present in test data point [True]
251 Text feature [model] present in test data point [True]
252 Text feature [risk] present in test data point [True]
253 Text feature [time] present in test data point [True]
254 Text feature [brca1] present in test data point [True]
255 Text feature [34] present in test data point [True]
259 Text feature [38] present in test data point [True]
261 Text feature [selected] present in test data point [True]
262 Text feature [altered] present in test data point [True]
264 Text feature [54] present in test data point [True]
265 Text feature [46] present in test data point [True]
266 Text feature [therapeutic] present in test data point [True]
268 Text feature [score] present in test data point [True]
269 Text feature [none] present in test data point [True]
270 Text feature [determined] present in test data point [True]
271 Text feature [60] present in test data point [True]
273 Text feature [four] present in test data point [True]
274 Text feature [52] present in test data point [True]
275 Text feature [75] present in test data point [True]
276 Text feature [due] present in test data point [True]
277 Text feature [statistical] present in test data point [True]
278 Text feature [five] present in test data point [True]
279 Text feature [group] present in test data point [True]
280 Text feature [prior] present in test data point [True]
281 Text feature [47] present in test data point [True]
282 Text feature [evidence] present in test data point [True]
284 Text feature [predicted] present in test data point [True]
285 Text feature [induce] present in test data point [True]
286 Text feature [screening] present in test data point [True]
289 Text feature [following] present in test data point [True]
291 Text feature [difference] present in test data point [True]
294 Text feature [single] present in test data point [True]
295 Text feature [72] present in test data point [True]
296 Text feature [observation] present in test data point [True]
297 Text feature [missense] present in test data point [True]
299 Text feature [times] present in test data point [True]
301 Text feature [coding] present in test data point [True]
306 Text feature [amplified] present in test data point [True]
309 Text feature [green] present in test data point [True]
311 Text feature [information] present in test data point [True]
312 Text feature [regulation] present in test data point [True]
313 Text feature [residues] present in test data point [True]
314 Text feature [studies] present in test data point [True]
318 Text feature [population] present in test data point [True]
319 Text feature [30] present in test data point [True]
322 Text feature [type] present in test data point [True]
326 Text feature [51] present in test data point [True]
327 Text feature [35] present in test data point [True]
329 Text feature [response] present in test data point [True]
330 Text feature [active] present in test data point [True]
331 Text feature [44] present in test data point [True]
335 Text feature [six] present in test data point [True]
336 Text feature [changes] present in test data point [True]
337 Text feature [identified] present in test data point [True]
339 Text feature [56] present in test data point [True]
```

```
342 Text feature [direct] present in test data point [True]
343 Text feature [structures] present in test data point [True]
344 Text feature [involved] present in test data point [True]
346 Text feature [set] present in test data point [True]
351 Text feature [23] present in test data point [True]
355 Text feature [strand] present in test data point [True]
358 Text feature [acids] present in test data point [True]
359 Text feature [required] present in test data point [True]
361 Text feature [genomic] present in test data point [True]
362 Text feature [lower] present in test data point [True]
363 Text feature [number] present in test data point [True]
368 Text feature [45] present in test data point [True]
369 Text feature [american] present in test data point [True]
370 Text feature [gel] present in test data point [True]
371 Text feature [given] present in test data point [True]
372 Text feature [resulted] present in test data point [True]
373 Text feature [important] present in test data point [True]
380 Text feature [significance] present in test data point [True]
387 Text feature [200] present in test data point [True]
389 Text feature [classification] present in test data point [True]
390 Text feature [selection] present in test data point [True]
391 Text feature [differences] present in test data point [True]
393 Text feature [frequency] present in test data point [True]
399 Text feature [distinct] present in test data point [True]
402 Text feature [36] present in test data point [True]
404 Text feature [spectrum] present in test data point [True]
407 Text feature [approach] present in test data point [True]
413 Text feature [association] present in test data point [True]
416 Text feature [evaluated] present in test data point [True]
417 Text feature [added] present in test data point [True]
418 Text feature [applied] present in test data point [True]
419 Text feature [family] present in test data point [True]
420 Text feature [mutant] present in test data point [True]
421 Text feature [respectively] present in test data point [True]
424 Text feature [embryonic] present in test data point [True]
425 Text feature [database] present in test data point [True]
427 Text feature [targeting] present in test data point [True]
428 Text feature [binding] present in test data point [True]
429 Text feature [region] present in test data point [True]
430 Text feature [proportion] present in test data point [True]
433 Text feature [occur] present in test data point [True]
434 Text feature [amino] present in test data point [True]
437 Text feature [normal] present in test data point [True]
440 Text feature [general] present in test data point [True]
441 Text feature [terminal] present in test data point [True]
442 Text feature [versus] present in test data point [True]
445 Text feature [shown] present in test data point [True]
446 Text feature [25] present in test data point [True]
449 Text feature [including] present in test data point [True]
450 Text feature [primer] present in test data point [True]
452 Text feature [calculated] present in test data point [True]
453 Text feature [open] present in test data point [True]
455 Text feature [well] present in test data point [True]
456 Text feature [factor] present in test data point [True]
458 Text feature [20] present in test data point [True]
461 Text feature [12] present in test data point [True]
462 Text feature [lines] present in test data point [True]
463 Text feature [70] present in test data point [True]
465 Text feature [measured] present in test data point [True]
467 Text feature [structure] present in test data point [True]
469 Text feature [26] present in test data point [True]
470 Text feature [since] present in test data point [True]
471 Text feature [methods] present in test data point [True]
472 Text feature [conserved] present in test data point [True]
474 Text feature [multiple] present in test data point [True]
477 Text feature [least] present in test data point [True]
478 Text feature [19] present in test data point [True]
479 Text feature [05] present in test data point [True]
481 Text feature [form] present in test data point [True]
482 Text feature [wild] present in test data point [True]
484 Text feature [would] present in test data point [True]
485 Text feature [thus] present in test data point [True]
486 Text feature [90] present in test data point [True]
490 Text feature [sequence] present in test data point [True]
492 Text feature [half] present in test data point [True]
493 Text feature [correlated] present in test data point [True]
495 Text feature [relatively] present in test data point [True]
```

```
496 Text feature [subsequent] present in test data point [True]
497 Text feature [ml] present in test data point [True]
499 Text feature [domains] present in test data point [True]
Out of the top  500  features  188 are present in query point
```

4.3.2. Without Class balancing

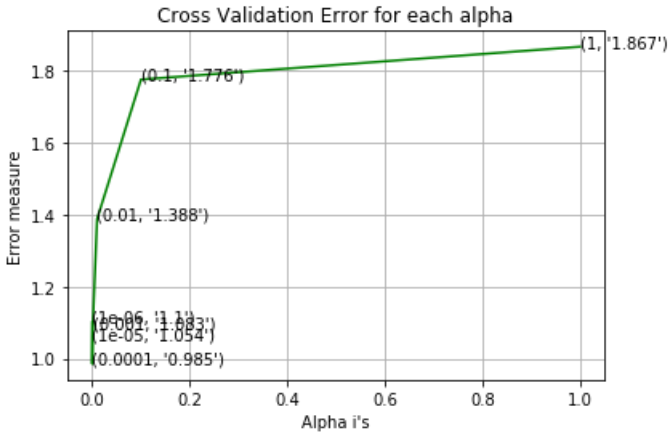
4.3.2.1. Hyper paramter tuning

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, 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))

# Variables that will be used in the end to make comparison table of all models
lr_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=
1e-15)
lr_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
lr_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-
15)
```

```
for alpha = 1e-06
Log Loss : 1.1002874306814379
for alpha = 1e-05
Log Loss : 1.05396980690526
for alpha = 0.0001
Log Loss : 0.9848369466933264
for alpha = 0.001
Log Loss : 1.0828807927043103
for alpha = 0.01
Log Loss : 1.3880086736620783
for alpha = 0.1
Log Loss : 1.7760433585980306
for alpha = 1
Log Loss : 1.86725717854176
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44298744424302083
For values of best alpha =  0.0001 The cross validation log loss is: 0.9848369466933264
For values of best alpha =  0.0001 The test log loss is: 1.0317763394409254
```

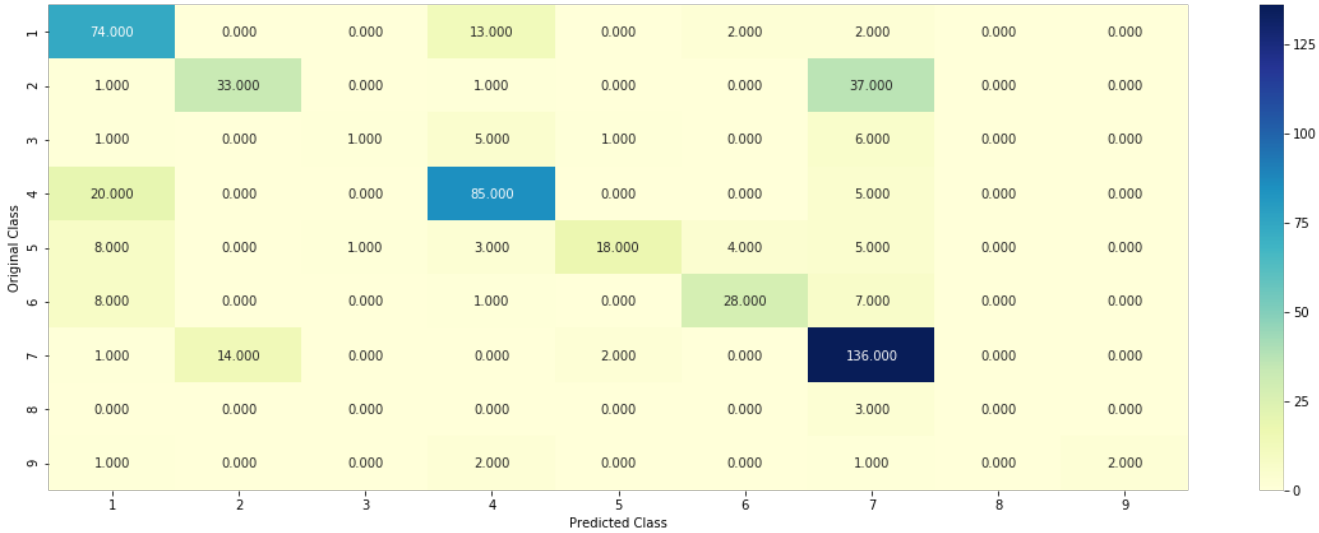
4.3.2.2. Testing model with best hyper parameters

In [71]:

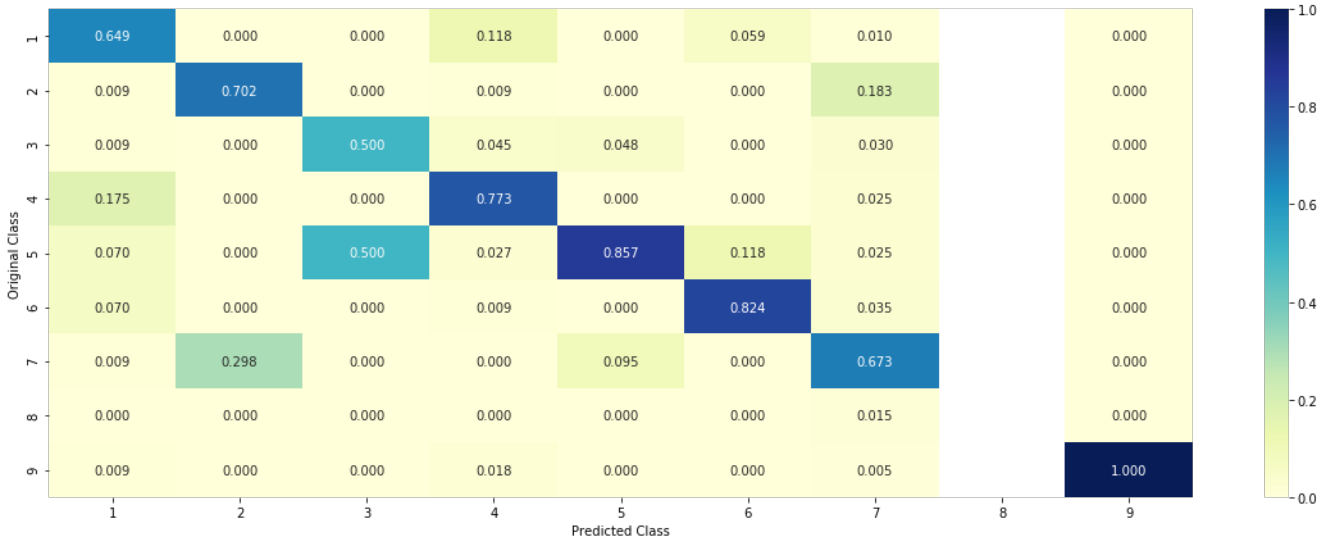
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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)

clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

# Variables that will be used in the end to make comparison table of models
lr_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

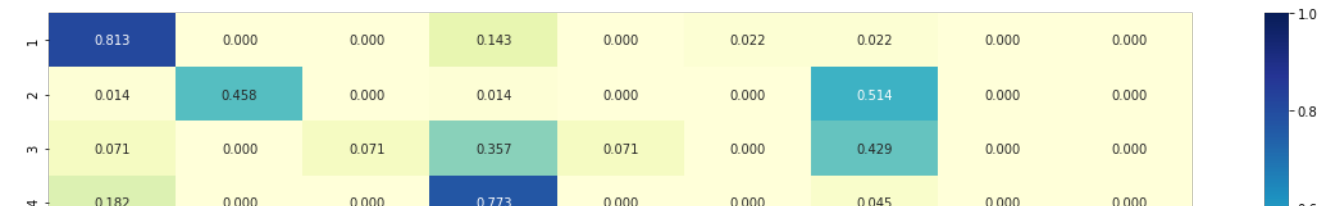
```
Log loss : 0.9848369466933264
Number of mis-classified points : 0.29135338345864664
-------------------- Confusion matrix --------------------
```

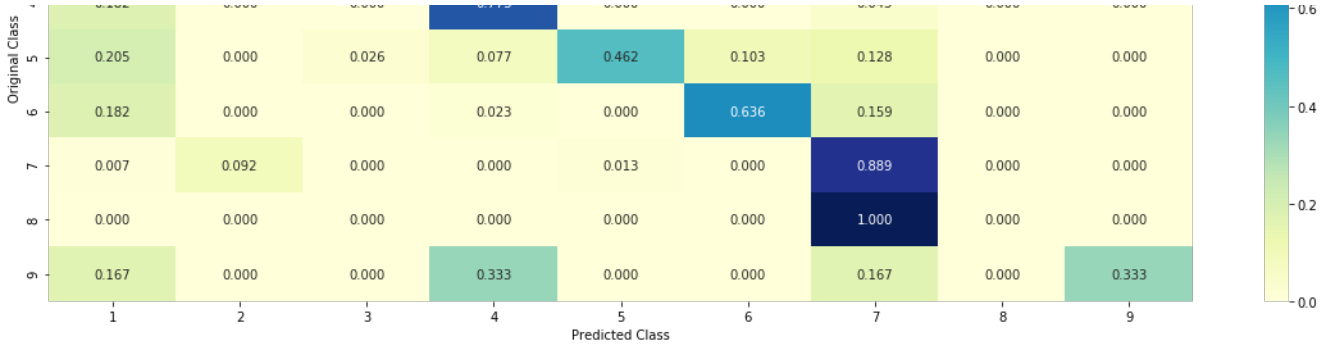



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.2.3. Feature Importance, Correctly Classified point

In [72]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[5.380e-02 9.700e-03 2.400e-03 9.122e-01 1.100e-02 3.600e-03 6.500
e-03
  6.000e-04 2.000e-04]]
Actual Class : 4
--------------------------------------------------
40 Text feature [suppressor] present in test data point [True]
94 Text feature [missense] present in test data point [True]
119 Text feature [unable] present in test data point [True]
125 Text feature [show] present in test data point [True]
126 Text feature [mm] present in test data point [True]
138 Text feature [iii] present in test data point [True]
142 Text feature [groups] present in test data point [True]
199 Text feature [stability] present in test data point [True]
235 Text feature [deletion] present in test data point [True]
237 Text feature [bind] present in test data point [True]
250 Text feature [express] present in test data point [True]
256 Text feature [cannot] present in test data point [True]
278 Text feature [skin] present in test data point [True]
281 Text feature [see] present in test data point [True]
286 Text feature [intermediate] present in test data point [True]
288 Text feature [high] present in test data point [True]
294 Text feature [germline] present in test data point [True]
304 Text feature [tagged] present in test data point [True]
307 Text feature [families] present in test data point [True]
318 Text feature [1998] present in test data point [True]
319 Text feature [represent] present in test data point [True]
320 Text feature [indeed] present in test data point [True]
329 Text feature [bound] present in test data point [True]
334 Text feature [ca] present in test data point [True]
339 Text feature [suggesting] present in test data point [True]
351 Text feature [age] present in test data point [True]
353 Text feature [functionally] present in test data point [True]
356 Text feature [protein] present in test data point [True]
361 Text feature [activity] present in test data point [True]
372 Text feature [splice] present in test data point [True]
382 Text feature [loss] present in test data point [True]
384 Text feature [cases] present in test data point [True]
385 Text feature [relatively] present in test data point [True]
397 Text feature [affected] present in test data point [True]
400 Text feature [correlation] present in test data point [True]
404 Text feature [cause] present in test data point [True]
```

413 Text feature [functional] present in test data point [True]
426 Text feature [nuclear] present in test data point [True]
434 Text feature [deletions] present in test data point [True]
438 Text feature [consequences] present in test data point [True]
440 Text feature [low] present in test data point [True]
444 Text feature [alterations] present in test data point [True]
449 Text feature [function] present in test data point [True]
450 Text feature [risk] present in test data point [True]
451 Text feature [cycle] present in test data point [True]
460 Text feature [kinases] present in test data point [True]
461 Text feature [comparison] present in test data point [True]
463 Text feature [therefore] present in test data point [True]
465 Text feature [described] present in test data point [True]
470 Text feature [contribute] present in test data point [True]
473 Text feature [predicted] present in test data point [True]
479 Text feature [recent] present in test data point [True]
484 Text feature [anti] present in test data point [True]
485 Text feature [nature] present in test data point [True]
487 Text feature [major] present in test data point [True]
489 Text feature [indicate] present in test data point [True]
490 Text feature [level] present in test data point [True]
492 Text feature [impact] present in test data point [True]
496 Text feature [several] present in test data point [True]
497 Text feature [dna] present in test data point [True]
499 Text feature [experimental] present in test data point [True]
Out of the top 500 features 61 are present in query point

4.3.2.4. Feature Importance, Inorrectly Classified point

In [73]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[2.029e-01 8.500e-03 1.500e-03 2.960e-02 1.598e-01 5.916e-01 5.500
e-03
  5.000e-04 1.000e-04]]
Actual Class : 4
--------------------------------------------------
59 Text feature [values] present in test data point [True]
86 Text feature [ci] present in test data point [True]
95 Text feature [brca] present in test data point [True]
102 Text feature [ovarian] present in test data point [True]
115 Text feature [ligase] present in test data point [True]
118 Text feature [substitutions] present in test data point [True]
119 Text feature [dose] present in test data point [True]
121 Text feature [conformation] present in test data point [True]
125 Text feature [degradation] present in test data point [True]
126 Text feature [considered] present in test data point [True]
139 Text feature [showing] present in test data point [True]
141 Text feature [site] present in test data point [True]
148 Text feature [49] present in test data point [True]
149 Text feature [substitution] present in test data point [True]
150 Text feature [free] present in test data point [True]
155 Text feature [models] present in test data point [True]
163 Text feature [interaction] present in test data point [True]
168 Text feature [cause] present in test data point [True]
169 Text feature [deleterious] present in test data point [True]
171 Text feature [000] present in test data point [True]
176 Text feature [significant] present in test data point [True]
180 Text feature [95] present in test data point [True]
181 Text feature [confer] present in test data point [True]
182 Text feature [occurs] present in test data point [True]
```

```
185 Text feature [basis] present in test data point [True]
189 Text feature [expected] present in test data point [True]
196 Text feature [overall] present in test data point [True]
200 Text feature [classified] present in test data point [True]
201 Text feature [ring] present in test data point [True]
202 Text feature [57] present in test data point [True]
205 Text feature [odds] present in test data point [True]
210 Text feature [history] present in test data point [True]
216 Text feature [loss] present in test data point [True]
217 Text feature [42] present in test data point [True]
219 Text feature [groups] present in test data point [True]
220 Text feature [associated] present in test data point [True]
221 Text feature [development] present in test data point [True]
222 Text feature [isolated] present in test data point [True]
223 Text feature [phase] present in test data point [True]
224 Text feature [structural] present in test data point [True]
226 Text feature [characteristics] present in test data point [True]
228 Text feature [potential] present in test data point [True]
229 Text feature [suppression] present in test data point [True]
230 Text feature [43] present in test data point [True]
231 Text feature [particular] present in test data point [True]
232 Text feature [individuals] present in test data point [True]
233 Text feature [decreased] present in test data point [True]
234 Text feature [higher] present in test data point [True]
235 Text feature [studied] present in test data point [True]
236 Text feature [status] present in test data point [True]
239 Text feature [40] present in test data point [True]
240 Text feature [copy] present in test data point [True]
242 Text feature [affect] present in test data point [True]
246 Text feature [nucleotide] present in test data point [True]
248 Text feature [brca1] present in test data point [True]
249 Text feature [risk] present in test data point [True]
250 Text feature [model] present in test data point [True]
253 Text feature [time] present in test data point [True]
254 Text feature [34] present in test data point [True]
261 Text feature [38] present in test data point [True]
262 Text feature [54] present in test data point [True]
263 Text feature [46] present in test data point [True]
264 Text feature [score] present in test data point [True]
266 Text feature [selected] present in test data point [True]
267 Text feature [therapeutic] present in test data point [True]
270 Text feature [52] present in test data point [True]
271 Text feature [none] present in test data point [True]
272 Text feature [altered] present in test data point [True]
273 Text feature [determined] present in test data point [True]
275 Text feature [four] present in test data point [True]
276 Text feature [60] present in test data point [True]
278 Text feature [predicted] present in test data point [True]
279 Text feature [75] present in test data point [True]
280 Text feature [evidence] present in test data point [True]
285 Text feature [group] present in test data point [True]
286 Text feature [statistical] present in test data point [True]
288 Text feature [47] present in test data point [True]
289 Text feature [prior] present in test data point [True]
290 Text feature [five] present in test data point [True]
291 Text feature [due] present in test data point [True]
292 Text feature [72] present in test data point [True]
295 Text feature [observation] present in test data point [True]
296 Text feature [screening] present in test data point [True]
297 Text feature [difference] present in test data point [True]
298 Text feature [amplified] present in test data point [True]
299 Text feature [single] present in test data point [True]
300 Text feature [following] present in test data point [True]
301 Text feature [missense] present in test data point [True]
302 Text feature [induce] present in test data point [True]
303 Text feature [44] present in test data point [True]
305 Text feature [56] present in test data point [True]
308 Text feature [coding] present in test data point [True]
310 Text feature [green] present in test data point [True]
311 Text feature [population] present in test data point [True]
313 Text feature [times] present in test data point [True]
315 Text feature [residues] present in test data point [True]
318 Text feature [studies] present in test data point [True]
323 Text feature [51] present in test data point [True]
328 Text feature [35] present in test data point [True]
329 Text feature [information] present in test data point [True]
330 Text feature [structures] present in test data point [True]
```

```
331 Text feature [changes] present in test data point [True]
332 Text feature [active] present in test data point [True]
335 Text feature [regulation] present in test data point [True]
336 Text feature [lower] present in test data point [True]
337 Text feature [30] present in test data point [True]
346 Text feature [23] present in test data point [True]
347 Text feature [genomic] present in test data point [True]
349 Text feature [involved] present in test data point [True]
350 Text feature [response] present in test data point [True]
351 Text feature [set] present in test data point [True]
352 Text feature [200] present in test data point [True]
355 Text feature [45] present in test data point [True]
356 Text feature [six] present in test data point [True]
358 Text feature [direct] present in test data point [True]
360 Text feature [identified] present in test data point [True]
363 Text feature [important] present in test data point [True]
367 Text feature [american] present in test data point [True]
370 Text feature [given] present in test data point [True]
371 Text feature [number] present in test data point [True]
372 Text feature [36] present in test data point [True]
374 Text feature [type] present in test data point [True]
380 Text feature [required] present in test data point [True]
383 Text feature [strand] present in test data point [True]
384 Text feature [acids] present in test data point [True]
386 Text feature [differences] present in test data point [True]
387 Text feature [gel] present in test data point [True]
389 Text feature [distinct] present in test data point [True]
390 Text feature [significance] present in test data point [True]
395 Text feature [frequency] present in test data point [True]
398 Text feature [classification] present in test data point [True]
399 Text feature [resulted] present in test data point [True]
400 Text feature [selection] present in test data point [True]
402 Text feature [family] present in test data point [True]
404 Text feature [approach] present in test data point [True]
406 Text feature [versus] present in test data point [True]
408 Text feature [primer] present in test data point [True]
409 Text feature [association] present in test data point [True]
415 Text feature [respectively] present in test data point [True]
417 Text feature [spectrum] present in test data point [True]
420 Text feature [region] present in test data point [True]
422 Text feature [binding] present in test data point [True]
423 Text feature [general] present in test data point [True]
424 Text feature [proportion] present in test data point [True]
428 Text feature [added] present in test data point [True]
429 Text feature [database] present in test data point [True]
430 Text feature [applied] present in test data point [True]
434 Text feature [targeting] present in test data point [True]
436 Text feature [occur] present in test data point [True]
437 Text feature [calculated] present in test data point [True]
439 Text feature [70] present in test data point [True]
440 Text feature [open] present in test data point [True]
441 Text feature [structure] present in test data point [True]
444 Text feature [normal] present in test data point [True]
445 Text feature [05] present in test data point [True]
449 Text feature [25] present in test data point [True]
451 Text feature [embryonic] present in test data point [True]
452 Text feature [26] present in test data point [True]
453 Text feature [mutant] present in test data point [True]
456 Text feature [shown] present in test data point [True]
457 Text feature [terminal] present in test data point [True]
459 Text feature [since] present in test data point [True]
460 Text feature [amino] present in test data point [True]
462 Text feature [factor] present in test data point [True]
465 Text feature [including] present in test data point [True]
466 Text feature [would] present in test data point [True]
467 Text feature [conserved] present in test data point [True]
469 Text feature [well] present in test data point [True]
471 Text feature [measured] present in test data point [True]
473 Text feature [methods] present in test data point [True]
474 Text feature [evaluated] present in test data point [True]
476 Text feature [sequence] present in test data point [True]
478 Text feature [lines] present in test data point [True]
479 Text feature [41] present in test data point [True]
480 Text feature [multiple] present in test data point [True]
481 Text feature [90] present in test data point [True]
482 Text feature [12] present in test data point [True]
483 Text feature [20] present in test data point [True]
```

```
484 Text feature [domains] present in test data point [True]
485 Text feature [relatively] present in test data point [True]
487 Text feature [form] present in test data point [True]
490 Text feature [reported] present in test data point [True]
492 Text feature [value] present in test data point [True]
493 Text feature [17] present in test data point [True]
494 Text feature [19] present in test data point [True]
495 Text feature [age] present in test data point [True]
496 Text feature [subsequent] present in test data point [True]
497 Text feature [thus] present in test data point [True]
498 Text feature [ml] present in test data point [True]
499 Text feature [least] present in test data point [True]
Out of the top  500  features  190 are present in query point
```

4.4. Linear Support Vector Machines

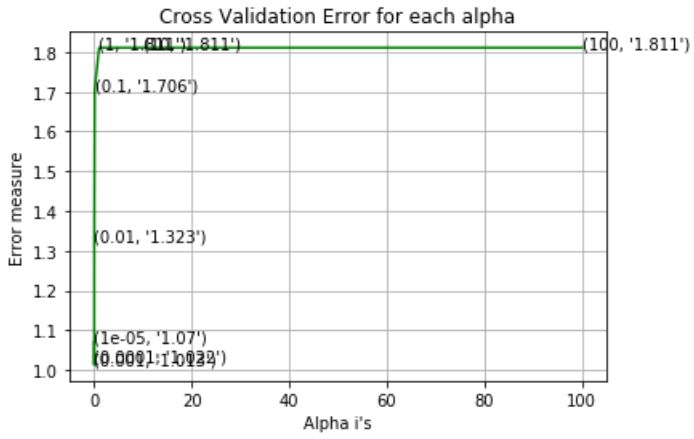
4.4.1. Hyper paramter tuning

In [74]:

```
# 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))

# Variables that will be used in the end to make comparison table of all models
svm_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps
=1e-15)
svm_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
svm_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e
-15)
```

```
for C = 1e-05
Log Loss : 1.0703870826326363
for C = 0.0001
Log Loss : 1.0224886859031384
for C = 0.001
Log Loss : 1.013402209329118
for C = 0.01
Log Loss : 1.323099642450596
for C = 0.1
Log Loss : 1.7056716101660465
for C = 1
Log Loss : 1.8106680361458611
for C = 10
Log Loss : 1.8106682603992044
for C = 100
Log Loss : 1.8106681990827265
```



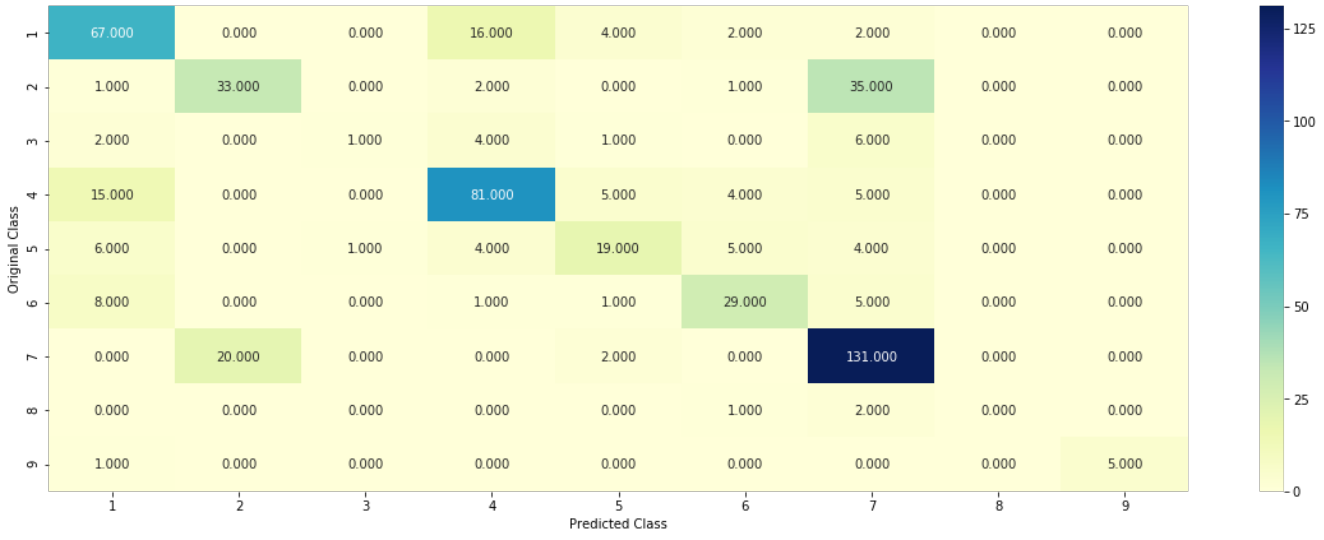

```
For values of best alpha =  0.001 The train log loss is: 0.6167958891284219
For values of best alpha =  0.001 The cross validation log loss is: 1.013402209329118
For values of best alpha =  0.001 The test log loss is: 1.1056289144530362
```

4.4.2. Testing model with best hyper parameters

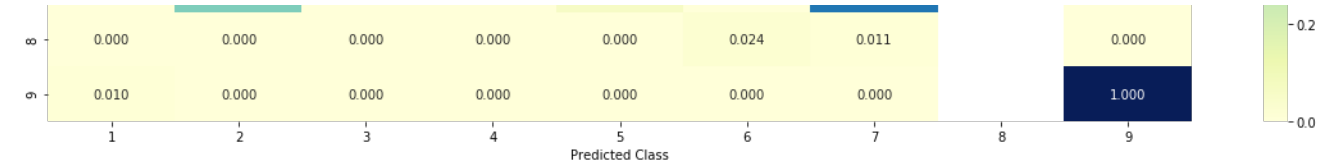
In [75]:

```
# 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)

clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

# Variables that will be used in the end to make comparison table of models
svm_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])*1
00
```

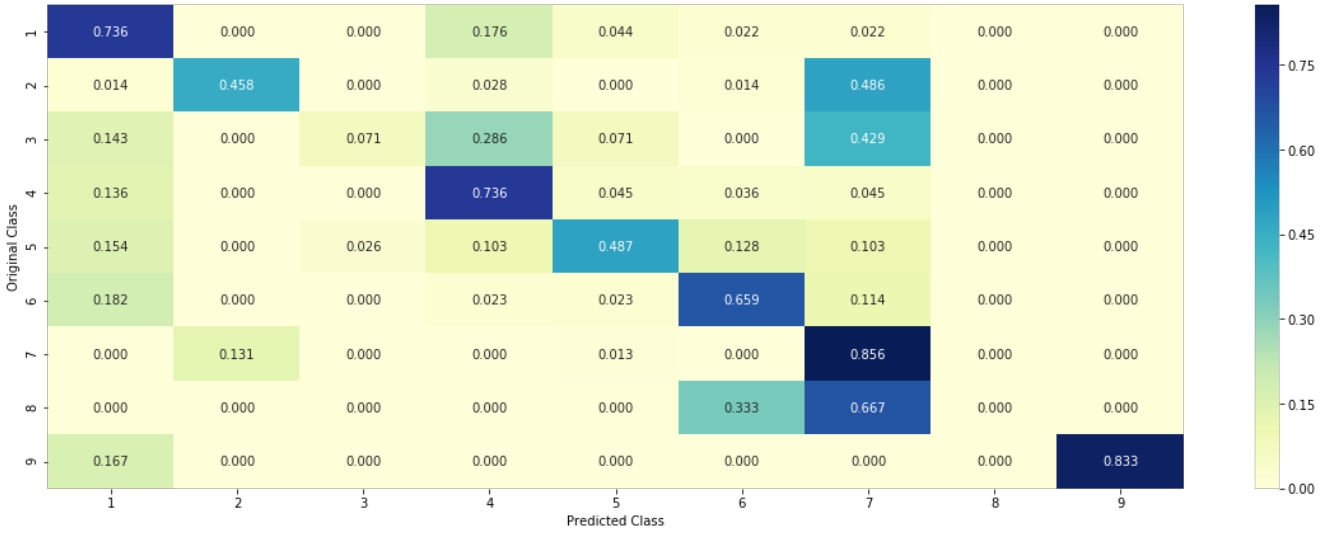
```
Log loss : 1.013402209329118
Number of mis-classified points : 0.31203007518796994
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [76]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
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: [[0.079  0.0317 0.0098 0.792  0.0295 0.0121 0.0428 0.0015 0.0016]]
Actual Class : 4
--------------------------------------------------
20 Text feature [suppressor] present in test data point [True]
21 Text feature [missense] present in test data point [True]
22 Text feature [iii] present in test data point [True]
30 Text feature [show] present in test data point [True]
32 Text feature [unable] present in test data point [True]
233 Text feature [mm] present in test data point [True]
234 Text feature [1998] present in test data point [True]
238 Text feature [see] present in test data point [True]
239 Text feature [stability] present in test data point [True]
244 Text feature [ca] present in test data point [True]
245 Text feature [suggesting] present in test data point [True]
247 Text feature [germline] present in test data point [True]
330 Text feature [cause] present in test data point [True]
334 Text feature [families] present in test data point [True]
340 Text feature [groups] present in test data point [True]
341 Text feature [tagged] present in test data point [True]
```

```
347 Text feature [express] present in test data point [True]
351 Text feature [relatively] present in test data point [True]
353 Text feature [consequences] present in test data point [True]
354 Text feature [comparison] present in test data point [True]
356 Text feature [deletion] present in test data point [True]
360 Text feature [affected] present in test data point [True]
361 Text feature [cycle] present in test data point [True]
362 Text feature [high] present in test data point [True]
421 Text feature [university] present in test data point [True]
422 Text feature [alterations] present in test data point [True]
423 Text feature [indicate] present in test data point [True]
425 Text feature [represent] present in test data point [True]
426 Text feature [splice] present in test data point [True]
428 Text feature [intermediate] present in test data point [True]
430 Text feature [protein] present in test data point [True]
431 Text feature [activity] present in test data point [True]
433 Text feature [cases] present in test data point [True]
434 Text feature [red] present in test data point [True]
435 Text feature [nature] present in test data point [True]
436 Text feature [loss] present in test data point [True]
437 Text feature [motif] present in test data point [True]
438 Text feature [bind] present in test data point [True]
441 Text feature [linked] present in test data point [True]
443 Text feature [contribute] present in test data point [True]
444 Text feature [cannot] present in test data point [True]
446 Text feature [age] present in test data point [True]
452 Text feature [primary] present in test data point [True]
453 Text feature [functional] present in test data point [True]
454 Text feature [able] present in test data point [True]
457 Text feature [green] present in test data point [True]
458 Text feature [appears] present in test data point [True]
460 Text feature [genomic] present in test data point [True]
462 Text feature [directly] present in test data point [True]
464 Text feature [skin] present in test data point [True]
468 Text feature [individuals] present in test data point [True]
470 Text feature [indeed] present in test data point [True]
473 Text feature [kinases] present in test data point [True]
474 Text feature [blood] present in test data point [True]
475 Text feature [indicating] present in test data point [True]
477 Text feature [mutants] present in test data point [True]
478 Text feature [bound] present in test data point [True]
480 Text feature [shows] present in test data point [True]
481 Text feature [nuclear] present in test data point [True]
Out of the top  500  features  59 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [77]:

```
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: [[0.1045 0.0371 0.008  0.066  0.1225 0.6097 0.0487 0.0016 0.0018]]
Actual Class : 4
--------------------------------------------------
3 Text feature [values] present in test data point [True]
4 Text feature [ligase] present in test data point [True]
5 Text feature [showing] present in test data point [True]
6 Text feature [brca] present in test data point [True]
9 Text feature [substitutions] present in test data point [True]
10 Text feature [classified] present in test data point [True]
122 Text feature [site] present in test data point [True]
123 Text feature [substitution] present in test data point [True]
```

```
125 Text feature [interaction] present in test data point [True]
126 Text feature [conformation] present in test data point [True]
130 Text feature [ring] present in test data point [True]
132 Text feature [expected] present in test data point [True]
133 Text feature [confer] present in test data point [True]
135 Text feature [deleterious] present in test data point [True]
137 Text feature [suppression] present in test data point [True]
138 Text feature [models] present in test data point [True]
158 Text feature [odds] present in test data point [True]
163 Text feature [49] present in test data point [True]
164 Text feature [57] present in test data point [True]
165 Text feature [dose] present in test data point [True]
166 Text feature [ci] present in test data point [True]
167 Text feature [studied] present in test data point [True]
168 Text feature [cause] present in test data point [True]
178 Text feature [ovarian] present in test data point [True]
179 Text feature [000] present in test data point [True]
180 Text feature [basis] present in test data point [True]
181 Text feature [free] present in test data point [True]
182 Text feature [characteristics] present in test data point [True]
183 Text feature [history] present in test data point [True]
184 Text feature [significant] present in test data point [True]
185 Text feature [none] present in test data point [True]
186 Text feature [model] present in test data point [True]
189 Text feature [loss] present in test data point [True]
191 Text feature [therapeutic] present in test data point [True]
192 Text feature [brca1] present in test data point [True]
193 Text feature [overall] present in test data point [True]
194 Text feature [potential] present in test data point [True]
195 Text feature [occurs] present in test data point [True]
196 Text feature [34] present in test data point [True]
197 Text feature [lower] present in test data point [True]
199 Text feature [degradation] present in test data point [True]
200 Text feature [associated] present in test data point [True]
201 Text feature [predicted] present in test data point [True]
203 Text feature [95] present in test data point [True]
233 Text feature [individuals] present in test data point [True]
234 Text feature [56] present in test data point [True]
235 Text feature [affect] present in test data point [True]
237 Text feature [status] present in test data point [True]
239 Text feature [green] present in test data point [True]
240 Text feature [times] present in test data point [True]
241 Text feature [considered] present in test data point [True]
242 Text feature [prior] present in test data point [True]
245 Text feature [43] present in test data point [True]
248 Text feature [altered] present in test data point [True]
249 Text feature [binding] present in test data point [True]
250 Text feature [active] present in test data point [True]
251 Text feature [significance] present in test data point [True]
253 Text feature [given] present in test data point [True]
254 Text feature [isolated] present in test data point [True]
256 Text feature [tumors] present in test data point [True]
257 Text feature [structural] present in test data point [True]
259 Text feature [four] present in test data point [True]
262 Text feature [would] present in test data point [True]
263 Text feature [acids] present in test data point [True]
264 Text feature [risk] present in test data point [True]
265 Text feature [single] present in test data point [True]
266 Text feature [five] present in test data point [True]
267 Text feature [screening] present in test data point [True]
268 Text feature [38] present in test data point [True]
270 Text feature [population] present in test data point [True]
271 Text feature [higher] present in test data point [True]
273 Text feature [evidence] present in test data point [True]
274 Text feature [development] present in test data point [True]
275 Text feature [40] present in test data point [True]
276 Text feature [42] present in test data point [True]
277 Text feature [studies] present in test data point [True]
278 Text feature [60] present in test data point [True]
279 Text feature [missense] present in test data point [True]
281 Text feature [changes] present in test data point [True]
282 Text feature [particular] present in test data point [True]
283 Text feature [52] present in test data point [True]
286 Text feature [54] present in test data point [True]
287 Text feature [set] present in test data point [True]
288 Text feature [35] present in test data point [True]
289 Text feature [neutral] present in test data point [True]
```

```
290 Text feature [45] present in test data point [True]
292 Text feature [selection] present in test data point [True]
294 Text feature [41] present in test data point [True]
296 Text feature [spectrum] present in test data point [True]
297 Text feature [23] present in test data point [True]
300 Text feature [nucleotide] present in test data point [True]
301 Text feature [targeted] present in test data point [True]
302 Text feature [occur] present in test data point [True]
303 Text feature [applied] present in test data point [True]
304 Text feature [amino] present in test data point [True]
305 Text feature [identified] present in test data point [True]
306 Text feature [family] present in test data point [True]
307 Text feature [general] present in test data point [True]
308 Text feature [72] present in test data point [True]
309 Text feature [observations] present in test data point [True]
310 Text feature [approach] present in test data point [True]
311 Text feature [domains] present in test data point [True]
312 Text feature [targeting] present in test data point [True]
340 Text feature [thus] present in test data point [True]
341 Text feature [six] present in test data point [True]
343 Text feature [direct] present in test data point [True]
345 Text feature [decreased] present in test data point [True]
346 Text feature [reported] present in test data point [True]
348 Text feature [sequence] present in test data point [True]
349 Text feature [majority] present in test data point [True]
351 Text feature [36] present in test data point [True]
356 Text feature [distribution] present in test data point [True]
359 Text feature [coding] present in test data point [True]
360 Text feature [frequency] present in test data point [True]
363 Text feature [involved] present in test data point [True]
364 Text feature [clinically] present in test data point [True]
369 Text feature [seven] present in test data point [True]
370 Text feature [required] present in test data point [True]
371 Text feature [05] present in test data point [True]
372 Text feature [many] present in test data point [True]
376 Text feature [tumor] present in test data point [True]
377 Text feature [groups] present in test data point [True]
378 Text feature [information] present in test data point [True]
379 Text feature [acid] present in test data point [True]
380 Text feature [group] present in test data point [True]
383 Text feature [51] present in test data point [True]
384 Text feature [response] present in test data point [True]
385 Text feature [01] present in test data point [True]
387 Text feature [examined] present in test data point [True]
388 Text feature [versus] present in test data point [True]
389 Text feature [type] present in test data point [True]
390 Text feature [due] present in test data point [True]
394 Text feature [phase] present in test data point [True]
395 Text feature [score] present in test data point [True]
396 Text feature [44] present in test data point [True]
397 Text feature [factor] present in test data point [True]
398 Text feature [47] present in test data point [True]
401 Text feature [number] present in test data point [True]
403 Text feature [showed] present in test data point [True]
406 Text feature [normal] present in test data point [True]
410 Text feature [differences] present in test data point [True]
412 Text feature [time] present in test data point [True]
413 Text feature [database] present in test data point [True]
414 Text feature [methods] present in test data point [True]
415 Text feature [classification] present in test data point [True]
417 Text feature [developed] present in test data point [True]
418 Text feature [26] present in test data point [True]
419 Text feature [statistical] present in test data point [True]
423 Text feature [likely] present in test data point [True]
426 Text feature [determine] present in test data point [True]
427 Text feature [bind] present in test data point [True]
428 Text feature [form] present in test data point [True]
429 Text feature [61] present in test data point [True]
431 Text feature [proportion] present in test data point [True]
433 Text feature [identification] present in test data point [True]
435 Text feature [hypothesis] present in test data point [True]
436 Text feature [genomic] present in test data point [True]
437 Text feature [total] present in test data point [True]
439 Text feature [ratio] present in test data point [True]
441 Text feature [observation] present in test data point [True]
443 Text feature [test] present in test data point [True]
444 Text feature [allele] present in test data point [True]
```

```
445 Text feature [purified] present in test data point [True]
447 Text feature [value] present in test data point [True]
448 Text feature [types] present in test data point [True]
449 Text feature [generated] present in test data point [True]
450 Text feature [46] present in test data point [True]
451 Text feature [reduced] present in test data point [True]
453 Text feature [copy] present in test data point [True]
457 Text feature [fact] present in test data point [True]
458 Text feature [require] present in test data point [True]
459 Text feature [variants] present in test data point [True]
460 Text feature [mediated] present in test data point [True]
461 Text feature [american] present in test data point [True]
462 Text feature [interestingly] present in test data point [True]
465 Text feature [relatively] present in test data point [True]
466 Text feature [open] present in test data point [True]
468 Text feature [resulted] present in test data point [True]
469 Text feature [two] present in test data point [True]
470 Text feature [selected] present in test data point [True]
471 Text feature [distinct] present in test data point [True]
472 Text feature [200] present in test data point [True]
473 Text feature [strand] present in test data point [True]
474 Text feature [association] present in test data point [True]
476 Text feature [combined] present in test data point [True]
477 Text feature [20] present in test data point [True]
478 Text feature [determined] present in test data point [True]
480 Text feature [well] present in test data point [True]
481 Text feature [whether] present in test data point [True]
483 Text feature [included] present in test data point [True]
484 Text feature [null] present in test data point [True]
485 Text feature [calculated] present in test data point [True]
486 Text feature [27] present in test data point [True]
487 Text feature [gel] present in test data point [True]
489 Text feature [including] present in test data point [True]
490 Text feature [following] present in test data point [True]
492 Text feature [21] present in test data point [True]
494 Text feature [difference] present in test data point [True]
496 Text feature [use] present in test data point [True]
498 Text feature [75] present in test data point [True]
Out of the top  500  features  200 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [78]:

```
# --------------------------------
# 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))

# Variables that will be used in the end to make comparison table of all models
rf_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=
1e-15)
rf_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
rf_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-
15)
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1942273221373252
for n_estimators = 100 and max depth =  10
Log Loss : 1.2179232517744036
for n_estimators = 200 and max depth =  5
Log Loss : 1.1833746167480466
for n_estimators = 200 and max depth =  10
Log Loss : 1.2083480275351055
for n_estimators = 500 and max depth =  5
Log Loss : 1.1759122386561291
for n_estimators = 500 and max depth =  10
```

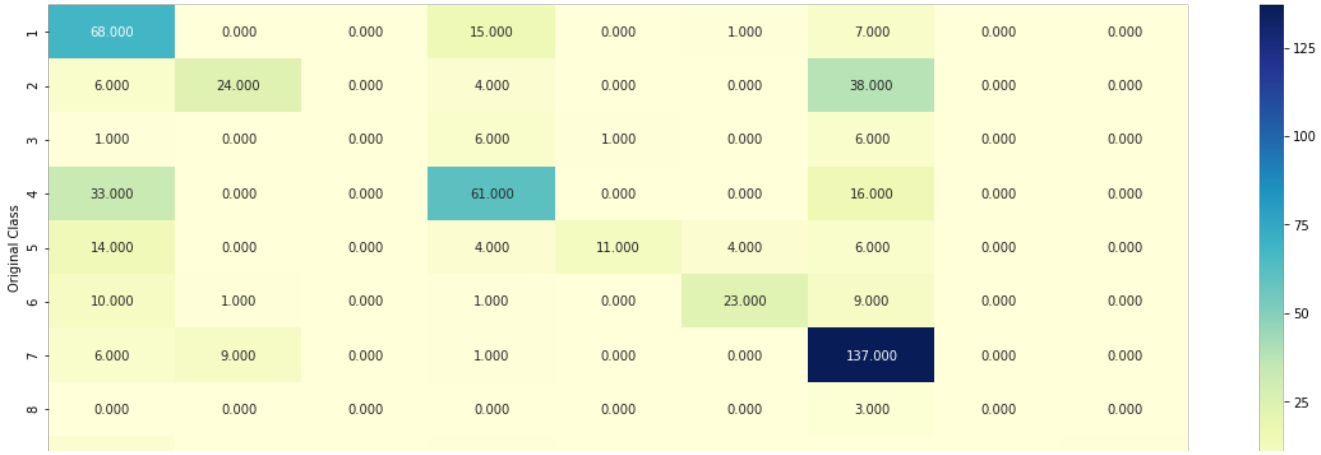
```
Log Loss : 1.2046779372703127
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1734770059012314
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2046247798706662
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1744399082999013
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2042149064193404
For values of best estimator =  1000 The train log loss is: 0.875351337805046
For values of best estimator =  1000 The cross validation log loss is: 1.1734770059012316
For values of best estimator =  1000 The test log loss is: 1.2136206634420559
```

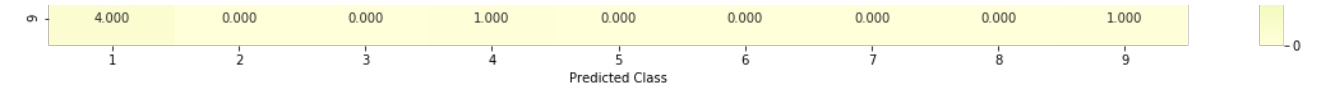
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [79]:

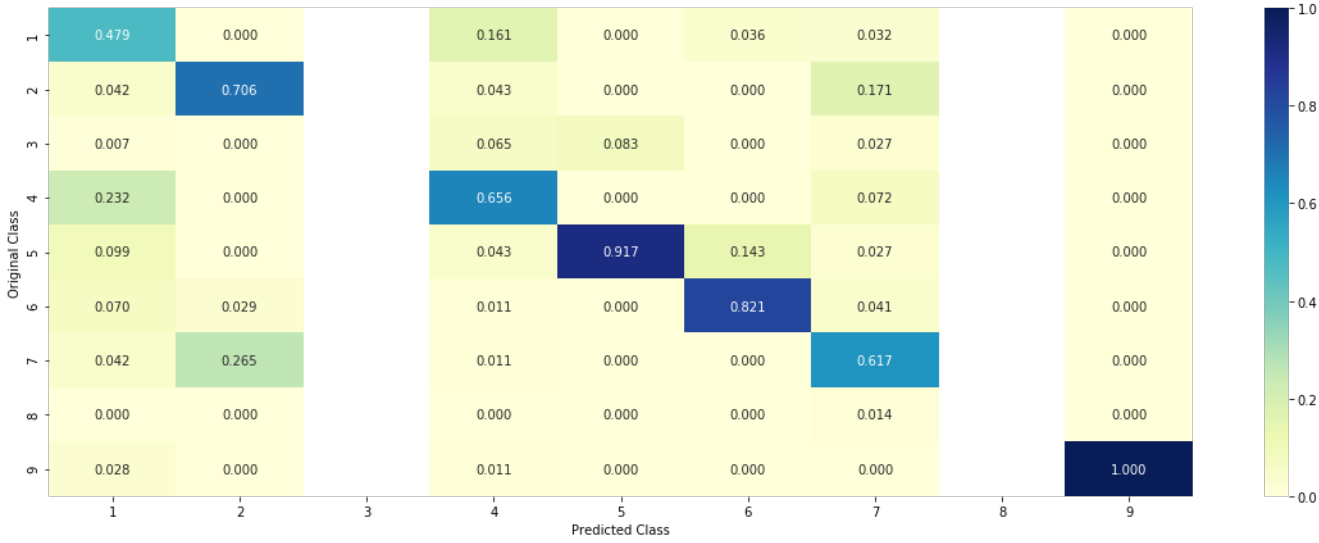
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, 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)

clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

# Variables that will be used in the end to make comparison table of models
rf_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
cv_y))/cv_y.shape[0])*100
```

```
Log loss : 1.173477005901232
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```

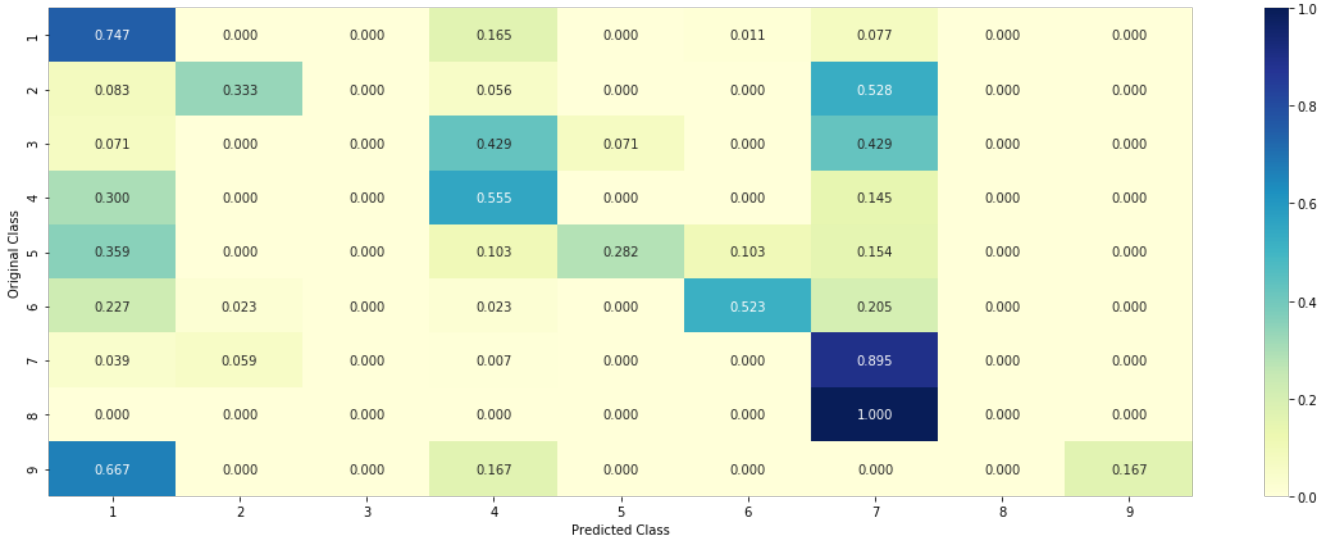





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [80]:

```
# 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: [[0.2924 0.0197 0.0227 0.4368 0.0912 0.0875 0.0345 0.0049 0.0104]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
5 Text feature [missense] present in test data point [True]
6 Text feature [function] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
9 Text feature [suppressor] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
11 Text feature [brca1] present in test data point [True]
13 Text feature [loss] present in test data point [True]
14 Text feature [inhibitor] present in test data point [True]
19 Text feature [pathogenic] present in test data point [True]
21 Text feature [cells] present in test data point [True]
22 Text feature [protein] present in test data point [True]
23 Text feature [stability] present in test data point [True]
25 Text feature [variants] present in test data point [True]
27 Text feature [odds] present in test data point [True]
28 Text feature [deleterious] present in test data point [True]
36 Text feature [functional] present in test data point [True]
42 Text feature [classified] present in test data point [True]
43 Text feature [cell] present in test data point [True]
46 Text feature [proteins] present in test data point [True]
47 Text feature [kinases] present in test data point [True]
48 Text feature [neutral] present in test data point [True]
49 Text feature [brca2] present in test data point [True]
53 Text feature [inhibition] present in test data point [True]
57 Text feature [patients] present in test data point [True]
59 Text feature [clinical] present in test data point [True]
60 Text feature [classification] present in test data point [True]
68 Text feature [expressing] present in test data point [True]
70 Text feature [proliferation] present in test data point [True]
82 Text feature [dna] present in test data point [True]
85 Text feature [type] present in test data point [True]
88 Text feature [splice] present in test data point [True]
90 Text feature [expression] present in test data point [True]
92 Text feature [variant] present in test data point [True]
93 Text feature [21] present in test data point [True]
94 Text feature [assays] present in test data point [True]
96 Text feature [predicted] present in test data point [True]
97 Text feature [information] present in test data point [True]
99 Text feature [families] present in test data point [True]
Out of the top  100  features  40 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [81]:

```
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: [[0.1797 0.004  0.009  0.03   0.1053 0.6579 0.0094 0.0025 0.0021]]
Actuall Class : 4
--------------------------------------------------
4 Text feature [activation] present in test data point [True]
5 Text feature [missense] present in test data point [True]
6 Text feature [function] present in test data point [True]
```

```
9 Text feature [suppressor] present in test data point [True]
11 Text feature [brca1] present in test data point [True]
13 Text feature [loss] present in test data point [True]
19 Text feature [pathogenic] present in test data point [True]
20 Text feature [brca] present in test data point [True]
21 Text feature [cells] present in test data point [True]
22 Text feature [protein] present in test data point [True]
25 Text feature [variants] present in test data point [True]
27 Text feature [odds] present in test data point [True]
28 Text feature [deleterious] present in test data point [True]
31 Text feature [repair] present in test data point [True]
33 Text feature [growth] present in test data point [True]
34 Text feature [57] present in test data point [True]
36 Text feature [functional] present in test data point [True]
40 Text feature [yeast] present in test data point [True]
42 Text feature [classified] present in test data point [True]
43 Text feature [cell] present in test data point [True]
46 Text feature [proteins] present in test data point [True]
48 Text feature [neutral] present in test data point [True]
49 Text feature [brca2] present in test data point [True]
52 Text feature [therapeutic] present in test data point [True]
54 Text feature [ovarian] present in test data point [True]
55 Text feature [brct] present in test data point [True]
57 Text feature [patients] present in test data point [True]
59 Text feature [clinical] present in test data point [True]
60 Text feature [classification] present in test data point [True]
64 Text feature [null] present in test data point [True]
68 Text feature [expressing] present in test data point [True]
74 Text feature [lines] present in test data point [True]
75 Text feature [damage] present in test data point [True]
76 Text feature [sensitivity] present in test data point [True]
80 Text feature [mammalian] present in test data point [True]
81 Text feature [ring] present in test data point [True]
82 Text feature [dna] present in test data point [True]
83 Text feature [vus] present in test data point [True]
85 Text feature [type] present in test data point [True]
87 Text feature [functions] present in test data point [True]
88 Text feature [splice] present in test data point [True]
89 Text feature [serum] present in test data point [True]
90 Text feature [expression] present in test data point [True]
91 Text feature [response] present in test data point [True]
92 Text feature [variant] present in test data point [True]
93 Text feature [21] present in test data point [True]
94 Text feature [assays] present in test data point [True]
96 Text feature [predicted] present in test data point [True]
97 Text feature [information] present in test data point [True]
99 Text feature [families] present in test data point [True]
Out of the top  100  features  50 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [82]:

```
# -----------------------------------
# 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))

# Variables that will be used in the end to make comparison table of all models
rf_response_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding),
labels=clf.classes_, eps=1e-15)
rf_response_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.classes_, ep
s=1e-15)
rf_response_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=clf.classe
s_, eps=1e-15)
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.138277045678003
for n_estimators = 10 and max depth =  3
Log Loss : 1.6145203531241525
for n_estimators = 10 and max depth =  5
Log Loss : 1.3281557361879377
```

```
Log Loss : 1.5261557561879577
for n_estimators = 10 and max depth =  10
Log Loss : 1.6915060445081311
for n_estimators = 50 and max depth =  2
Log Loss : 1.5811842060032495
for n_estimators = 50 and max depth =  3
Log Loss : 1.4171909601318506
for n_estimators = 50 and max depth =  5
Log Loss : 1.234039883229894
for n_estimators = 50 and max depth =  10
Log Loss : 1.8079398553099557
for n_estimators = 100 and max depth =  2
Log Loss : 1.437788970485041
for n_estimators = 100 and max depth =  3
Log Loss : 1.3866356958512822
for n_estimators = 100 and max depth =  5
Log Loss : 1.2015508204999736
for n_estimators = 100 and max depth =  10
Log Loss : 1.6920194921882468
for n_estimators = 200 and max depth =  2
Log Loss : 1.4989972685272175
for n_estimators = 200 and max depth =  3
Log Loss : 1.400968224246809
for n_estimators = 200 and max depth =  5
Log Loss : 1.2669196338543443
for n_estimators = 200 and max depth =  10
Log Loss : 1.6662465631082106
for n_estimators = 500 and max depth =  2
Log Loss : 1.5445219896092734
for n_estimators = 500 and max depth =  3
Log Loss : 1.4543997311607728
for n_estimators = 500 and max depth =  5
Log Loss : 1.317931266895718
for n_estimators = 500 and max depth =  10
Log Loss : 1.6482552554657741
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5402013789122908
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4611030317540632
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3098966687879894
for n_estimators = 1000 and max depth =  10
Log Loss : 1.664613818505019
For values of best alpha =  100 The train log loss is: 0.052791703424134445
For values of best alpha =  100 The cross validation log loss is: 1.2015508204999736
For values of best alpha =  100 The test log loss is: 1.2797335308960462
```

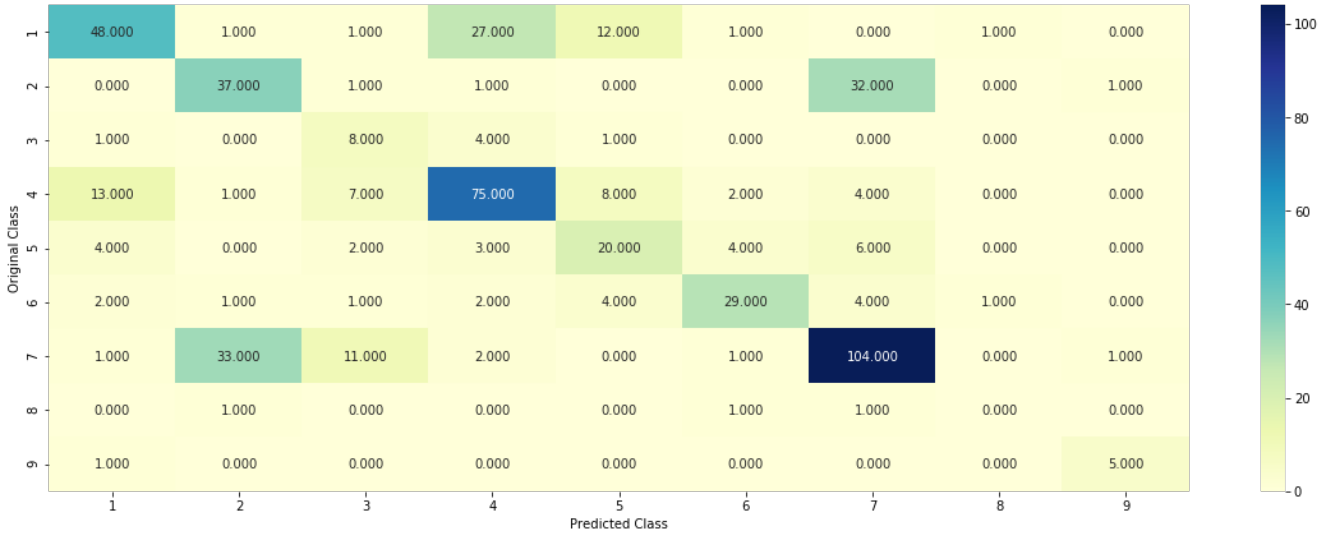
4.5.4. Testing model with best hyper parameters (Response Coding)

In [83]:

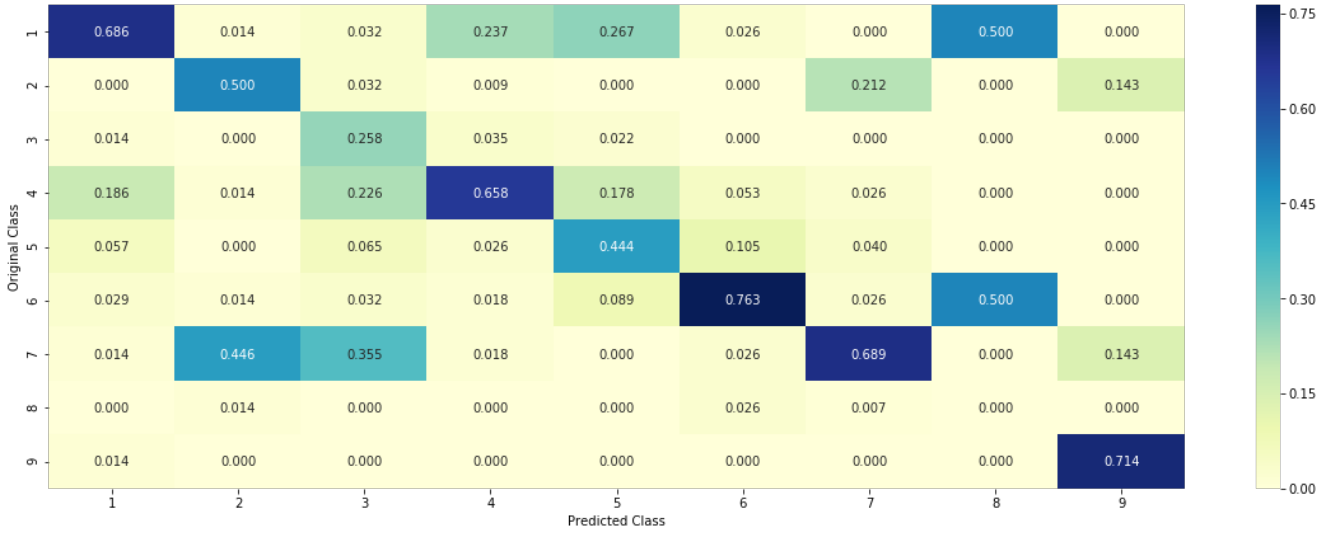
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

```
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)

clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

# Variables that will be used in the end to make comparison table of models
rf_response_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y))/cv_y.s
hape[0])*100
```

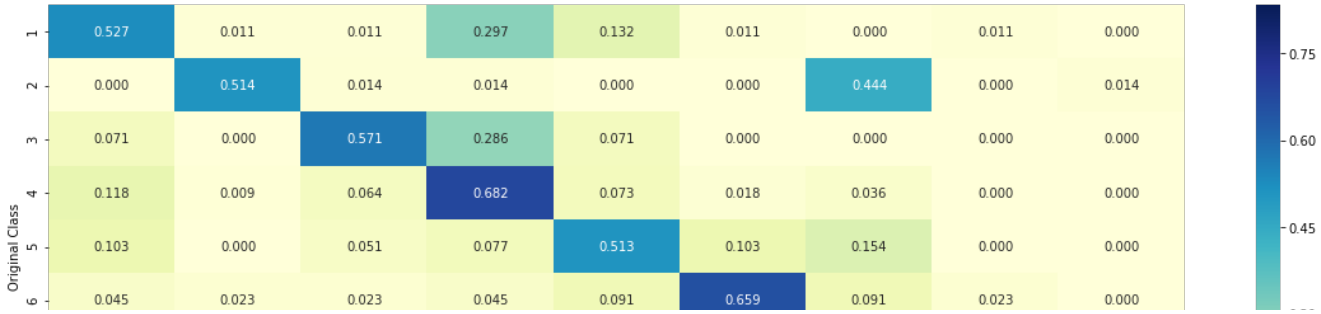
```
Log loss : 1.2015508204999734
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```

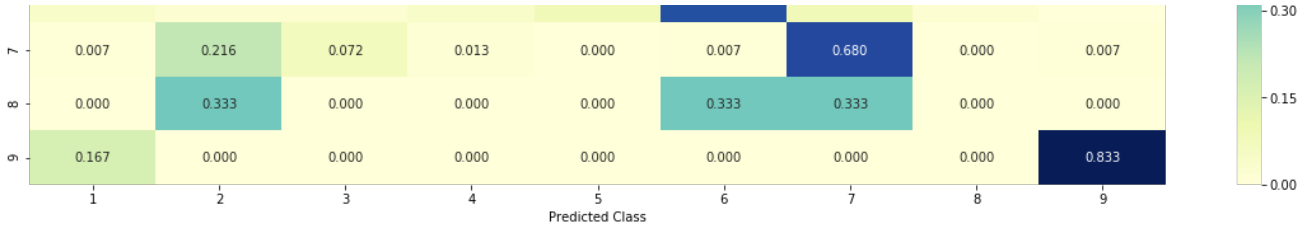



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [84]:

```
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: [[0.1524 0.0221 0.183  0.4789 0.0281 0.0731 0.0136 0.0228 0.0261]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [85]:

```
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: [[0.0278 0.0034 0.0188 0.0295 0.2406 0.668  0.0026 0.005  0.0044]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [86]:

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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


# 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_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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


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

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

```
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking 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
```

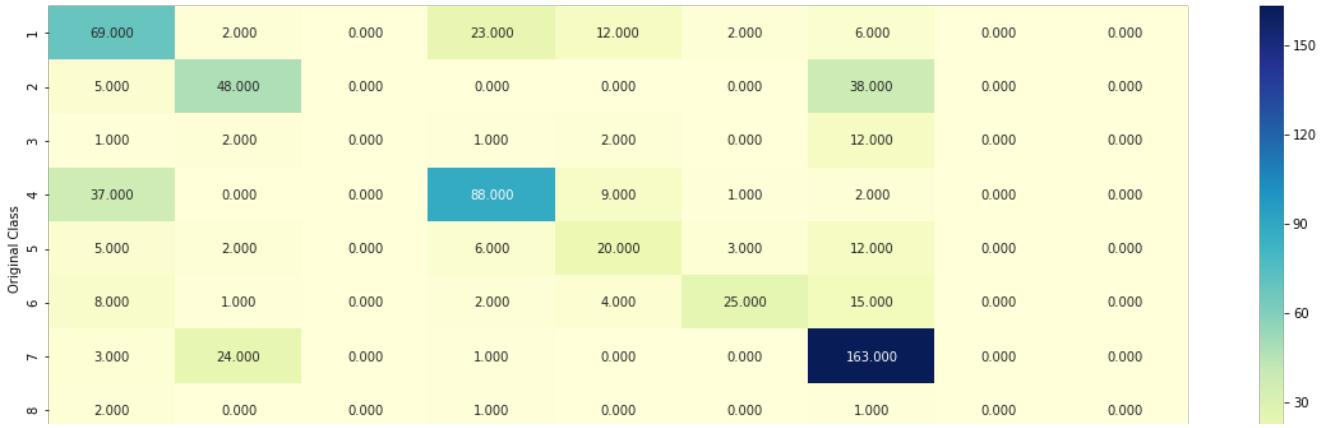
```
Logistic Regression :  Log Loss: 1.01
Support vector machines : Log Loss: 1.81
Naive Bayes : Log Loss: 1.16
---------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.485
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.117
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.298
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.705
```

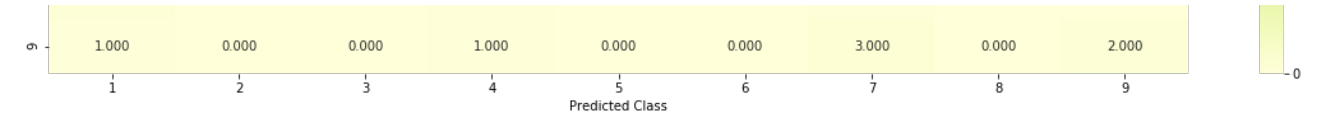
4.7.2 testing the model with the best hyper parameters

In [87]:

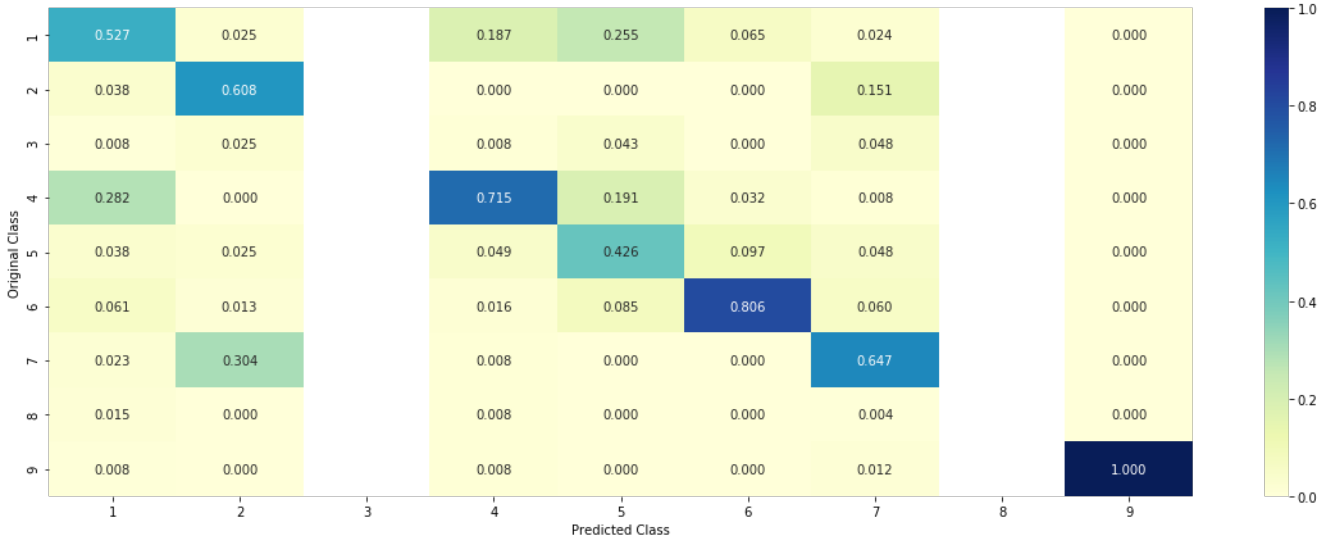
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error1 = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error2 = 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))

# Variables that will be used in the end to make comparison table of all models
stack_train = log_error
stack_cv = log_error1
stack_test = log_error2
stack_misclassified = (np.count_nonzero((sclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0
])*100
```

```
Log loss (train) on the stacking classifier : 0.5481568365472762
Log loss (CV) on the stacking classifier : 0.5481568365472762
Log loss (test) on the stacking classifier : 0.5481568365472762
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

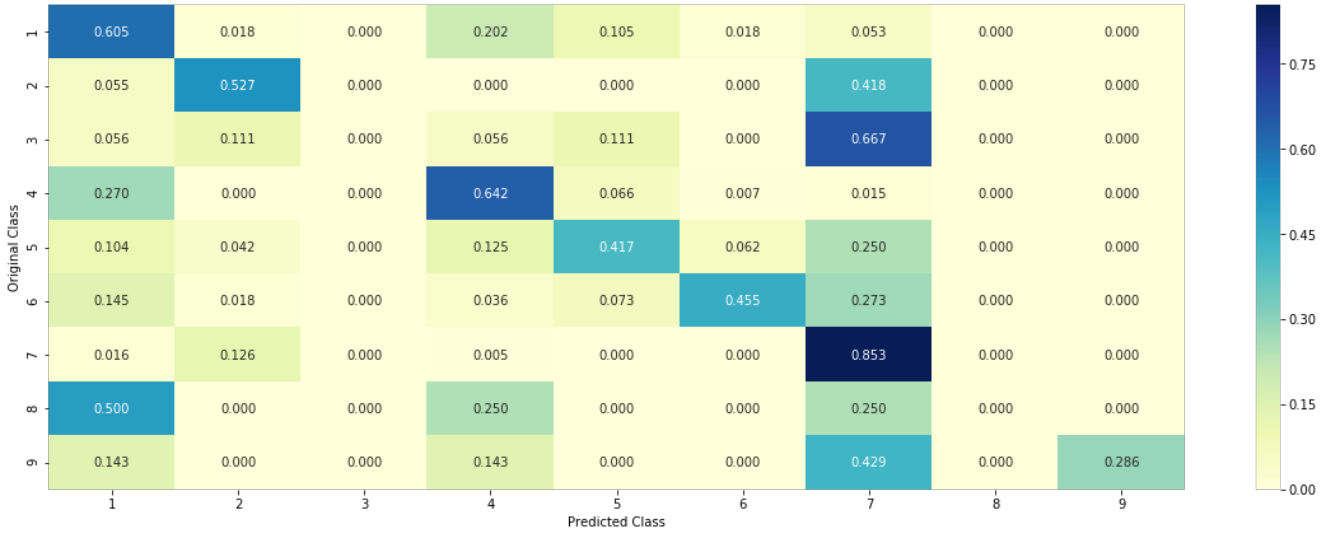





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



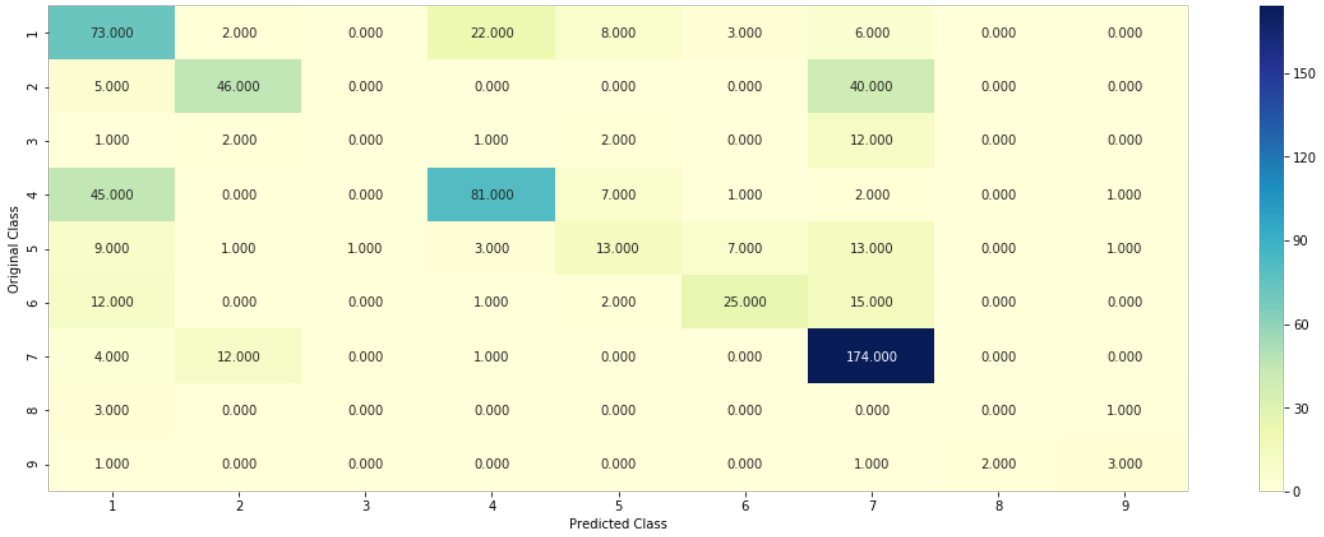
4.7.3 Maximum Voting classifier

In [88]:

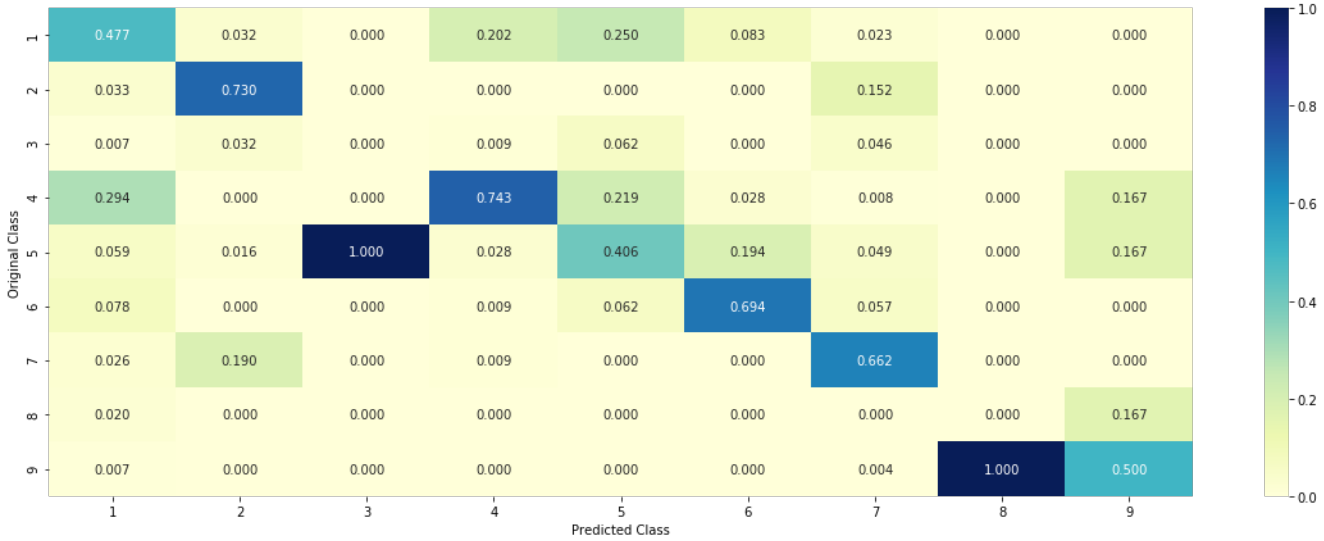
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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))

# Variables that will be used in the end to make comparison table of all models
mvc_train = log_loss(train_y, vclf.predict_proba(train_x_onehotCoding))
```

```
mvc_cv = log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
mvc_test = log_loss(test_y, vclf.predict_proba(test_x_onehotCoding))
mvc_misclassified = (np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
*100
```

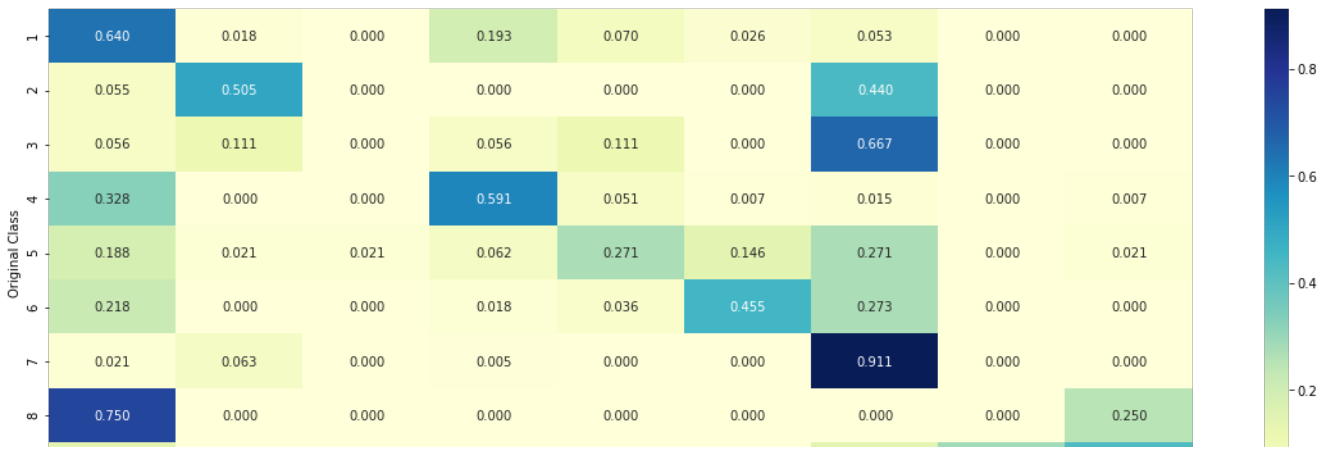
```
Log loss (train) on the VotingClassifier : 0.8392747499944152
Log loss (CV) on the VotingClassifier : 1.1682181117847334
Log loss (test) on the VotingClassifier : 1.219016527827425
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

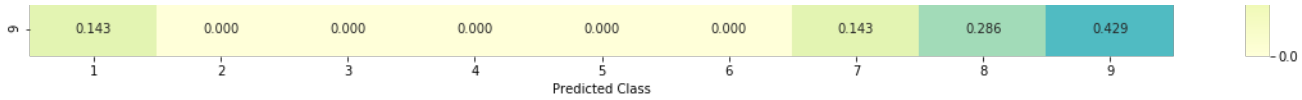



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





CONCLUSION

(a). Procedure Followed :-

STEP 1:- Replace TfidfVectorizer() by TfidfVectorizer(max_features=1000) in the one hot encoding section of text feature to get top 1000 words based of tf-idf values

(b).Table (Model Performances):-

In [89]:

```
# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names =['Naive Bayes','K-Nearest Neighbour','LR With Class Balancing',\
        'LR Without Class Balancing','Linear SVM',\
        'RF With One hot Encoding','RF With Response Coding',\
        'Stacking Classifier','Maximum Voting Classifier']

# Training loss
train_loss =
[nb_train,knn_train,lr_balance_train,lr_train,svm_train,rf_train,rf_response_train,stack_train,mvc_
train]

# Cross Validation loss
cv_loss = [nb_cv,knn_cv,lr_balance_cv,lr_cv,svm_cv,rf_cv,rf_response_cv,stack_cv,mvc_cv]

# Test loss
test_loss = [nb_test,knn_test,lr_balance_test,lr_test,svm_test,rf_test,rf_response_test,stack_test
,mvc_test]

# Percentage Misclassified points
misclassified =
[nb_misclassified,knn_misclassified,lr_balance_misclassified,lr_misclassified,svm_misclassified,\
                rf_misclassified,rf_response_misclassified,stack_misclassified,mvc_misclassified]

numbering = [1,2,3,4,5,6,7,8,9]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Train_loss",train_loss)
ptable.add_column("CV_loss",cv_loss)
ptable.add_column("Test_loss",test_loss)
ptable.add_column("Misclassified(%)",misclassified)

# Printing the Table
print(ptable)
```

```
+-------+---------------------------+---------------------+--------------------+-----------------
+--------------------+
| S.NO. |           MODEL           |      Train_loss     |      CV_loss       |     Test_loss
|  Misclassified(%)  |
+-------+---------------------------+---------------------+--------------------+-----------------
+--------------------+
|   1   |        Naive Bayes        |  0.5357886635941876 | 1.1627203702835112 |
1.213810087257443  | 36.278195488721806 |
|   2   |    K-Nearest Neighbour    |  0.5059032744599191 | 1.0130873350555543 |
1.0780972527681583 | 33.45864661654135  |
|   3   |  LR With Class Balancing  | 0.45127756128393487 | 0.9555910935177091 | 1.01413931108474
54 | 29.6992481203007   |
```

```
54 | 29.69924812030075  |
|   4   | LR Without Class Balancing | 0.44298744424302083  | 0.9848369466933264 |
1.0317763394409254 | 29.135338345864664 |
|   5   |         Linear SVM         |  0.6167958891284219  | 1.013402209329118  |
1.1056289144530362 | 31.203007518796994 |
|   6   |  RF With One hot Encoding  |  0.875351337805046   | 1.1734770059012316 |
1.2136206634420559 | 38.90977443609023  |
|   7   |  RF With Response Coding   | 0.052791703424134445 | 1.2015508204999736 |
1.2797335308960462 | 38.721804511278194 |
|   8   |    Stacking Classifier     |  0.5481568365472762  | 1.1171925281484958 |
1.1771866415011438 | 37.59398496240601  |
|   9   | Maximum Voting Classifier  |  0.8392747499944152  | 1.1682181117847334 | 1.21901652782742
5  | 37.59398496240601  |
+-------+----------------------------+----------------------+--------------------+-----------------
+--------------------+
```