Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25 (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 (https://www.youtube.com/watch?v=UwbuW7oK8rk)
3. https://www.youtube.com/watch?v=qxXRKVompI8 (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 (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 (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 [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

#from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
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[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",nam
es=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "sec
onds")
```

```
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 : 197.678021421 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

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

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Varia
tion']
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of outpu
t 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 dis
tribution 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

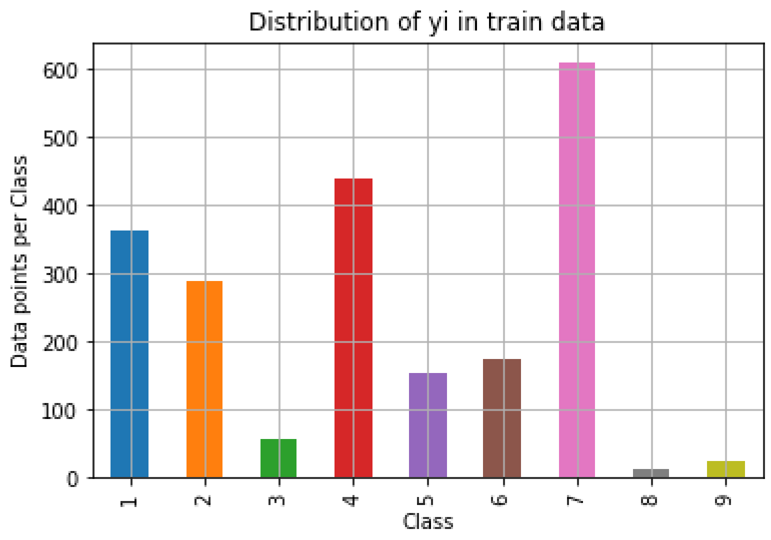
In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

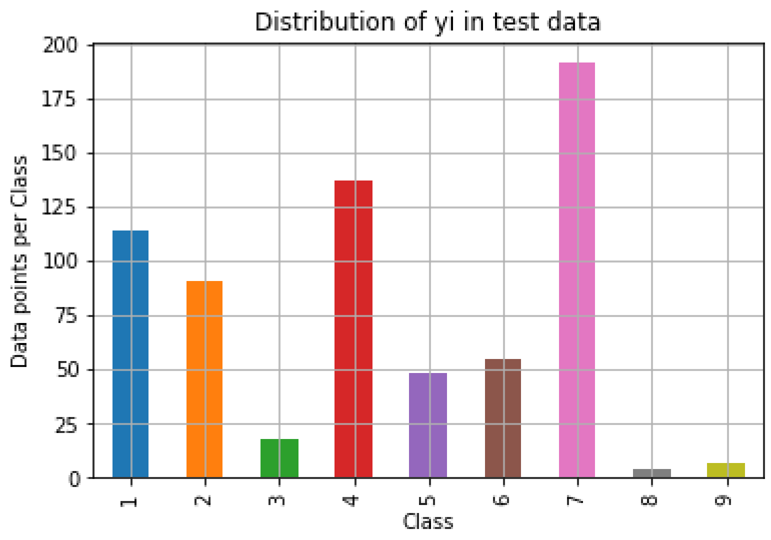
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:
```
# it returns a dict, keys as class labels and values as the number of data poi
nts 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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.
values[i], '(', np.round((train_class_distribution.values[i]/train_df.shape[0]
*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.v
alues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*10
0), 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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
```

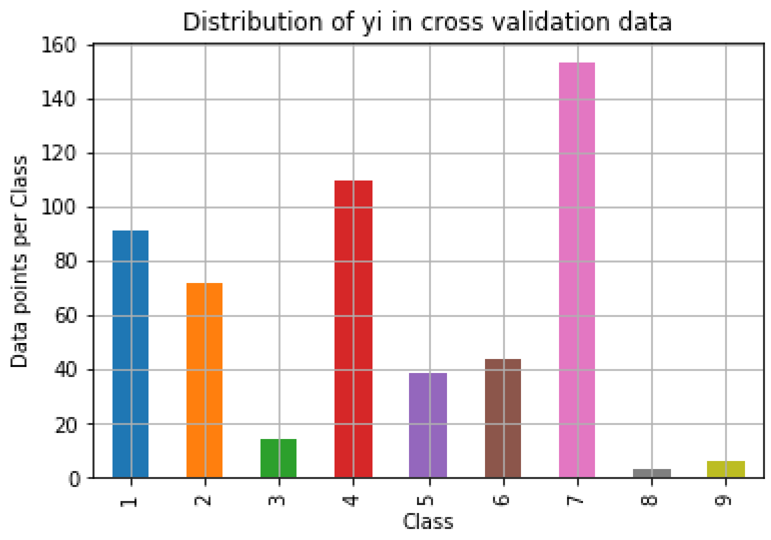
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```



```
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 %)
```

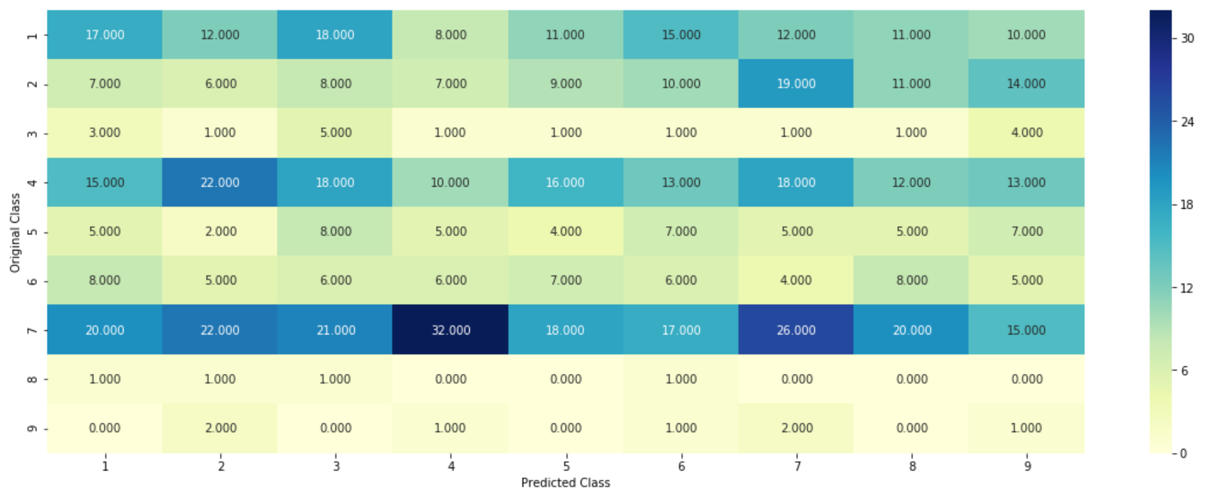
3.2 Prediction using a 'Random' Model

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

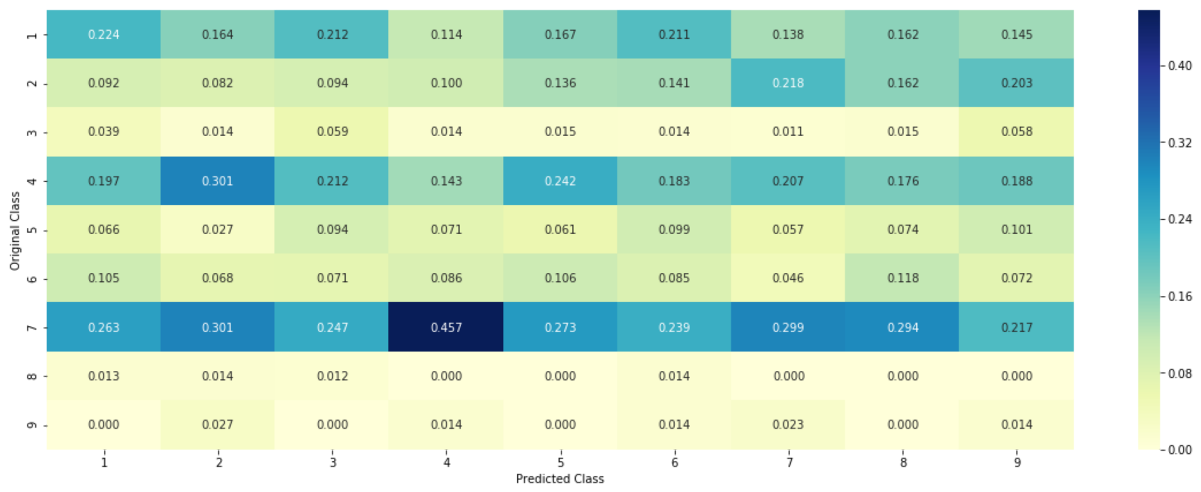
In [13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to
 rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    
    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, y
ticklabels=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, y
ticklabels=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, y
ticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [14]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
r 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_predicte
d_y, eps=1e-15))

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

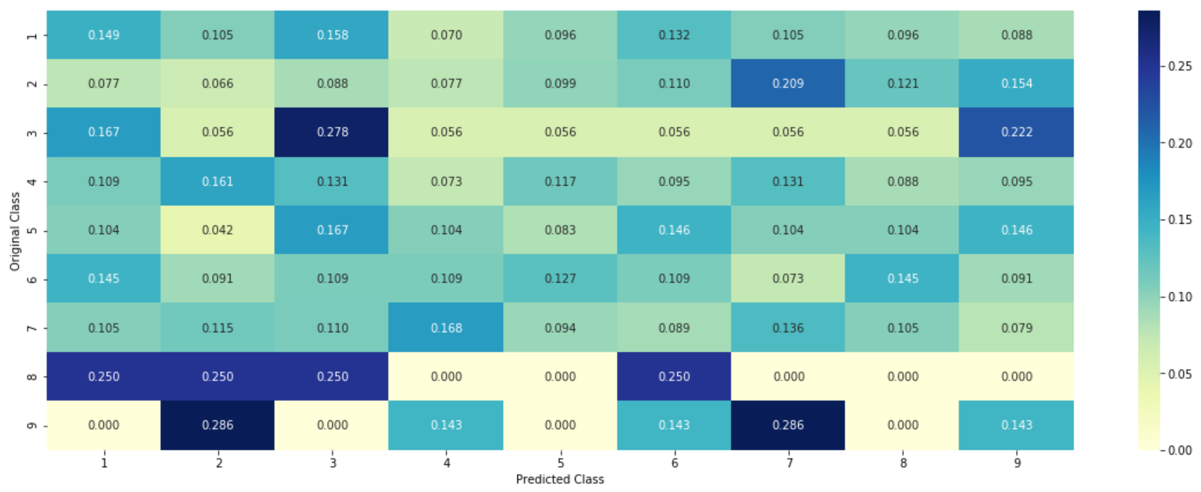
```
Log loss on Cross Validation Data using Random Model 2.434172618638161
Log loss on Test Data using Random Model 2.491440401877262
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in
 train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in cl
ass1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representat
ion 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 occu
```

```
red 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 occured 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, 0.0681818181818
18177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.
03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897959
18366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.0510
20408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818
1818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.05
6818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060
6060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069
182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920
5295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.2715
2317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333
33334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.299
99999999999999, 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 fea
ture value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is
 there in the train data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#              gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we 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?

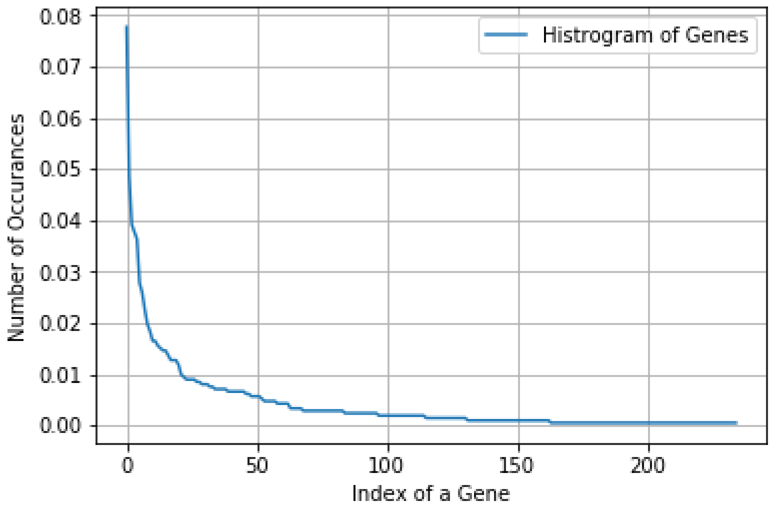
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 235
BRCA1     165
TP53      103
EGFR       83
BRCA2      80
PTEN       77
BRAF       59
KIT        55
ERBB2      48
ALK        42
PDGFRA     39
Name: Gene, dtype: int64
```

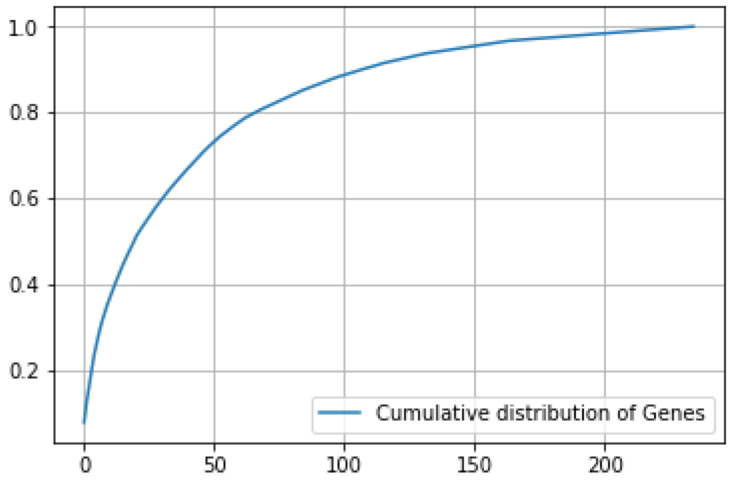
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

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

In [18]:
```
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 [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

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

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", tra
in_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test
_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df
))
```

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding m
ethod. The shape of gene feature: (2124, 9)
```

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
1054      TSC2
181       EGFR
1191    PIK3CA
525       TP53
2882     BRCA2
Name: Gene, dtype: object
```

In [24]:
```
gene_vectorizer.get_feature_names()
```

Out[24]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid2',
 'arid5b',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf4',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt1',
 'flt3',
 'foxa1',
 'foxl2',
 'foxo1',
 'foxp1',
 'fubp1',
 'gata3',
 'gna11',
 'gnas',
 'h3f3a',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'il7r',
 'inpp4b',
 'jak1',
 'jak2',
 'jun',
 'kdm5a',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'klf4',
 'kmt2a',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats1',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51c',
 'rad51d',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'srsf2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature:", train_gene_feature_onehotCoding.shap
e)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding m
ethod. The shape of gene feature: (2124, 234)
```

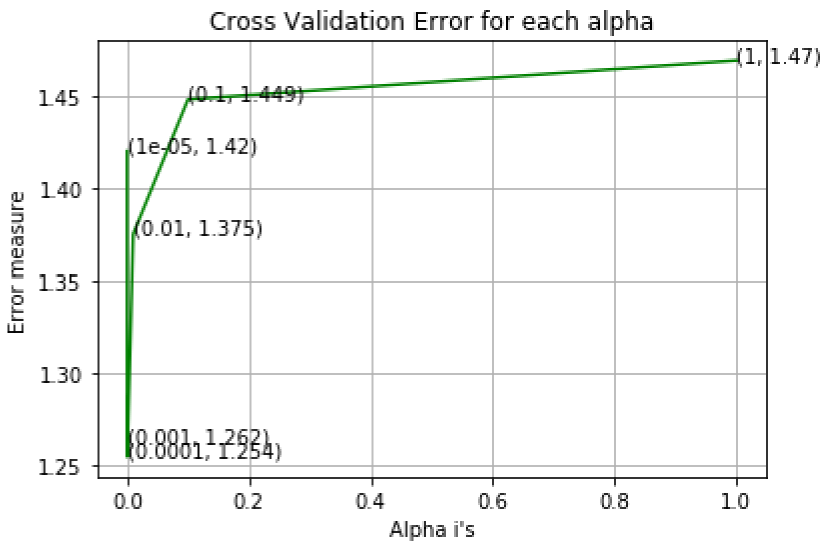
Q4. How good is this gene feature in predicting y_i?

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 [26]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))

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


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.4203870552050428
For values of alpha =  0.0001 The log loss is: 1.2543061430674634
For values of alpha =  0.001 The log loss is: 1.2623609305823833
For values of alpha =  0.01 The log loss is: 1.3751369835270013
For values of alpha =  0.1 The log loss is: 1.448539264998038
For values of alpha =  1 The log loss is: 1.4697740366537904
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0235949015635497
For values of best alpha =  0.0001 The cross validation log loss is: 1.254306
1430674634
For values of best alpha =  0.0001 The test log loss is: 1.2028865339198334
```

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

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

In [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",
unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  235  gen
es in train dataset?
Ans
1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 518 out of  532 : 97.36842105263158
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

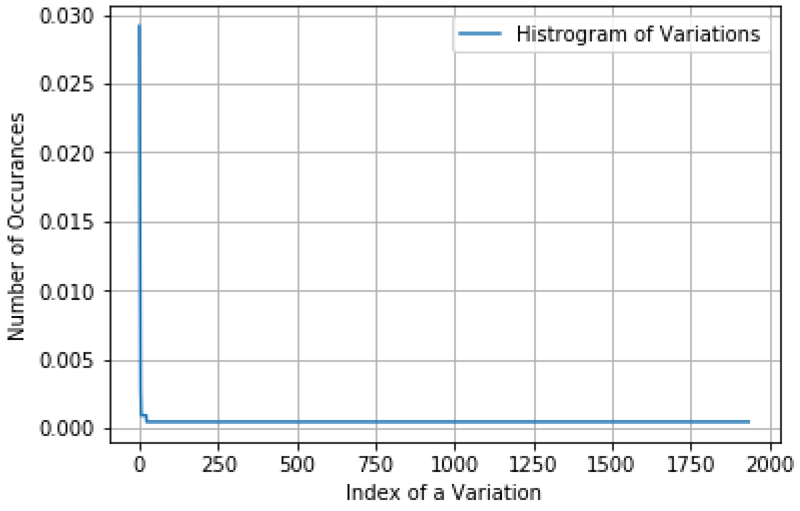
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1935
Truncating_Mutations    62
Deletion                47
Amplification           42
Fusions                 18
Overexpression           6
Q61R                     3
I31M                     2
G35R                     2
G12A                     2
G12D                     2
Name: Variation, dtype: int64
```

In [29]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

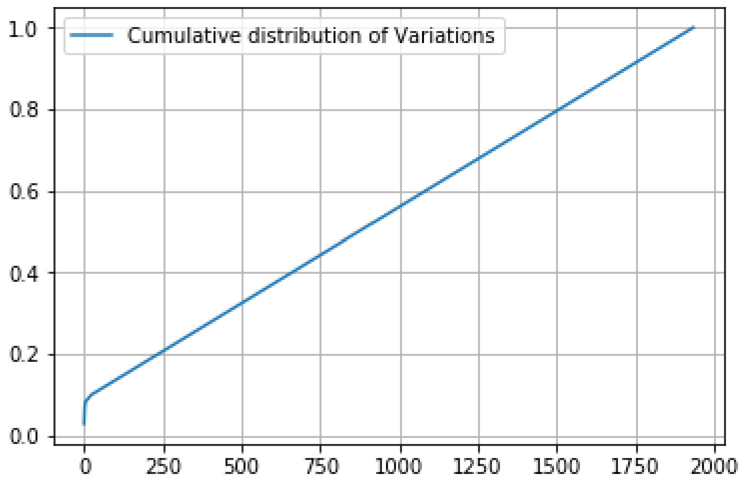
```
Ans: There are 1935 different categories of variations in the train data, and
they are distibuted as follows
```

In [30]:
```
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 [31]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02919021 0.05131827 0.07109228 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

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

In [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Varia
tion", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variat
ion", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
n", cv_df))
```

In [33]:
```
print("train_variation_feature_responseCoding is a converted feature using the
response coding method. The shape of Variation feature:", train_variation_feat
ure_responseCoding.shape)
```

```
train_variation_feature_responseCoding is a converted feature using the respo
nse coding method. The shape of Variation feature: (2124, 9)
```

In [34]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(trai
n_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df[
'Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Vari
ation'])
```

In [35]:
```
print("train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature:", train_variation_feat
ure_onehotCoding.shape)
```

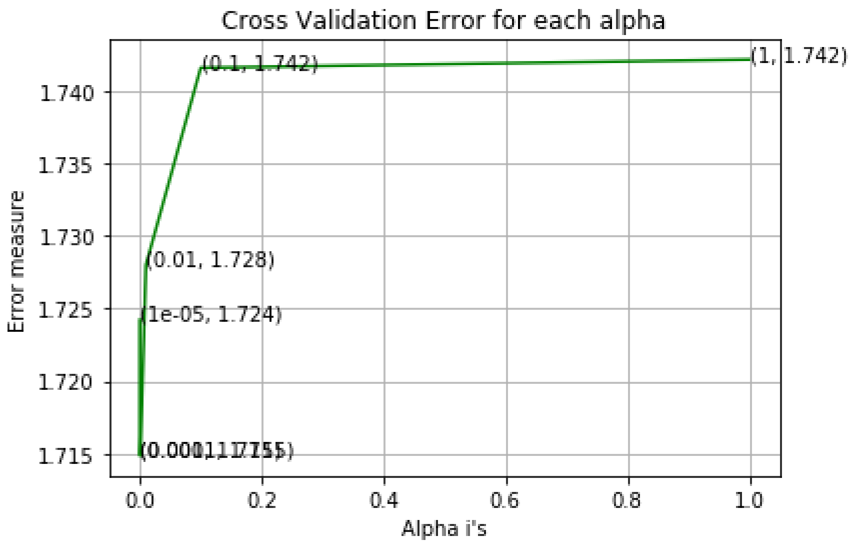
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1956)
```

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

Let's build a model just like the earlier!

In [36]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7242243821209344
For values of alpha =  0.0001 The log loss is: 1.7148330309304212
For values of alpha =  0.001 The log loss is: 1.7148849532389099
For values of alpha =  0.01 The log loss is: 1.7279763138225086
For values of alpha =  0.1 The log loss is: 1.7416011098877988
For values of alpha =  1 The log loss is: 1.7421724885858891
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7570242560225544
For values of best alpha =  0.0001 The cross validation log loss is: 1.714833
0309304212
For values of best alpha =  0.0001 The test log loss is: 1.7166468658850094
```

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

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

In [37]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1935  genes in test and cross
validation data sets?
Ans
1. In test data 75 out of 665 : 11.278195488721805
2. In cross validation data 56 out of  532 : 10.526315789473683
```

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 [38]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_di
ct.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
```

bigrams

In [43]:
```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2),max_features = 20
00)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# 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 return
s (1*number of features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of
 times it 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 : 2000
```

In [44]:
```
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 [45]:
```
#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 [46]:
```
# 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_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [47]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=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, axi
s=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 [48]:
```
#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 [49]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3279: 5, 1919: 5, 1661: 5, 1625: 5, 2458: 4, 2427: 4, 2082: 4, 2004:
4, 1992: 4, 1888: 4, 1667: 4, 1568: 4, 1526: 4, 1495: 4, 1446: 4, 5897: 3, 41
26: 3, 3558: 3, 3510: 3, 3207: 3, 2653: 3, 2433: 3, 2413: 3, 2389: 3, 2388:
3, 2285: 3, 2275: 3, 2246: 3, 2062: 3, 1975: 3, 1950: 3, 1926: 3, 1853: 3, 18
11: 3, 1806: 3, 1793: 3, 1784: 3, 1774: 3, 1772: 3, 1725: 3, 1685: 3, 1618:
3, 1613: 3, 1602: 3, 1600: 3, 1593: 3, 1592: 3, 1584: 3, 1565: 3, 1564: 3, 15
63: 3, 1552: 3, 1540: 3, 1531: 3, 1527: 3, 1512: 3, 1476: 3, 1464: 3, 1459:
3, 1445: 3, 1436: 3, 1429: 3, 1428: 3, 1424: 3, 1405: 3, 8899: 2, 8253: 2, 75
89: 2, 7507: 2, 6919: 2, 6524: 2, 6075: 2, 5487: 2, 5308: 2, 5222: 2, 4919:
2, 4803: 2, 4470: 2, 4409: 2, 4191: 2, 4180: 2, 4132: 2, 3939: 2, 3895: 2, 38
70: 2, 3820: 2, 3787: 2, 3711: 2, 3674: 2, 3660: 2, 3618: 2, 3605: 2, 3588:
2, 3522: 2, 3520: 2, 3482: 2, 3421: 2, 3411: 2, 3387: 2, 3364: 2, 3334: 2, 32
80: 2, 3214: 2, 3198: 2, 3126: 2, 3109: 2, 3097: 2, 3088: 2, 3045: 2, 3019:
2, 3011: 2, 3006: 2, 2966: 2, 2938: 2, 2855: 2, 2844: 2, 2797: 2, 2754: 2, 27
27: 2, 2719: 2, 2710: 2, 2703: 2, 2676: 2, 2668: 2, 2649: 2, 2616: 2, 2606:
2, 2590: 2, 2584: 2, 2554: 2, 2550: 2, 2547: 2, 2531: 2, 2507: 2, 2506: 2, 24
79: 2, 2476: 2, 2473: 2, 2462: 2, 2442: 2, 2429: 2, 2420: 2, 2399: 2, 2396:
2, 2322: 2, 2305: 2, 2279: 2, 2253: 2, 2252: 2, 2188: 2, 2185: 2, 2180: 2, 21
74: 2, 2169: 2, 2162: 2, 2154: 2, 2139: 2, 2134: 2, 2131: 2, 2125: 2, 2116:
2, 2112: 2, 2107: 2, 2106: 2, 2079: 2, 2074: 2, 2067: 2, 2064: 2, 2045: 2, 20
30: 2, 2020: 2, 2015: 2, 2010: 2, 2005: 2, 1994: 2, 1978: 2, 1969: 2, 1968:
2, 1963: 2, 1951: 2, 1941: 2, 1914: 2, 1911: 2, 1910: 2, 1908: 2, 1902: 2, 19
01: 2, 1900: 2, 1892: 2, 1889: 2, 1886: 2, 1869: 2, 1864: 2, 1859: 2, 1852:
2, 1850: 2, 1843: 2, 1830: 2, 1820: 2, 1815: 2, 1813: 2, 1810: 2, 1802: 2, 17
86: 2, 1781: 2, 1769: 2, 1764: 2, 1763: 2, 1752: 2, 1748: 2, 1744: 2, 1743:
2, 1741: 2, 1722: 2, 1712: 2, 1709: 2, 1702: 2, 1698: 2, 1695: 2, 1666: 2, 16
65: 2, 1657: 2, 1653: 2, 1647: 2, 1646: 2, 1645: 2, 1642: 2, 1639: 2, 1627:
2, 1620: 2, 1610: 2, 1607: 2, 1601: 2, 1599: 2, 1596: 2, 1591: 2, 1590: 2, 15
89: 2, 1586: 2, 1583: 2, 1578: 2, 1575: 2, 1574: 2, 1569: 2, 1567: 2, 1566:
2, 1561: 2, 1558: 2, 1554: 2, 1548: 2, 1547: 2, 1545: 2, 1543: 2, 1537: 2, 15
36: 2, 1535: 2, 1534: 2, 1524: 2, 1520: 2, 1502: 2, 1501: 2, 1500: 2, 1498:
2, 1497: 2, 1490: 2, 1486: 2, 1483: 2, 1482: 2, 1481: 2, 1480: 2, 1474: 2, 14
73: 2, 1471: 2, 1468: 2, 1462: 2, 1453: 2, 1450: 2, 1427: 2, 1425: 2, 1420:
2, 1416: 2, 1415: 2, 1413: 2, 1411: 2, 1406: 2, 1404: 2, 1402: 2, 152044: 1,
117917: 1, 79637: 1, 67013: 1, 66883: 1, 66202: 1, 66016: 1, 65478: 1, 64148:
1, 62859: 1, 54634: 1, 52920: 1, 50287: 1, 48565: 1, 46110: 1, 45862: 1, 4414
3: 1, 43215: 1, 42505: 1, 41154: 1, 40642: 1, 40423: 1, 40212: 1, 40095: 1, 3
9864: 1, 38824: 1, 37496: 1, 36701: 1, 35735: 1, 35540: 1, 35380: 1, 35287:
1, 33805: 1, 33542: 1, 33037: 1, 32899: 1, 31718: 1, 31187: 1, 29249: 1, 2817
7: 1, 26921: 1, 26134: 1, 25952: 1, 25768: 1, 25766: 1, 25397: 1, 24789: 1, 2
4522: 1, 24418: 1, 24314: 1, 24227: 1, 23989: 1, 23864: 1, 22988: 1, 22173:
1, 22077: 1, 21947: 1, 21731: 1, 21645: 1, 21470: 1, 20663: 1, 20575: 1, 2039
9: 1, 19722: 1, 19720: 1, 19654: 1, 19557: 1, 19486: 1, 19232: 1, 19222: 1, 1
9076: 1, 18985: 1, 18906: 1, 18805: 1, 18456: 1, 18310: 1, 18273: 1, 18252:
1, 18138: 1, 18111: 1, 18011: 1, 17996: 1, 17935: 1, 17875: 1, 17859: 1, 1743
1: 1, 17428: 1, 17193: 1, 17186: 1, 17107: 1, 17087: 1, 16943: 1, 16911: 1, 1
6865: 1, 16841: 1, 16722: 1, 16437: 1, 16324: 1, 16177: 1, 16025: 1, 16013:
1, 15653: 1, 15498: 1, 15470: 1, 15453: 1, 15325: 1, 15307: 1, 15277: 1, 1523
9: 1, 15101: 1, 15090: 1, 15035: 1, 14878: 1, 14779: 1, 14662: 1, 14614: 1, 1
4601: 1, 14569: 1, 14502: 1, 14468: 1, 14303: 1, 14115: 1, 13962: 1, 13894:
1, 13823: 1, 13777: 1, 13709: 1, 13595: 1, 13578: 1, 13528: 1, 13493: 1, 1348
8: 1, 13384: 1, 13214: 1, 13174: 1, 13166: 1, 13081: 1, 13066: 1, 13026: 1, 1
2856: 1, 12779: 1, 12702: 1, 12645: 1, 12521: 1, 12506: 1, 12499: 1, 12432:
1, 12396: 1, 12376: 1, 12320: 1, 12284: 1, 12265: 1, 12180: 1, 12126: 1, 1207
4: 1, 12038: 1, 12014: 1, 12013: 1, 11971: 1, 11952: 1, 11936: 1, 11934: 1, 1
1904: 1, 11875: 1, 11867: 1, 11793: 1, 11763: 1, 11744: 1, 11600: 1, 11592:
1, 11580: 1, 11532: 1, 11514: 1, 11492: 1, 11456: 1, 11385: 1, 11374: 1, 1131
```

```
6: 1, 11303: 1, 11278: 1, 11212: 1, 10942: 1, 10873: 1, 10826: 1, 10807: 1, 1
0769: 1, 10708: 1, 10650: 1, 10583: 1, 10554: 1, 10432: 1, 10394: 1, 10320:
1, 10306: 1, 10283: 1, 10148: 1, 10134: 1, 10132: 1, 10128: 1, 10117: 1, 1006
3: 1, 10020: 1, 9987: 1, 9985: 1, 9948: 1, 9937: 1, 9933: 1, 9884: 1, 9854:
1, 9822: 1, 9821: 1, 9795: 1, 9738: 1, 9626: 1, 9600: 1, 9597: 1, 9589: 1, 95
17: 1, 9456: 1, 9413: 1, 9407: 1, 9399: 1, 9385: 1, 9384: 1, 9332: 1, 9315:
1, 9288: 1, 9258: 1, 9221: 1, 9209: 1, 9166: 1, 9148: 1, 9023: 1, 8973: 1, 89
44: 1, 8924: 1, 8862: 1, 8854: 1, 8833: 1, 8822: 1, 8795: 1, 8792: 1, 8780:
1, 8769: 1, 8739: 1, 8696: 1, 8682: 1, 8680: 1, 8677: 1, 8644: 1, 8527: 1, 83
95: 1, 8375: 1, 8329: 1, 8297: 1, 8272: 1, 8252: 1, 8230: 1, 8178: 1, 8158:
1, 8147: 1, 8141: 1, 8126: 1, 8099: 1, 8083: 1, 8043: 1, 8020: 1, 8008: 1, 79
98: 1, 7986: 1, 7985: 1, 7930: 1, 7915: 1, 7903: 1, 7892: 1, 7883: 1, 7882:
1, 7862: 1, 7854: 1, 7773: 1, 7735: 1, 7715: 1, 7705: 1, 7696: 1, 7692: 1, 76
91: 1, 7654: 1, 7650: 1, 7649: 1, 7648: 1, 7637: 1, 7617: 1, 7594: 1, 7566:
1, 7553: 1, 7537: 1, 7531: 1, 7519: 1, 7497: 1, 7444: 1, 7436: 1, 7432: 1, 74
15: 1, 7394: 1, 7390: 1, 7372: 1, 7371: 1, 7293: 1, 7292: 1, 7226: 1, 7217:
1, 7211: 1, 7204: 1, 7196: 1, 7183: 1, 7165: 1, 7149: 1, 7148: 1, 7144: 1, 71
21: 1, 7114: 1, 7106: 1, 7094: 1, 7091: 1, 7079: 1, 7072: 1, 7047: 1, 7033:
1, 7011: 1, 6997: 1, 6961: 1, 6935: 1, 6933: 1, 6932: 1, 6930: 1, 6917: 1, 68
89: 1, 6870: 1, 6852: 1, 6845: 1, 6831: 1, 6810: 1, 6808: 1, 6780: 1, 6765:
1, 6763: 1, 6666: 1, 6647: 1, 6639: 1, 6617: 1, 6615: 1, 6609: 1, 6592: 1, 65
76: 1, 6572: 1, 6571: 1, 6560: 1, 6556: 1, 6514: 1, 6511: 1, 6509: 1, 6466:
1, 6449: 1, 6443: 1, 6432: 1, 6424: 1, 6410: 1, 6390: 1, 6361: 1, 6357: 1, 63
27: 1, 6324: 1, 6311: 1, 6299: 1, 6295: 1, 6292: 1, 6280: 1, 6268: 1, 6247:
1, 6239: 1, 6207: 1, 6201: 1, 6181: 1, 6179: 1, 6170: 1, 6135: 1, 6120: 1, 61
17: 1, 6108: 1, 6102: 1, 6094: 1, 6092: 1, 6064: 1, 6063: 1, 6050: 1, 6022:
1, 6021: 1, 6017: 1, 5996: 1, 5965: 1, 5957: 1, 5950: 1, 5932: 1, 5907: 1, 58
93: 1, 5880: 1, 5870: 1, 5869: 1, 5864: 1, 5837: 1, 5833: 1, 5825: 1, 5819:
1, 5816: 1, 5807: 1, 5773: 1, 5770: 1, 5764: 1, 5746: 1, 5745: 1, 5744: 1, 57
35: 1, 5721: 1, 5709: 1, 5688: 1, 5671: 1, 5669: 1, 5667: 1, 5651: 1, 5639:
1, 5631: 1, 5624: 1, 5602: 1, 5595: 1, 5562: 1, 5537: 1, 5517: 1, 5510: 1, 54
94: 1, 5491: 1, 5473: 1, 5465: 1, 5457: 1, 5434: 1, 5411: 1, 5401: 1, 5382:
1, 5372: 1, 5369: 1, 5367: 1, 5331: 1, 5330: 1, 5324: 1, 5311: 1, 5307: 1, 52
73: 1, 5262: 1, 5239: 1, 5230: 1, 5196: 1, 5191: 1, 5177: 1, 5170: 1, 5166:
1, 5156: 1, 5152: 1, 5135: 1, 5126: 1, 5114: 1, 5107: 1, 5103: 1, 5086: 1, 50
85: 1, 5080: 1, 5071: 1, 5068: 1, 5046: 1, 5034: 1, 5019: 1, 5018: 1, 5014:
1, 4998: 1, 4993: 1, 4991: 1, 4982: 1, 4972: 1, 4966: 1, 4948: 1, 4935: 1, 49
34: 1, 4926: 1, 4915: 1, 4911: 1, 4899: 1, 4898: 1, 4895: 1, 4886: 1, 4875:
1, 4868: 1, 4863: 1, 4854: 1, 4849: 1, 4848: 1, 4840: 1, 4836: 1, 4831: 1, 48
11: 1, 4789: 1, 4786: 1, 4784: 1, 4773: 1, 4765: 1, 4759: 1, 4757: 1, 4751:
1, 4749: 1, 4741: 1, 4739: 1, 4723: 1, 4711: 1, 4698: 1, 4683: 1, 4672: 1, 46
69: 1, 4665: 1, 4660: 1, 4634: 1, 4619: 1, 4604: 1, 4593: 1, 4587: 1, 4563:
1, 4560: 1, 4548: 1, 4545: 1, 4541: 1, 4536: 1, 4530: 1, 4517: 1, 4512: 1, 45
09: 1, 4506: 1, 4505: 1, 4501: 1, 4496: 1, 4494: 1, 4490: 1, 4486: 1, 4478:
1, 4467: 1, 4466: 1, 4434: 1, 4427: 1, 4421: 1, 4419: 1, 4400: 1, 4393: 1, 43
84: 1, 4381: 1, 4380: 1, 4367: 1, 4354: 1, 4350: 1, 4349: 1, 4347: 1, 4339:
1, 4336: 1, 4333: 1, 4321: 1, 4315: 1, 4309: 1, 4296: 1, 4295: 1, 4290: 1, 42
79: 1, 4277: 1, 4266: 1, 4263: 1, 4262: 1, 4243: 1, 4235: 1, 4228: 1, 4224:
1, 4220: 1, 4218: 1, 4215: 1, 4193: 1, 4192: 1, 4190: 1, 4181: 1, 4173: 1, 41
71: 1, 4169: 1, 4166: 1, 4158: 1, 4142: 1, 4138: 1, 4136: 1, 4133: 1, 4125:
1, 4124: 1, 4110: 1, 4108: 1, 4103: 1, 4101: 1, 4096: 1, 4094: 1, 4087: 1, 40
82: 1, 4078: 1, 4059: 1, 4053: 1, 4050: 1, 4047: 1, 4041: 1, 4019: 1, 4018:
1, 4013: 1, 4012: 1, 4008: 1, 4005: 1, 3996: 1, 3993: 1, 3990: 1, 3980: 1, 39
73: 1, 3971: 1, 3963: 1, 3962: 1, 3961: 1, 3950: 1, 3945: 1, 3940: 1, 3935:
1, 3934: 1, 3932: 1, 3913: 1, 3910: 1, 3894: 1, 3876: 1, 3865: 1, 3851: 1, 38
49: 1, 3848: 1, 3845: 1, 3840: 1, 3835: 1, 3821: 1, 3816: 1, 3812: 1, 3793:
1, 3791: 1, 3778: 1, 3775: 1, 3769: 1, 3766: 1, 3755: 1, 3746: 1, 3745: 1, 37
```

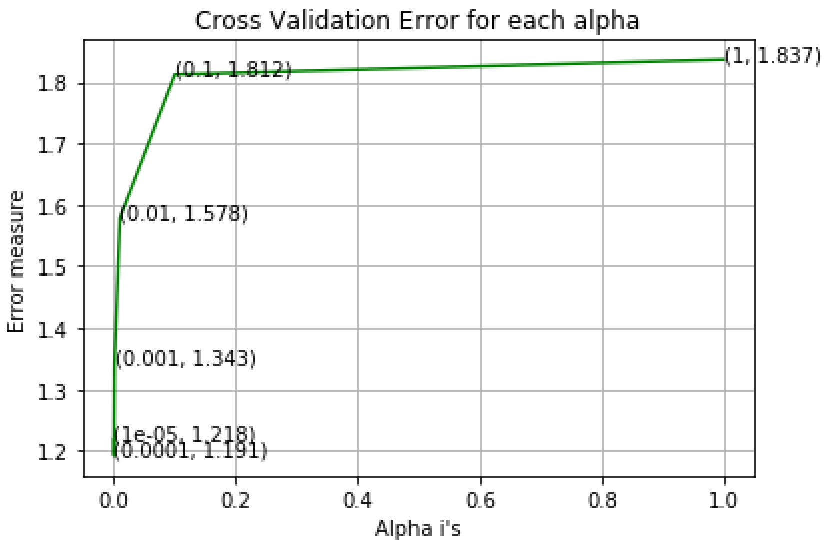
```
43: 1, 3740: 1, 3737: 1, 3736: 1, 3731: 1, 3726: 1, 3722: 1, 3721: 1, 3717:
1, 3705: 1, 3698: 1, 3692: 1, 3690: 1, 3686: 1, 3680: 1, 3669: 1, 3667: 1, 36
56: 1, 3654: 1, 3647: 1, 3636: 1, 3635: 1, 3627: 1, 3624: 1, 3622: 1, 3620:
1, 3619: 1, 3616: 1, 3604: 1, 3599: 1, 3596: 1, 3595: 1, 3579: 1, 3575: 1, 35
70: 1, 3561: 1, 3560: 1, 3553: 1, 3536: 1, 3530: 1, 3523: 1, 3515: 1, 3506:
1, 3505: 1, 3504: 1, 3503: 1, 3500: 1, 3486: 1, 3480: 1, 3477: 1, 3476: 1, 34
65: 1, 3463: 1, 3462: 1, 3459: 1, 3458: 1, 3457: 1, 3446: 1, 3442: 1, 3439:
1, 3433: 1, 3431: 1, 3428: 1, 3422: 1, 3419: 1, 3416: 1, 3414: 1, 3412: 1, 34
07: 1, 3401: 1, 3400: 1, 3398: 1, 3393: 1, 3389: 1, 3383: 1, 3380: 1, 3372:
1, 3356: 1, 3354: 1, 3348: 1, 3340: 1, 3339: 1, 3338: 1, 3332: 1, 3326: 1, 33
25: 1, 3321: 1, 3317: 1, 3310: 1, 3307: 1, 3305: 1, 3304: 1, 3297: 1, 3295:
1, 3282: 1, 3276: 1, 3272: 1, 3268: 1, 3267: 1, 3263: 1, 3262: 1, 3258: 1, 32
57: 1, 3254: 1, 3252: 1, 3247: 1, 3238: 1, 3234: 1, 3227: 1, 3224: 1, 3218:
1, 3215: 1, 3212: 1, 3204: 1, 3200: 1, 3199: 1, 3188: 1, 3187: 1, 3186: 1, 31
84: 1, 3183: 1, 3178: 1, 3176: 1, 3172: 1, 3167: 1, 3162: 1, 3161: 1, 3160:
1, 3158: 1, 3153: 1, 3150: 1, 3148: 1, 3137: 1, 3132: 1, 3131: 1, 3124: 1, 31
18: 1, 3116: 1, 3114: 1, 3107: 1, 3106: 1, 3105: 1, 3098: 1, 3095: 1, 3094:
1, 3086: 1, 3085: 1, 3080: 1, 3076: 1, 3075: 1, 3069: 1, 3062: 1, 3055: 1, 30
35: 1, 3034: 1, 3027: 1, 3024: 1, 3023: 1, 3017: 1, 3005: 1, 3001: 1, 2997:
1, 2996: 1, 2990: 1, 2989: 1, 2976: 1, 2975: 1, 2970: 1, 2969: 1, 2963: 1, 29
48: 1, 2943: 1, 2941: 1, 2935: 1, 2934: 1, 2933: 1, 2932: 1, 2927: 1, 2925:
1, 2915: 1, 2914: 1, 2910: 1, 2908: 1, 2907: 1, 2905: 1, 2899: 1, 2891: 1, 28
89: 1, 2886: 1, 2881: 1, 2877: 1, 2871: 1, 2867: 1, 2854: 1, 2852: 1, 2850:
1, 2848: 1, 2831: 1, 2830: 1, 2829: 1, 2828: 1, 2825: 1, 2820: 1, 2816: 1, 28
13: 1, 2810: 1, 2808: 1, 2807: 1, 2805: 1, 2802: 1, 2796: 1, 2783: 1, 2774:
1, 2770: 1, 2766: 1, 2763: 1, 2756: 1, 2752: 1, 2749: 1, 2737: 1, 2736: 1, 27
16: 1, 2701: 1, 2698: 1, 2690: 1, 2689: 1, 2688: 1, 2684: 1, 2678: 1, 2677:
1, 2675: 1, 2665: 1, 2664: 1, 2663: 1, 2661: 1, 2660: 1, 2651: 1, 2645: 1, 26
43: 1, 2640: 1, 2631: 1, 2629: 1, 2628: 1, 2627: 1, 2624: 1, 2623: 1, 2614:
1, 2613: 1, 2612: 1, 2605: 1, 2602: 1, 2597: 1, 2596: 1, 2589: 1, 2585: 1, 25
82: 1, 2581: 1, 2574: 1, 2571: 1, 2570: 1, 2569: 1, 2565: 1, 2564: 1, 2562:
1, 2560: 1, 2559: 1, 2558: 1, 2557: 1, 2555: 1, 2553: 1, 2551: 1, 2546: 1, 25
43: 1, 2540: 1, 2536: 1, 2535: 1, 2530: 1, 2529: 1, 2528: 1, 2525: 1, 2518:
1, 2516: 1, 2514: 1, 2508: 1, 2503: 1, 2500: 1, 2499: 1, 2498: 1, 2495: 1, 24
91: 1, 2488: 1, 2484: 1, 2478: 1, 2475: 1, 2472: 1, 2470: 1, 2466: 1, 2465:
1, 2463: 1, 2459: 1, 2457: 1, 2452: 1, 2451: 1, 2450: 1, 2449: 1, 2446: 1, 24
45: 1, 2443: 1, 2437: 1, 2435: 1, 2428: 1, 2425: 1, 2424: 1, 2423: 1, 2417:
1, 2412: 1, 2411: 1, 2410: 1, 2408: 1, 2404: 1, 2401: 1, 2398: 1, 2395: 1, 23
94: 1, 2390: 1, 2387: 1, 2386: 1, 2385: 1, 2384: 1, 2378: 1, 2375: 1, 2373:
1, 2371: 1, 2370: 1, 2366: 1, 2365: 1, 2356: 1, 2350: 1, 2349: 1, 2348: 1, 23
47: 1, 2346: 1, 2344: 1, 2341: 1, 2340: 1, 2339: 1, 2338: 1, 2332: 1, 2320:
1, 2319: 1, 2318: 1, 2315: 1, 2313: 1, 2307: 1, 2306: 1, 2304: 1, 2303: 1, 23
02: 1, 2301: 1, 2299: 1, 2297: 1, 2293: 1, 2287: 1, 2284: 1, 2282: 1, 2281:
1, 2278: 1, 2267: 1, 2263: 1, 2260: 1, 2259: 1, 2258: 1, 2256: 1, 2254: 1, 22
49: 1, 2247: 1, 2245: 1, 2243: 1, 2240: 1, 2239: 1, 2238: 1, 2235: 1, 2234:
1, 2231: 1, 2227: 1, 2219: 1, 2216: 1, 2215: 1, 2210: 1, 2209: 1, 2207: 1, 22
04: 1, 2203: 1, 2202: 1, 2199: 1, 2198: 1, 2197: 1, 2194: 1, 2193: 1, 2190:
1, 2184: 1, 2178: 1, 2177: 1, 2175: 1, 2173: 1, 2172: 1, 2167: 1, 2161: 1, 21
56: 1, 2153: 1, 2148: 1, 2145: 1, 2142: 1, 2141: 1, 2140: 1, 2135: 1, 2133:
1, 2132: 1, 2124: 1, 2123: 1, 2122: 1, 2121: 1, 2117: 1, 2115: 1, 2114: 1, 21
13: 1, 2111: 1, 2110: 1, 2109: 1, 2102: 1, 2098: 1, 2093: 1, 2091: 1, 2089:
1, 2087: 1, 2083: 1, 2081: 1, 2080: 1, 2078: 1, 2076: 1, 2075: 1, 2070: 1, 20
68: 1, 2063: 1, 2061: 1, 2058: 1, 2050: 1, 2049: 1, 2047: 1, 2043: 1, 2042:
1, 2040: 1, 2034: 1, 2033: 1, 2031: 1, 2025: 1, 2024: 1, 2023: 1, 2022: 1, 20
21: 1, 2018: 1, 2017: 1, 2013: 1, 2012: 1, 2011: 1, 2009: 1, 1998: 1, 1991:
1, 1989: 1, 1986: 1, 1984: 1, 1983: 1, 1981: 1, 1979: 1, 1971: 1, 1970: 1, 19
67: 1, 1965: 1, 1964: 1, 1957: 1, 1955: 1, 1954: 1, 1948: 1, 1947: 1, 1946:
```

```
1, 1939: 1, 1937: 1, 1936: 1, 1932: 1, 1931: 1, 1930: 1, 1929: 1, 1927: 1, 19
25: 1, 1924: 1, 1923: 1, 1920: 1, 1918: 1, 1915: 1, 1912: 1, 1909: 1, 1907:
1, 1904: 1, 1903: 1, 1899: 1, 1897: 1, 1895: 1, 1890: 1, 1887: 1, 1882: 1, 18
75: 1, 1874: 1, 1870: 1, 1868: 1, 1867: 1, 1866: 1, 1861: 1, 1858: 1, 1854:
1, 1851: 1, 1848: 1, 1847: 1, 1845: 1, 1844: 1, 1839: 1, 1838: 1, 1835: 1, 18
33: 1, 1831: 1, 1829: 1, 1825: 1, 1824: 1, 1823: 1, 1818: 1, 1816: 1, 1809:
1, 1805: 1, 1801: 1, 1798: 1, 1797: 1, 1795: 1, 1794: 1, 1790: 1, 1789: 1, 17
88: 1, 1783: 1, 1782: 1, 1779: 1, 1777: 1, 1775: 1, 1773: 1, 1768: 1, 1766:
1, 1762: 1, 1760: 1, 1758: 1, 1757: 1, 1756: 1, 1753: 1, 1751: 1, 1745: 1, 17
38: 1, 1736: 1, 1734: 1, 1731: 1, 1729: 1, 1723: 1, 1720: 1, 1718: 1, 1717:
1, 1715: 1, 1713: 1, 1711: 1, 1710: 1, 1708: 1, 1703: 1, 1701: 1, 1699: 1, 16
97: 1, 1696: 1, 1694: 1, 1690: 1, 1687: 1, 1683: 1, 1682: 1, 1680: 1, 1679:
1, 1673: 1, 1672: 1, 1671: 1, 1670: 1, 1668: 1, 1662: 1, 1660: 1, 1659: 1, 16
54: 1, 1650: 1, 1649: 1, 1648: 1, 1644: 1, 1641: 1, 1638: 1, 1634: 1, 1633:
1, 1632: 1, 1630: 1, 1629: 1, 1628: 1, 1626: 1, 1621: 1, 1617: 1, 1616: 1, 16
12: 1, 1611: 1, 1604: 1, 1603: 1, 1582: 1, 1581: 1, 1580: 1, 1577: 1, 1573:
1, 1570: 1, 1562: 1, 1560: 1, 1557: 1, 1556: 1, 1555: 1, 1551: 1, 1544: 1, 15
38: 1, 1532: 1, 1530: 1, 1529: 1, 1523: 1, 1522: 1, 1521: 1, 1519: 1, 1516:
1, 1515: 1, 1510: 1, 1509: 1, 1508: 1, 1507: 1, 1506: 1, 1504: 1, 1496: 1, 14
91: 1, 1488: 1, 1484: 1, 1479: 1, 1478: 1, 1470: 1, 1467: 1, 1461: 1, 1456:
1, 1454: 1, 1452: 1, 1451: 1, 1449: 1, 1448: 1, 1447: 1, 1443: 1, 1441: 1, 14
40: 1, 1439: 1, 1438: 1, 1437: 1, 1435: 1, 1434: 1, 1433: 1, 1432: 1, 1431:
1, 1426: 1, 1423: 1, 1422: 1, 1421: 1, 1419: 1, 1414: 1, 1410: 1, 1409: 1, 14
08: 1, 1407: 1, 1401: 1})
```

In [50]:
```
# 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/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate=’optimal’, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
```

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

```
For values of alpha =  1e-05 The log loss is: 1.2183679601563289
For values of alpha =  0.0001 The log loss is: 1.1914560192024692
For values of alpha =  0.001 The log loss is: 1.343334013452947
For values of alpha =  0.01 The log loss is: 1.5784569674494398
For values of alpha =  0.1 The log loss is: 1.8124125163915863
For values of alpha =  1 The log loss is: 1.8368569947577014
```



```
For values of best alpha =  0.0001 The train log loss is: 0.8394127104365436
For values of best alpha =  0.0001 The cross validation log loss is: 1.191456
0192024692
For values of best alpha =  0.0001 The test log loss is: 1.1541963912774114
```

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

Ans. Yes, it seems like!

In [51]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3,ngram_range=(1,2),max_features = 20
00)
    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 [52]:
```
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")
```

```
93.25 % of word of test data appeared in train data
93.1 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [53]:
```
#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 bel
ongs 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 [54]:
```
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 [55]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3,ngram_range=(1,2),max_features =
2000)

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

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

Stacking the three types of features

In [56]:
```
# 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_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,t
rain_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,tes
t_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_vari
ation_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_fea
ture_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_r
esponseCoding))
```

In [57]:
```
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_o
nehotCoding.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, 4190)
(number of data points * number of features) in test data =  (665, 4190)
(number of data points * number of features) in cross validation data = (532,
4190)
```

In [58]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_r
esponseCoding.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

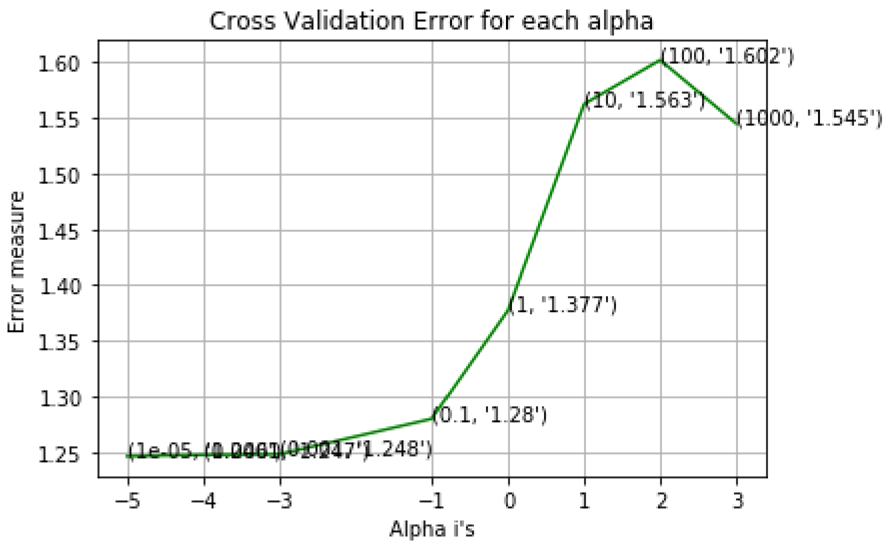
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [59]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

# 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probab
ility 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 i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.2460628191064078
for alpha = 0.0001
Log Loss : 1.2472863728864536
for alpha = 0.001
Log Loss : 1.2478091034203205
for alpha = 0.1
Log Loss : 1.2797109927797174
for alpha = 1
Log Loss : 1.3773993187997902
for alpha = 10
Log Loss : 1.562596897428551
for alpha = 100
Log Loss : 1.6021710990402622
for alpha = 1000
Log Loss : 1.5449492576897816
```

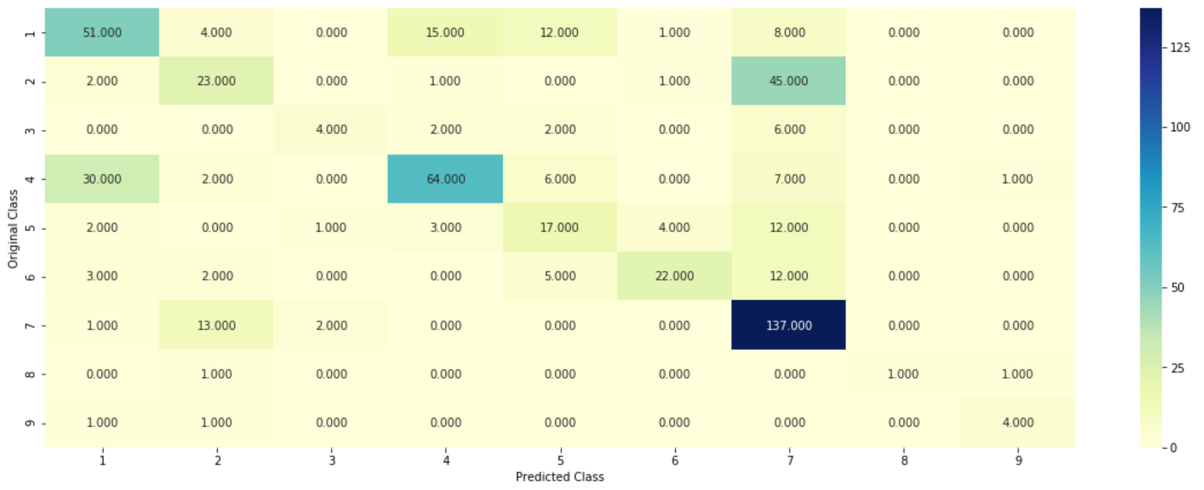


```
For values of best alpha =  1e-05 The train log loss is: 0.5225792122912197
For values of best alpha =  1e-05 The cross validation log loss is: 1.2460628
191064078
For values of best alpha =  1e-05 The test log loss is: 1.2343126638170523
```

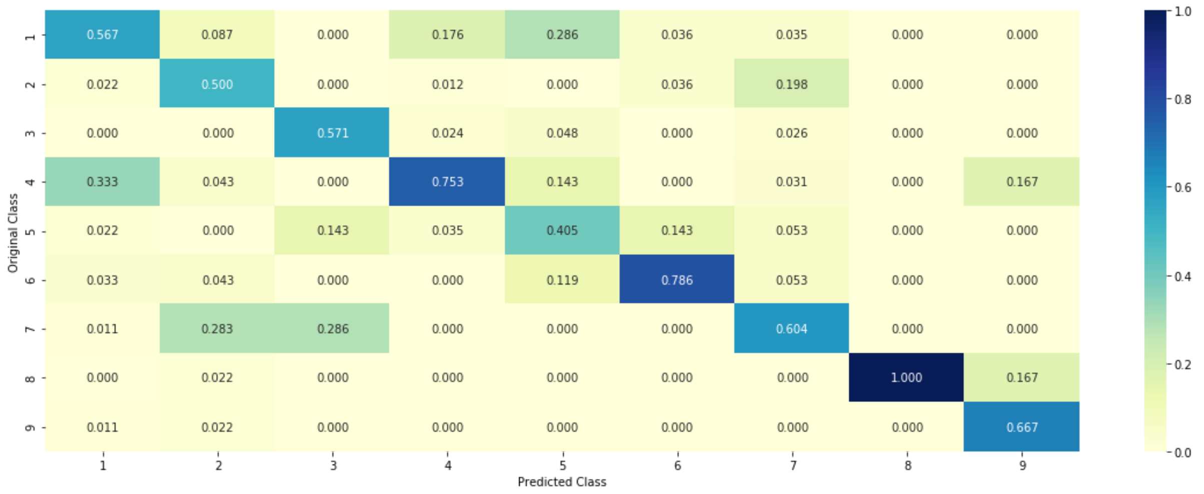
4.1.1.2. Testing the model with best hyper paramters

In [60]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

# 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probabilit
y estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv
_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

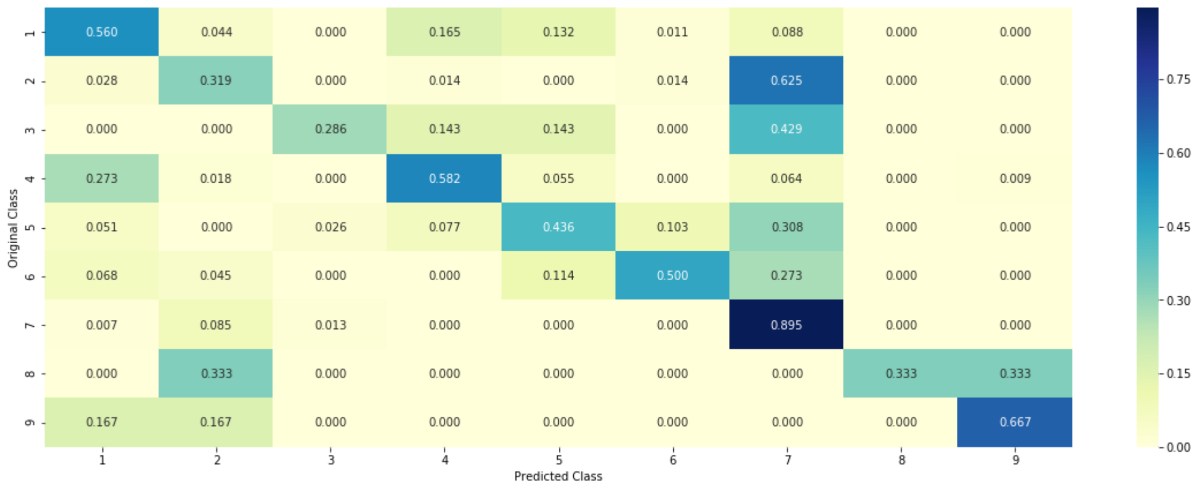
```
Log Loss : 1.2460628191064078
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [61]:
```
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3976 0.0513 0.012  0.3694 0.0398 0.0346 0.
0886 0.0046 0.0021]]
Actual Class : 4
--------------------------------------------------
13 Text feature [one] present in test data point [True]
14 Text feature [two] present in test data point [True]
15 Text feature [protein] present in test data point [True]
16 Text feature [type] present in test data point [True]
17 Text feature [dna] present in test data point [True]
18 Text feature [affect] present in test data point [True]
20 Text feature [sequence] present in test data point [True]
22 Text feature [function] present in test data point [True]
23 Text feature [wild] present in test data point [True]
24 Text feature [reduced] present in test data point [True]
25 Text feature [region] present in test data point [True]
28 Text feature [containing] present in test data point [True]
29 Text feature [present] present in test data point [True]
31 Text feature [specific] present in test data point [True]
33 Text feature [corresponding] present in test data point [True]
34 Text feature [indicate] present in test data point [True]
35 Text feature [sequences] present in test data point [True]
36 Text feature [five] present in test data point [True]
38 Text feature [involved] present in test data point [True]
39 Text feature [six] present in test data point [True]
40 Text feature [three] present in test data point [True]
42 Text feature [large] present in test data point [True]
43 Text feature [table] present in test data point [True]
45 Text feature [results] present in test data point [True]
46 Text feature [also] present in test data point [True]
48 Text feature [least] present in test data point [True]
52 Text feature [important] present in test data point [True]
55 Text feature [identified] present in test data point [True]
57 Text feature [proteins] present in test data point [True]
60 Text feature [indicated] present in test data point [True]
62 Text feature [expected] present in test data point [True]
66 Text feature [result] present in test data point [True]
68 Text feature [indicates] present in test data point [True]
69 Text feature [analysis] present in test data point [True]
72 Text feature [using] present in test data point [True]
73 Text feature [shown] present in test data point [True]
75 Text feature [used] present in test data point [True]
77 Text feature [central] present in test data point [True]
79 Text feature [data] present in test data point [True]
81 Text feature [gene] present in test data point [True]
84 Text feature [likely] present in test data point [True]
85 Text feature [may] present in test data point [True]
86 Text feature [genetic] present in test data point [True]
87 Text feature [possibility] present in test data point [True]
88 Text feature [coding] present in test data point [True]
90 Text feature [terminal] present in test data point [True]
91 Text feature [either] present in test data point [True]
93 Text feature [different] present in test data point [True]
97 Text feature [whether] present in test data point [True]
98 Text feature [addition] present in test data point [True]
Out of the top  100  features  50 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [62]:
```
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

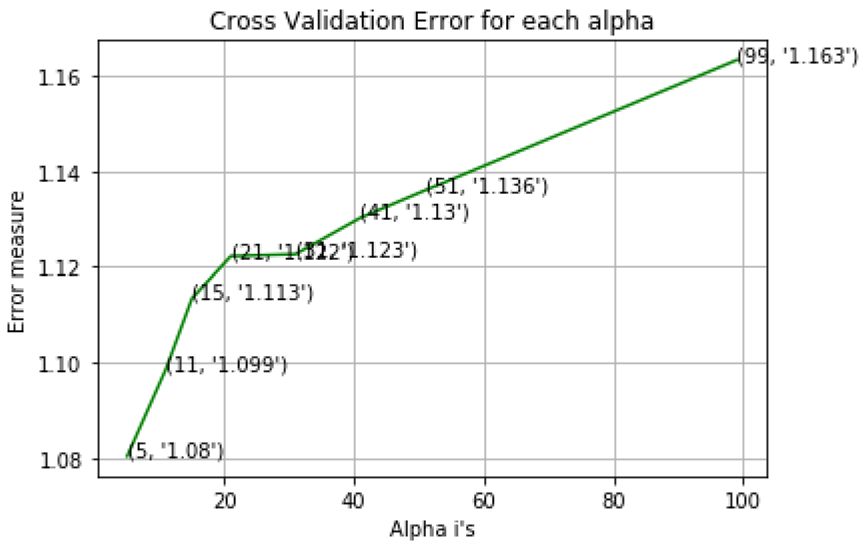
```
Predicted Class : 4
Predicted Class Probabilities: [[0.3282 0.0524 0.0123 0.4342 0.0405 0.0354 0.
0902 0.0047 0.0021]]
Actual Class : 1
--------------------------------------------------
10 Text feature [function] present in test data point [True]
11 Text feature [protein] present in test data point [True]
18 Text feature [proteins] present in test data point [True]
19 Text feature [functional] present in test data point [True]
20 Text feature [activity] present in test data point [True]
24 Text feature [experiments] present in test data point [True]
25 Text feature [mammalian] present in test data point [True]
28 Text feature [results] present in test data point [True]
29 Text feature [type] present in test data point [True]
30 Text feature [retained] present in test data point [True]
32 Text feature [suppressor] present in test data point [True]
34 Text feature [stability] present in test data point [True]
35 Text feature [partially] present in test data point [True]
36 Text feature [wild] present in test data point [True]
37 Text feature [critical] present in test data point [True]
39 Text feature [determined] present in test data point [True]
40 Text feature [indicate] present in test data point [True]
41 Text feature [whereas] present in test data point [True]
43 Text feature [loss] present in test data point [True]
44 Text feature [whether] present in test data point [True]
45 Text feature [determine] present in test data point [True]
48 Text feature [transfected] present in test data point [True]
55 Text feature [possible] present in test data point [True]
56 Text feature [ability] present in test data point [True]
57 Text feature [two] present in test data point [True]
58 Text feature [tested] present in test data point [True]
59 Text feature [made] present in test data point [True]
60 Text feature [important] present in test data point [True]
63 Text feature [therefore] present in test data point [True]
65 Text feature [co] present in test data point [True]
66 Text feature [three] present in test data point [True]
68 Text feature [shown] present in test data point [True]
69 Text feature [related] present in test data point [True]
70 Text feature [either] present in test data point [True]
77 Text feature [transfection] present in test data point [True]
78 Text feature [effect] present in test data point [True]
79 Text feature [although] present in test data point [True]
80 Text feature [associated] present in test data point [True]
81 Text feature [vector] present in test data point [True]
83 Text feature [generated] present in test data point [True]
84 Text feature [assay] present in test data point [True]
88 Text feature [described] present in test data point [True]
89 Text feature [general] present in test data point [True]
94 Text feature [predicted] present in test data point [True]
96 Text feature [cannot] present in test data point [True]
97 Text feature [system] present in test data point [True]
98 Text feature [effects] present in test data point [True]
99 Text feature [also] present in test data point [True]
Out of the top  100  features  48 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [63]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probab
ility 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 i
s:",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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0804687134697877
for alpha = 11
Log Loss : 1.0987479798622142
for alpha = 15
Log Loss : 1.1134594420328627
for alpha = 21
Log Loss : 1.1222223492768386
for alpha = 31
Log Loss : 1.1226326486132225
for alpha = 41
Log Loss : 1.130225386768865
for alpha = 51
Log Loss : 1.135986995229164
for alpha = 99
Log Loss : 1.1632099668874718
```

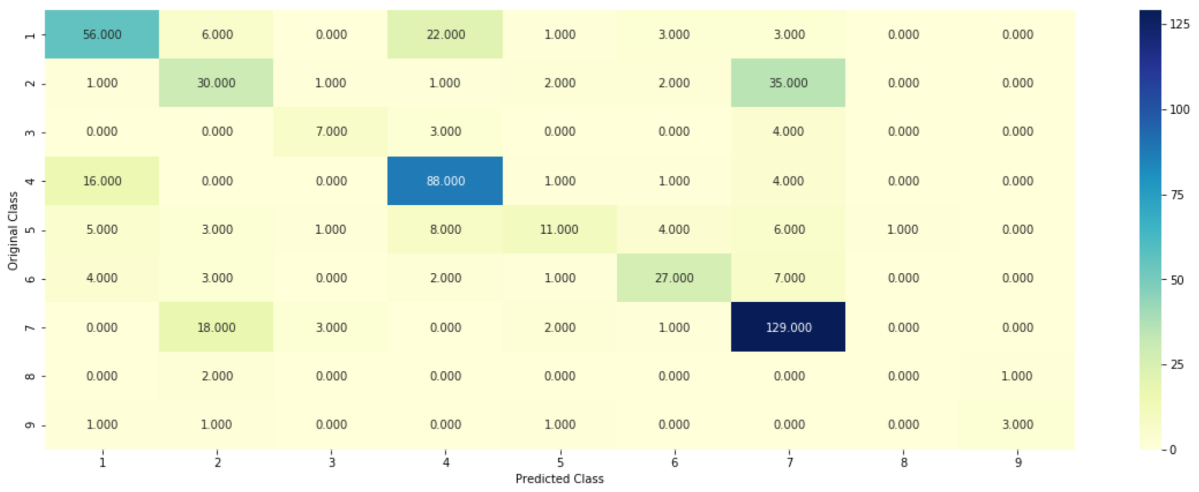


```
For values of best alpha =  5 The train log loss is: 0.4812118020499718
For values of best alpha =  5 The cross validation log loss is: 1.08046871346
97877
For values of best alpha =  5 The test log loss is: 1.0683039844452502
```

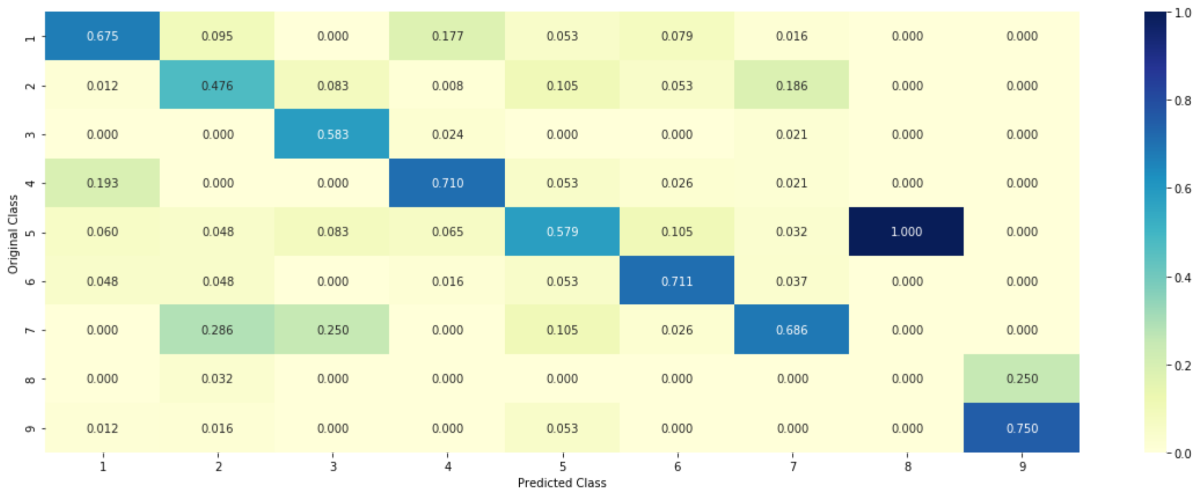
4.2.2. Testing the model with best hyper paramters

In [64]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

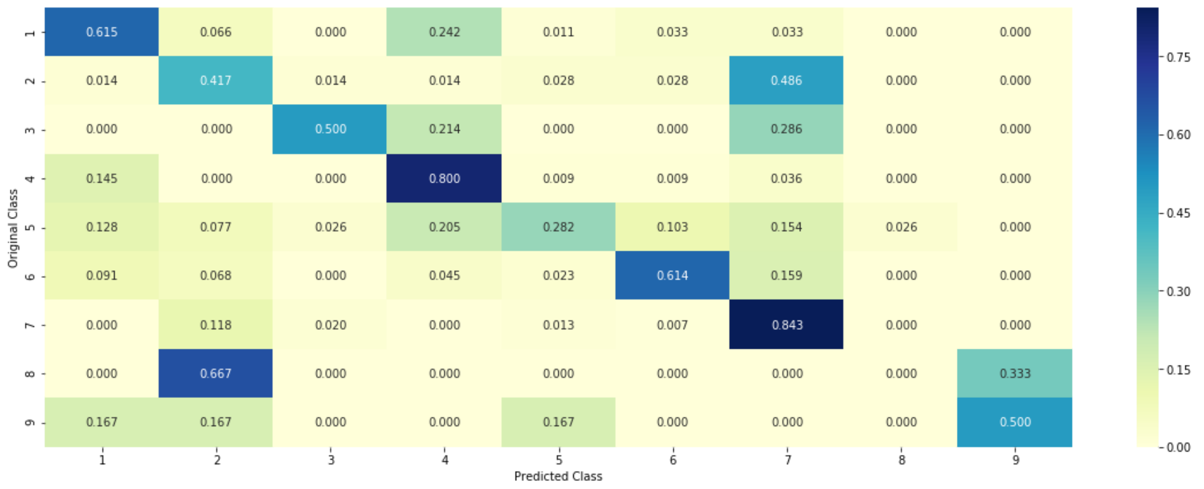
```
Log loss : 1.0804687134697877
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

In [65]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

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

4.2.4. Sample Query Point-2

In [66]:
```
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].reshap
e(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 o
f the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

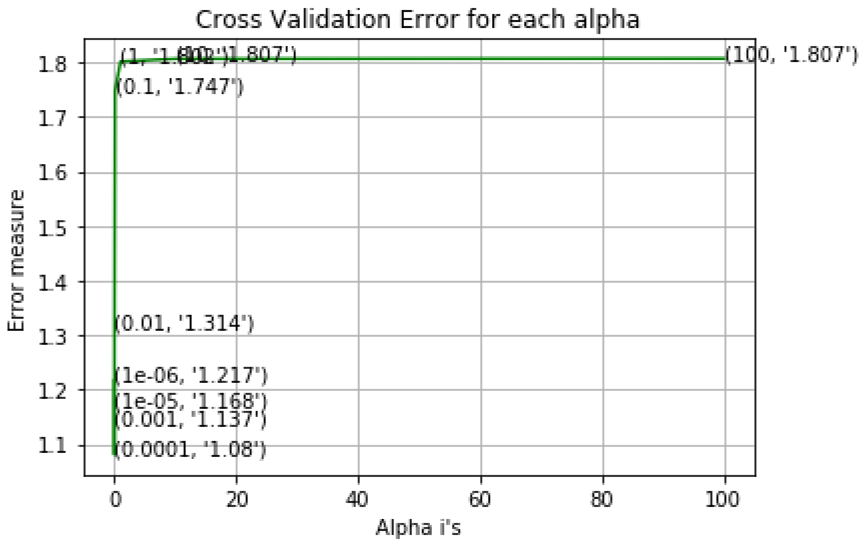
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [67]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2167692402946246
for alpha = 1e-05
Log Loss : 1.1680052631353013
for alpha = 0.0001
Log Loss : 1.080362663950907
for alpha = 0.001
Log Loss : 1.137377847342491
for alpha = 0.01
Log Loss : 1.3140259571357593
for alpha = 0.1
Log Loss : 1.7474571554555869
for alpha = 1
Log Loss : 1.8015389630901113
for alpha = 10
Log Loss : 1.806512467699887
for alpha = 100
Log Loss : 1.807039508313185
```

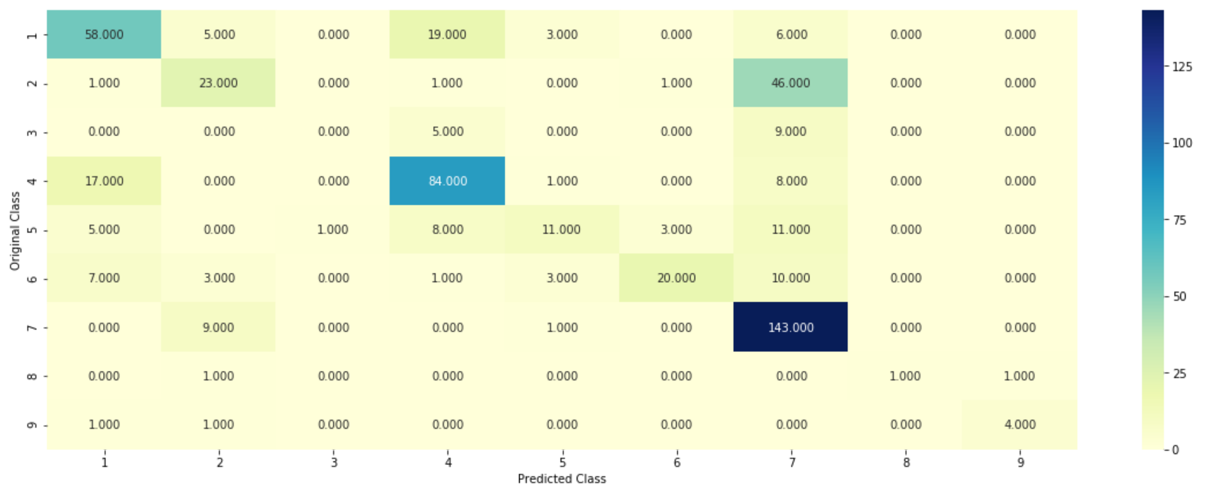


```
For values of best alpha =  0.0001 The train log loss is: 0.42279888995422626
For values of best alpha =  0.0001 The cross validation log loss is: 1.080362
663950907
For values of best alpha =  0.0001 The test log loss is: 1.071553022656809
```

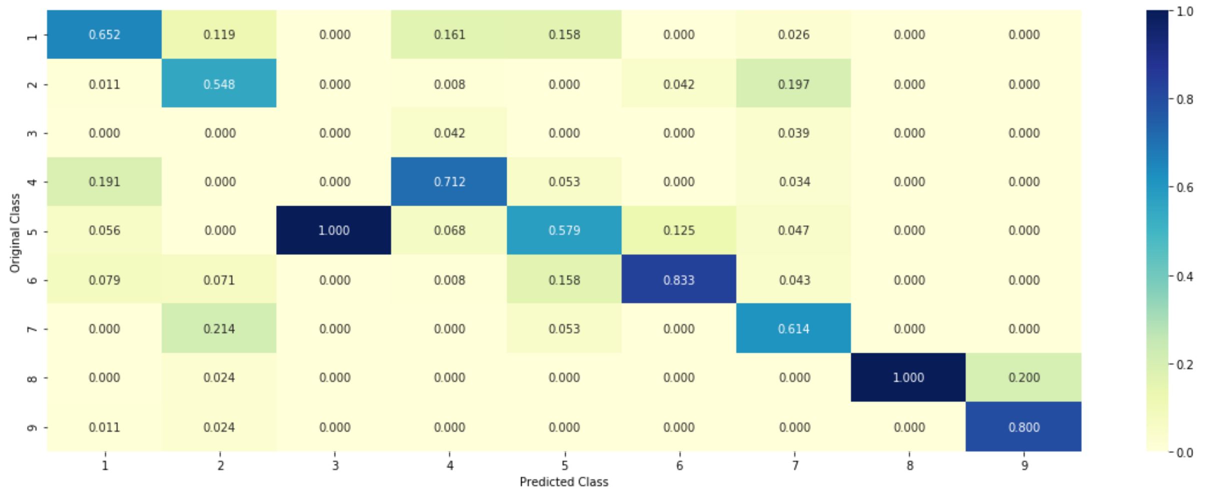
4.3.1.2. Testing the model with best hyper paramters

In [68]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

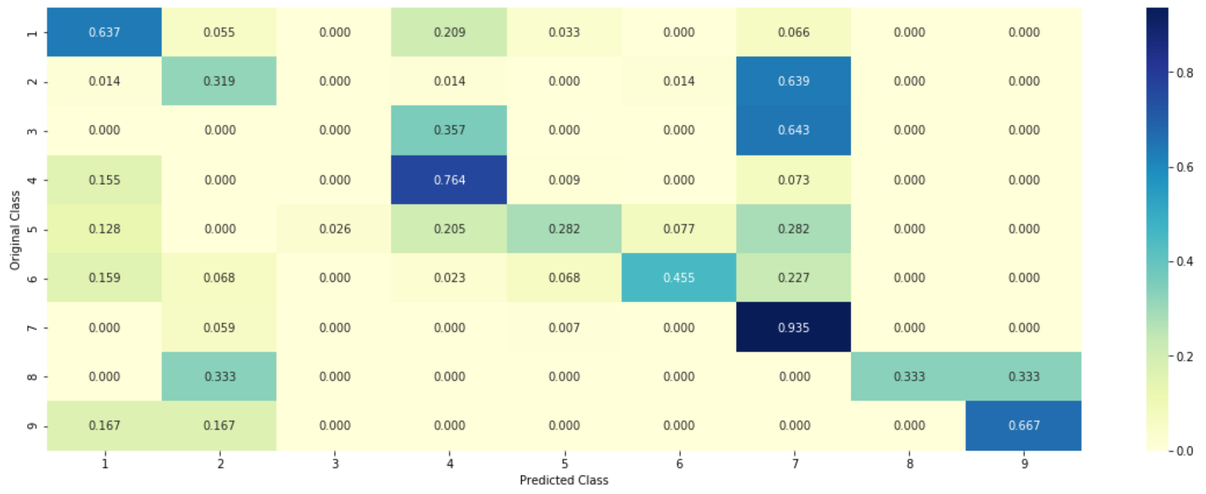
```
Log loss : 1.080362663950907
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [69]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], ye
s_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query poin
t")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," cl
ass:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or
Not']))
```

4.3.1.3.1. Correctly Classified point

In [70]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3813 0.0455 0.0202 0.4286 0.0418 0.0242 0.
0476 0.0062 0.0045]]
Actual Class : 4
--------------------------------------------------
61 Text feature [suppressor] present in test data point [True]
71 Text feature [iii] present in test data point [True]
183 Text feature [missense] present in test data point [True]
199 Text feature [nonsense] present in test data point [True]
227 Text feature [frameshift] present in test data point [True]
244 Text feature [putative] present in test data point [True]
254 Text feature [inactivation] present in test data point [True]
264 Text feature [weight] present in test data point [True]
322 Text feature [indicate] present in test data point [True]
327 Text feature [density] present in test data point [True]
335 Text feature [functional] present in test data point [True]
340 Text feature [tumorigenesis] present in test data point [True]
345 Text feature [mouse] present in test data point [True]
360 Text feature [resulting] present in test data point [True]
392 Text feature [protein] present in test data point [True]
398 Text feature [assay] present in test data point [True]
411 Text feature [genetic] present in test data point [True]
421 Text feature [process] present in test data point [True]
424 Text feature [domains] present in test data point [True]
432 Text feature [large] present in test data point [True]
444 Text feature [changes] present in test data point [True]
447 Text feature [show] present in test data point [True]
450 Text feature [represent] present in test data point [True]
458 Text feature [renal] present in test data point [True]
476 Text feature [mutants] present in test data point [True]
Out of the top  500  features  25 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [71]:
```
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

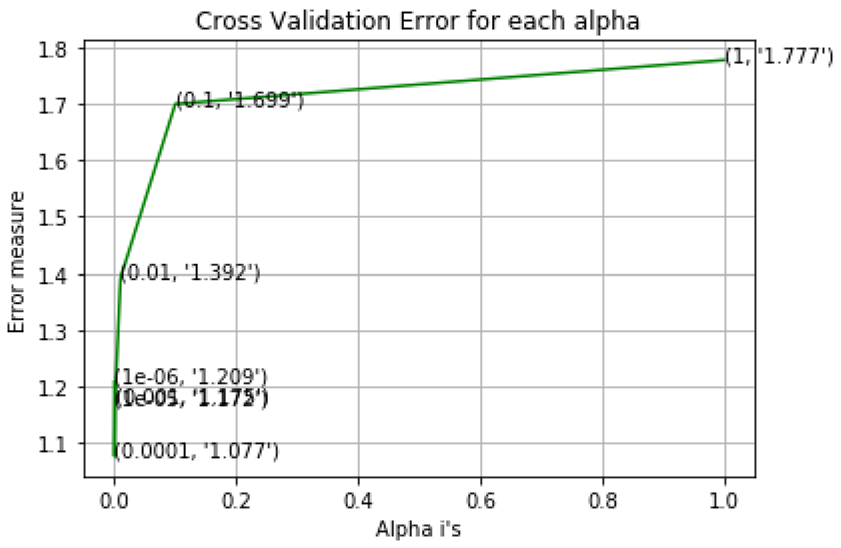
```
Predicted Class : 4
Predicted Class Probabilities: [[0.289  0.0248 0.0057 0.6274 0.0219 0.0198 0.
006  0.004  0.0015]]
Actual Class : 1
--------------------------------------------------
61 Text feature [suppressor] present in test data point [True]
135 Text feature [mammalian] present in test data point [True]
136 Text feature [stability] present in test data point [True]
145 Text feature [defect] present in test data point [True]
154 Text feature [brca1] present in test data point [True]
155 Text feature [mice] present in test data point [True]
172 Text feature [loss] present in test data point [True]
214 Text feature [analysed] present in test data point [True]
272 Text feature [cannot] present in test data point [True]
287 Text feature [incidence] present in test data point [True]
292 Text feature [nucleus] present in test data point [True]
293 Text feature [deleted] present in test data point [True]
303 Text feature [endogenous] present in test data point [True]
307 Text feature [plasmids] present in test data point [True]
315 Text feature [repair] present in test data point [True]
318 Text feature [plasmid] present in test data point [True]
319 Text feature [tumours] present in test data point [True]
322 Text feature [indicate] present in test data point [True]
335 Text feature [functional] present in test data point [True]
340 Text feature [tumorigenesis] present in test data point [True]
343 Text feature [ref] present in test data point [True]
354 Text feature [co] present in test data point [True]
357 Text feature [frequent] present in test data point [True]
359 Text feature [phenotype] present in test data point [True]
362 Text feature [promotes] present in test data point [True]
372 Text feature [iarc] present in test data point [True]
392 Text feature [protein] present in test data point [True]
398 Text feature [assay] present in test data point [True]
403 Text feature [displayed] present in test data point [True]
411 Text feature [genetic] present in test data point [True]
421 Text feature [process] present in test data point [True]
423 Text feature [fibroblasts] present in test data point [True]
427 Text feature [control] present in test data point [True]
432 Text feature [large] present in test data point [True]
435 Text feature [risk] present in test data point [True]
442 Text feature [null] present in test data point [True]
447 Text feature [show] present in test data point [True]
448 Text feature [ubiquitin] present in test data point [True]
450 Text feature [represent] present in test data point [True]
451 Text feature [correlation] present in test data point [True]
476 Text feature [mutants] present in test data point [True]
482 Text feature [defective] present in test data point [True]
484 Text feature [strongly] present in test data point [True]
493 Text feature [1996] present in test data point [True]
Out of the top  500  features  44 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [72]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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 i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.2088546782918668
for alpha = 1e-05
Log Loss : 1.171751231737619
for alpha = 0.0001
Log Loss : 1.0771604908005332
for alpha = 0.001
Log Loss : 1.1754005668461796
for alpha = 0.01
Log Loss : 1.391736264560467
for alpha = 0.1
Log Loss : 1.6993229602843969
for alpha = 1
Log Loss : 1.7769484652264647
```

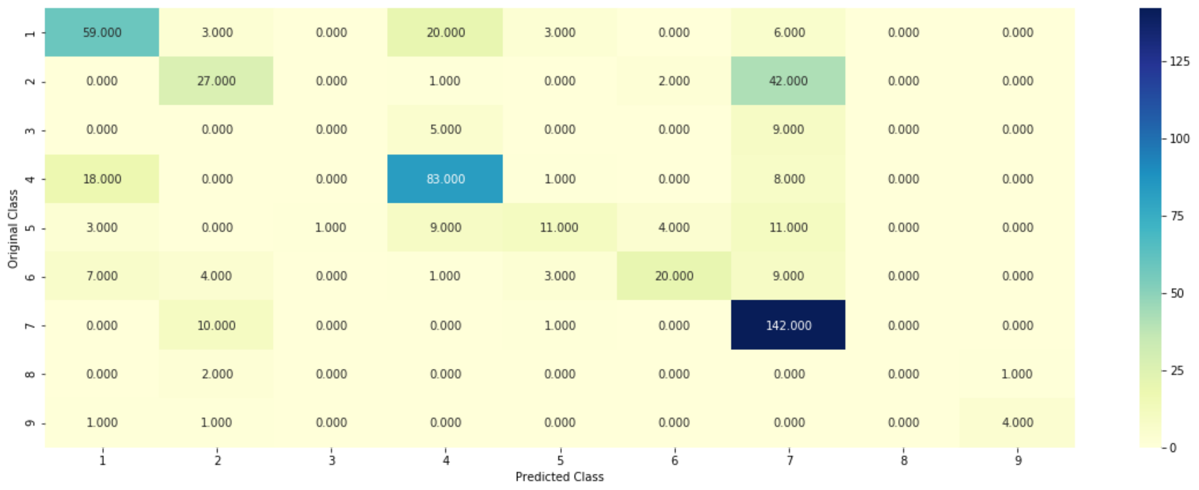


```
For values of best alpha =  0.0001 The train log loss is: 0.41883623133906267
For values of best alpha =  0.0001 The cross validation log loss is: 1.077160
4908005332
For values of best alpha =  0.0001 The test log loss is: 1.070800785276452
```

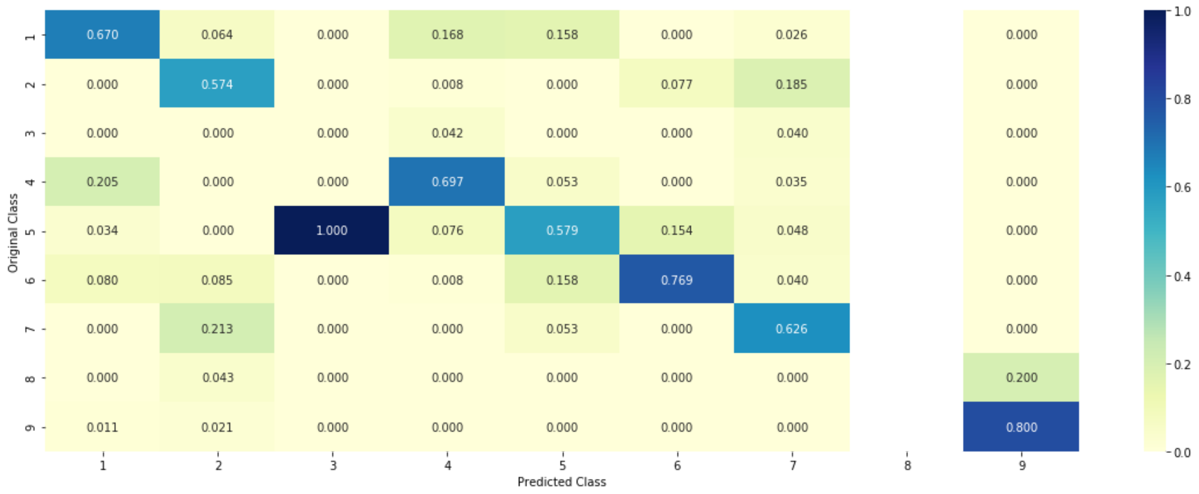
4.3.2.2. Testing model with best hyper parameters

In [73]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_onehotCo
ding, cv_y, clf)
```

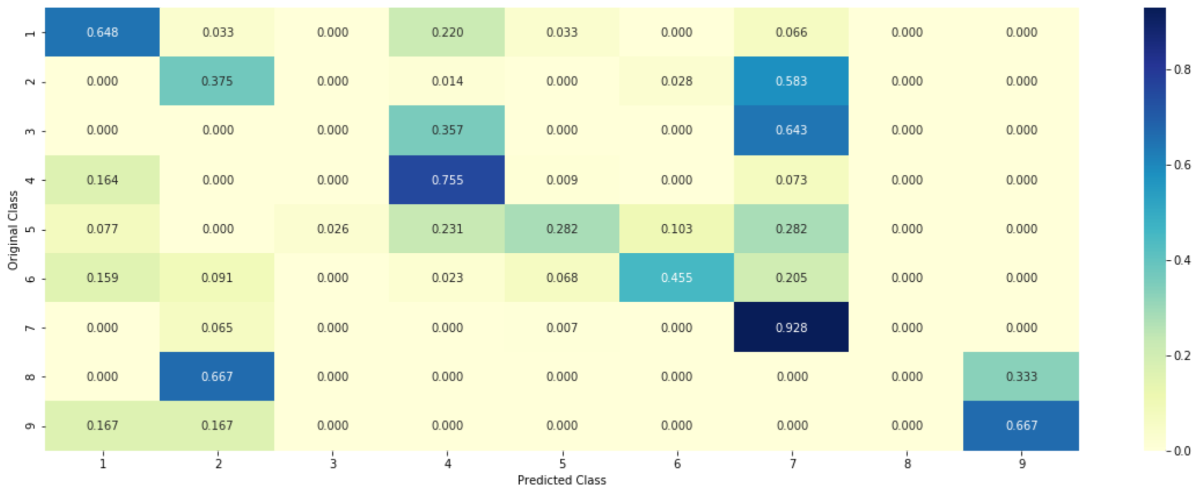
```
Log loss : 1.0771604908005332
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [74]:
```
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3731 0.0468 0.0146 0.4371 0.0395 0.0243 0.
0521 0.0074 0.0051]]
Actual Class : 4
--------------------------------------------------
81 Text feature [iii] present in test data point [True]
90 Text feature [suppressor] present in test data point [True]
199 Text feature [putative] present in test data point [True]
207 Text feature [missense] present in test data point [True]
245 Text feature [nonsense] present in test data point [True]
248 Text feature [weight] present in test data point [True]
289 Text feature [frameshift] present in test data point [True]
304 Text feature [density] present in test data point [True]
312 Text feature [inactivation] present in test data point [True]
326 Text feature [indicate] present in test data point [True]
333 Text feature [functional] present in test data point [True]
355 Text feature [resulting] present in test data point [True]
364 Text feature [mouse] present in test data point [True]
379 Text feature [tumorigenesis] present in test data point [True]
386 Text feature [protein] present in test data point [True]
388 Text feature [process] present in test data point [True]
394 Text feature [domains] present in test data point [True]
401 Text feature [genetic] present in test data point [True]
403 Text feature [assay] present in test data point [True]
412 Text feature [represent] present in test data point [True]
444 Text feature [large] present in test data point [True]
445 Text feature [show] present in test data point [True]
474 Text feature [renal] present in test data point [True]
486 Text feature [comprehensive] present in test data point [True]
Out of the top  500  features  24 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [75]:
```
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[2.974e-01 2.660e-02 4.700e-03 6.191e-01 2.02
0e-02 1.940e-02 8.200e-03
  4.000e-03 4.000e-04]]
Actual Class : 1
--------------------------------------------------
90 Text feature [suppressor] present in test data point [True]
162 Text feature [stability] present in test data point [True]
169 Text feature [brca1] present in test data point [True]
175 Text feature [defect] present in test data point [True]
177 Text feature [mammalian] present in test data point [True]
190 Text feature [mice] present in test data point [True]
198 Text feature [loss] present in test data point [True]
217 Text feature [analysed] present in test data point [True]
252 Text feature [cannot] present in test data point [True]
269 Text feature [incidence] present in test data point [True]
276 Text feature [nucleus] present in test data point [True]
291 Text feature [endogenous] present in test data point [True]
297 Text feature [deleted] present in test data point [True]
318 Text feature [frequent] present in test data point [True]
326 Text feature [indicate] present in test data point [True]
327 Text feature [ref] present in test data point [True]
332 Text feature [repair] present in test data point [True]
333 Text feature [functional] present in test data point [True]
339 Text feature [co] present in test data point [True]
340 Text feature [plasmid] present in test data point [True]
344 Text feature [plasmids] present in test data point [True]
347 Text feature [tumours] present in test data point [True]
379 Text feature [tumorigenesis] present in test data point [True]
386 Text feature [protein] present in test data point [True]
388 Text feature [process] present in test data point [True]
392 Text feature [promotes] present in test data point [True]
400 Text feature [displayed] present in test data point [True]
401 Text feature [genetic] present in test data point [True]
403 Text feature [assay] present in test data point [True]
412 Text feature [represent] present in test data point [True]
421 Text feature [iarc] present in test data point [True]
430 Text feature [brca2] present in test data point [True]
436 Text feature [control] present in test data point [True]
437 Text feature [risk] present in test data point [True]
440 Text feature [fibroblasts] present in test data point [True]
441 Text feature [correlation] present in test data point [True]
444 Text feature [large] present in test data point [True]
445 Text feature [show] present in test data point [True]
449 Text feature [level] present in test data point [True]
459 Text feature [phenotype] present in test data point [True]
479 Text feature [strongly] present in test data point [True]
484 Text feature [defective] present in test data point [True]
486 Text feature [comprehensive] present in test data point [True]
488 Text feature [therefore] present in test data point [True]
494 Text feature [null] present in test data point [True]
498 Text feature [strong] present in test data point [True]
Out of the top  500  features  46 are present in query point
```

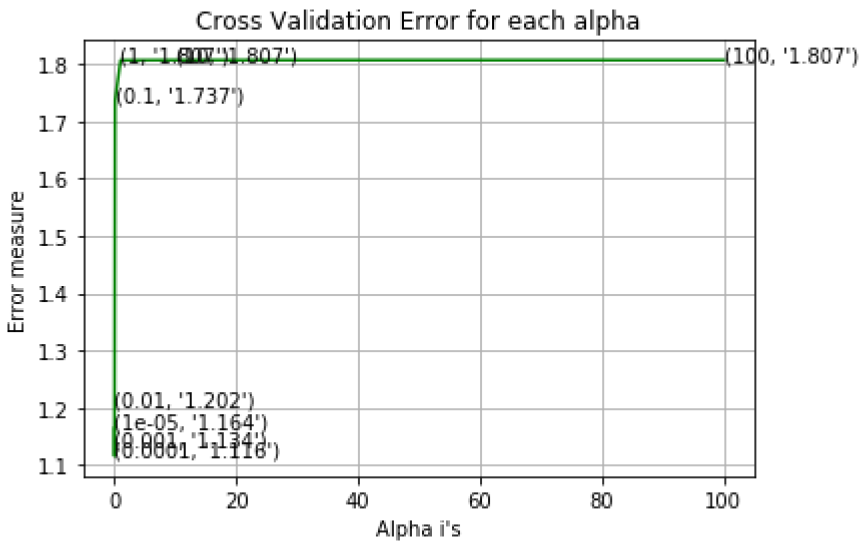
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [76]:
```
# read more about support vector machines with linear kernals here http://scik
it-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_func
tion_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.164350978865992
for C = 0.0001
Log Loss : 1.1156626150010351
for C = 0.001
Log Loss : 1.133579798855881
for C = 0.01
Log Loss : 1.2024679598986974
for C = 0.1
Log Loss : 1.7367077821485868
for C = 1
Log Loss : 1.8068520310961207
for C = 10
Log Loss : 1.8071387260613037
for C = 100
Log Loss : 1.8071387356906854
```

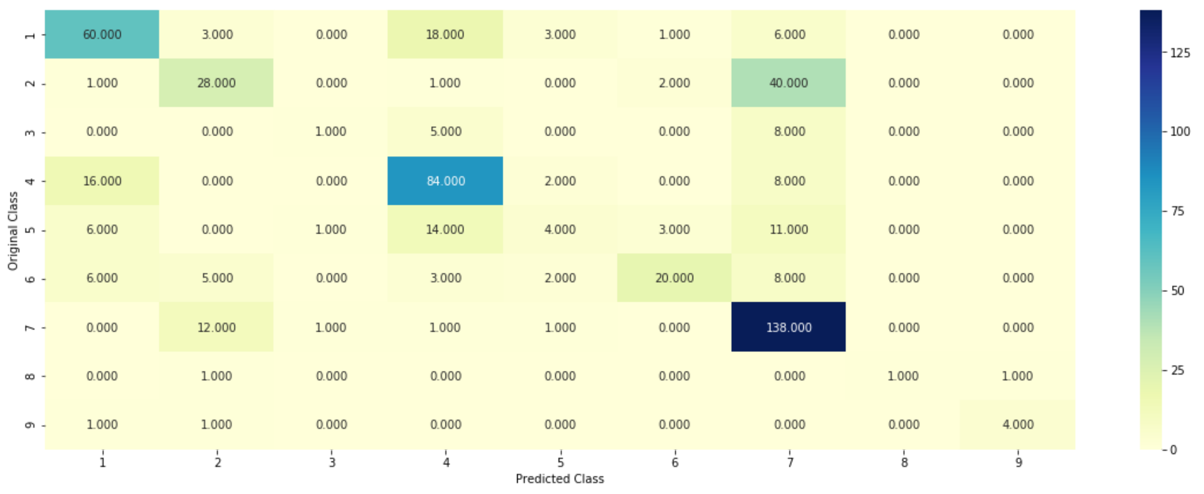


```
For values of best alpha =  0.0001 The train log loss is: 0.4960495556628227
For values of best alpha =  0.0001 The cross validation log loss is: 1.115662
6150010351
For values of best alpha =  0.0001 The test log loss is: 1.1162048568318508
```

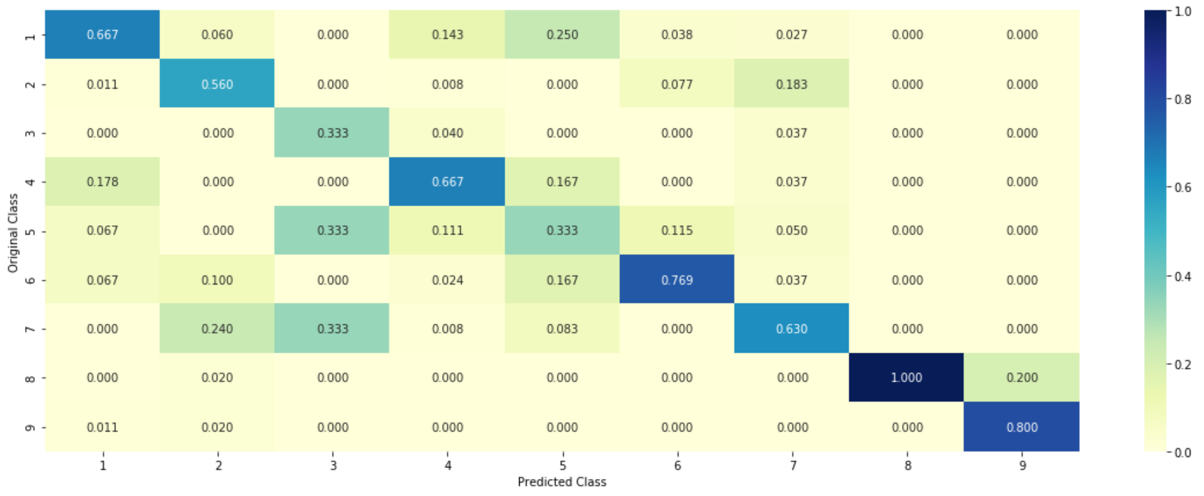
4.4.2. Testing model with best hyper parameters

In [77]:
```
# read more about support vector machines with linear kernals here http://scik
it-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_func
tion_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

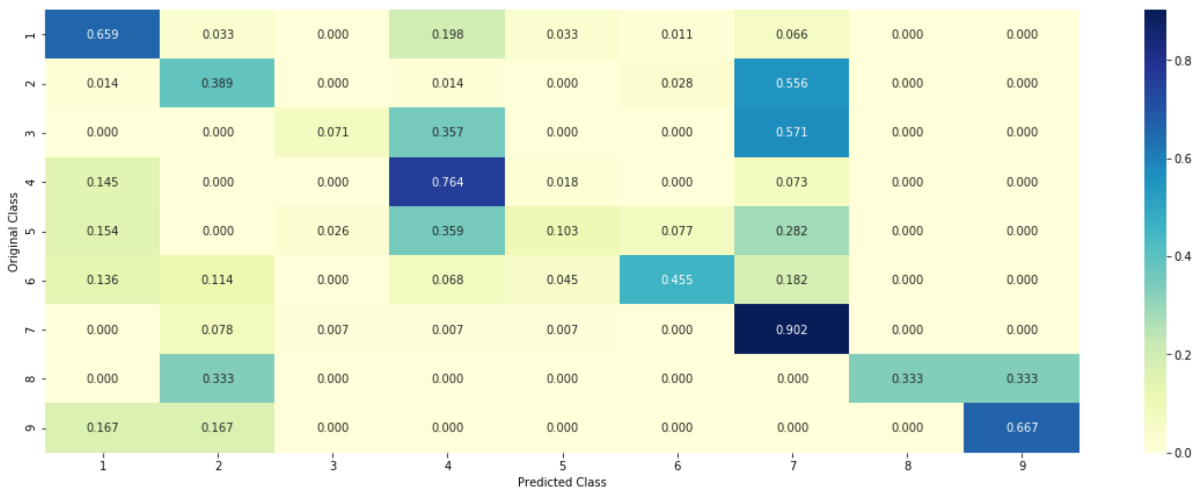
```
Log loss : 1.1156626150010351
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [78]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2432 0.0646 0.0181 0.4402 0.0858 0.028  0.
1094 0.006  0.0047]]
Actual Class : 4
--------------------------------------------------
192 Text feature [iii] present in test data point [True]
203 Text feature [suppressor] present in test data point [True]
212 Text feature [represent] present in test data point [True]
213 Text feature [missense] present in test data point [True]
221 Text feature [indicate] present in test data point [True]
230 Text feature [putative] present in test data point [True]
233 Text feature [weight] present in test data point [True]
236 Text feature [density] present in test data point [True]
239 Text feature [resulting] present in test data point [True]
243 Text feature [nonsense] present in test data point [True]
247 Text feature [frameshift] present in test data point [True]
251 Text feature [lacking] present in test data point [True]
265 Text feature [mouse] present in test data point [True]
267 Text feature [show] present in test data point [True]
268 Text feature [genetic] present in test data point [True]
484 Text feature [large] present in test data point [True]
494 Text feature [regions] present in test data point [True]
Out of the top  500  features  17 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [79]:
```
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1701 0.0504 0.0082 0.6282 0.0474 0.0403 0.
0494 0.004  0.0021]]
Actual Class : 1
--------------------------------------------------
203 Text feature [suppressor] present in test data point [True]
204 Text feature [brca1] present in test data point [True]
205 Text feature [mice] present in test data point [True]
206 Text feature [stability] present in test data point [True]
207 Text feature [loss] present in test data point [True]
209 Text feature [frequent] present in test data point [True]
212 Text feature [represent] present in test data point [True]
218 Text feature [mammalian] present in test data point [True]
221 Text feature [indicate] present in test data point [True]
227 Text feature [co] present in test data point [True]
232 Text feature [defect] present in test data point [True]
248 Text feature [incidence] present in test data point [True]
252 Text feature [endogenous] present in test data point [True]
254 Text feature [strong] present in test data point [True]
267 Text feature [show] present in test data point [True]
268 Text feature [genetic] present in test data point [True]
269 Text feature [triplicate] present in test data point [True]
270 Text feature [deleted] present in test data point [True]
271 Text feature [analysed] present in test data point [True]
272 Text feature [induced] present in test data point [True]
484 Text feature [large] present in test data point [True]
486 Text feature [colony] present in test data point [True]
487 Text feature [level] present in test data point [True]
493 Text feature [nucleus] present in test data point [True]
494 Text feature [regions] present in test data point [True]
495 Text feature [events] present in test data point [True]
497 Text feature [cannot] present in test data point [True]
498 Text feature [phenotype] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [80]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, eps=1e-15))
```

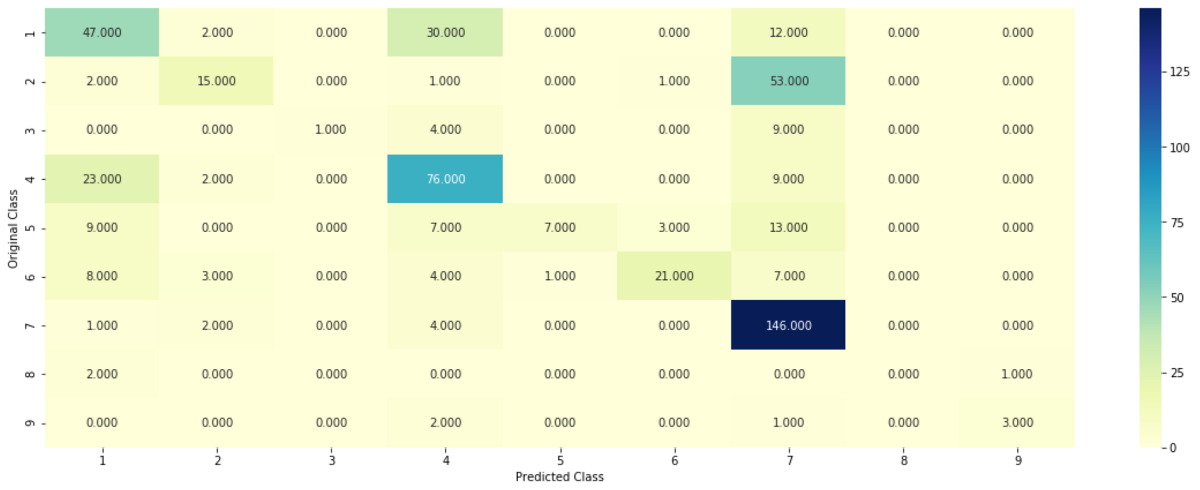
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''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],c
v_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=1
e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test l
og loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1655556264346734
for n_estimators = 100 and max depth =  10
Log Loss : 1.1632258214076927
for n_estimators = 200 and max depth =  5
Log Loss : 1.1560680901794516
for n_estimators = 200 and max depth =  10
Log Loss : 1.1650716818054685
for n_estimators = 500 and max depth =  5
Log Loss : 1.1560959087203098
for n_estimators = 500 and max depth =  10
Log Loss : 1.155226635939851
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1560928471737937
for n_estimators = 1000 and max depth =  10
Log Loss : 1.156849620447531
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1541370907617696
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1584041611026292
For values of best estimator =  2000 The train log loss is: 0.899014842800751
8
For values of best estimator =  2000 The cross validation log loss is: 1.1541
370907617696
For values of best estimator =  2000 The test log loss is: 1.1867607486963705
```

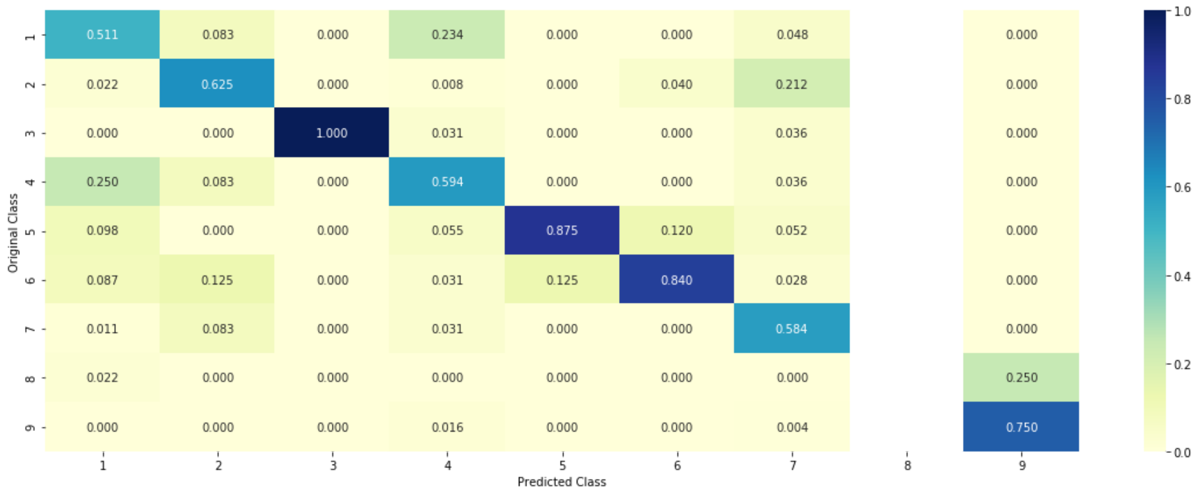
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [81]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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

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

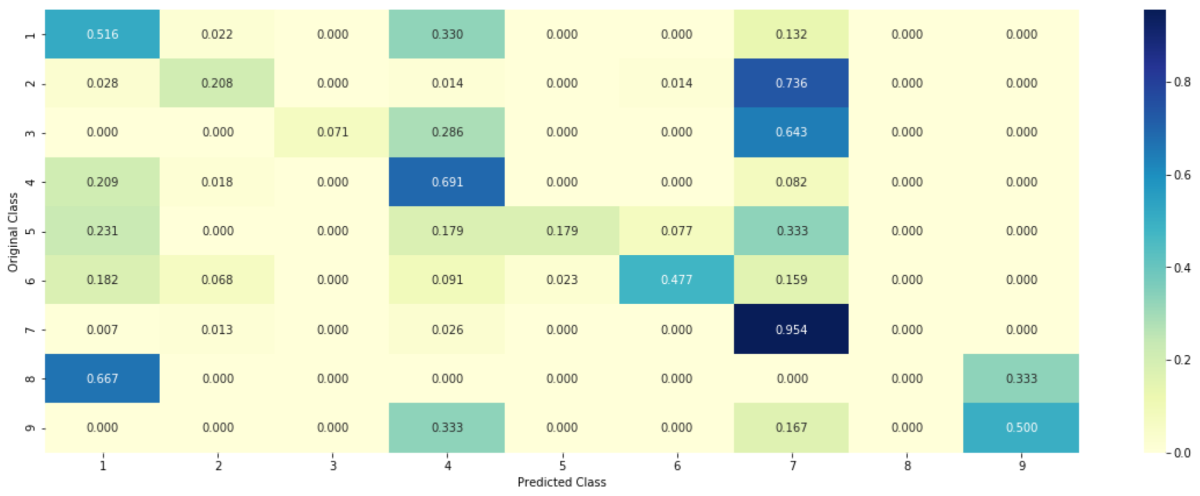
```
Log loss : 1.1541370907617696
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [82]:
```
# 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_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1862 0.0771 0.0224 0.4499 0.0607 0.0521 0.
1389 0.0075 0.0053]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
12 Text feature [missense] present in test data point [True]
13 Text feature [function] present in test data point [True]
17 Text feature [phosphorylation] present in test data point [True]
18 Text feature [suppressor] present in test data point [True]
21 Text feature [signaling] present in test data point [True]
22 Text feature [nonsense] present in test data point [True]
33 Text feature [functional] present in test data point [True]
36 Text feature [cells] present in test data point [True]
37 Text feature [therapy] present in test data point [True]
44 Text feature [growth] present in test data point [True]
48 Text feature [therapeutic] present in test data point [True]
50 Text feature [cell] present in test data point [True]
54 Text feature [receptor] present in test data point [True]
56 Text feature [expression] present in test data point [True]
63 Text feature [protein] present in test data point [True]
68 Text feature [proteins] present in test data point [True]
77 Text feature [phosphatase] present in test data point [True]
80 Text feature [3t3] present in test data point [True]
86 Text feature [proliferation] present in test data point [True]
87 Text feature [truncating] present in test data point [True]
89 Text feature [kinases] present in test data point [True]
Out of the top  100  features  24 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [83]:
```
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_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3141 0.0282 0.0186 0.4206 0.065  0.0805 0.
0592 0.0063 0.0077]]
Actuall Class : 1
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
7 Text feature [activated] present in test data point [True]
8 Text feature [activation] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
13 Text feature [function] present in test data point [True]
18 Text feature [suppressor] present in test data point [True]
20 Text feature [brca1] present in test data point [True]
26 Text feature [constitutively] present in test data point [True]
28 Text feature [oncogenic] present in test data point [True]
31 Text feature [inhibitor] present in test data point [True]
33 Text feature [functional] present in test data point [True]
35 Text feature [loss] present in test data point [True]
36 Text feature [cells] present in test data point [True]
44 Text feature [growth] present in test data point [True]
50 Text feature [cell] present in test data point [True]
51 Text feature [activate] present in test data point [True]
53 Text feature [defective] present in test data point [True]
56 Text feature [expression] present in test data point [True]
57 Text feature [stability] present in test data point [True]
62 Text feature [brca] present in test data point [True]
63 Text feature [protein] present in test data point [True]
64 Text feature [brca2] present in test data point [True]
68 Text feature [proteins] present in test data point [True]
86 Text feature [proliferation] present in test data point [True]
87 Text feature [truncating] present in test data point [True]
92 Text feature [carriers] present in test data point [True]
97 Text feature [inhibition] present in test data point [True]
Out of the top  100  features  27 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [84]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, eps=1e-15))
```

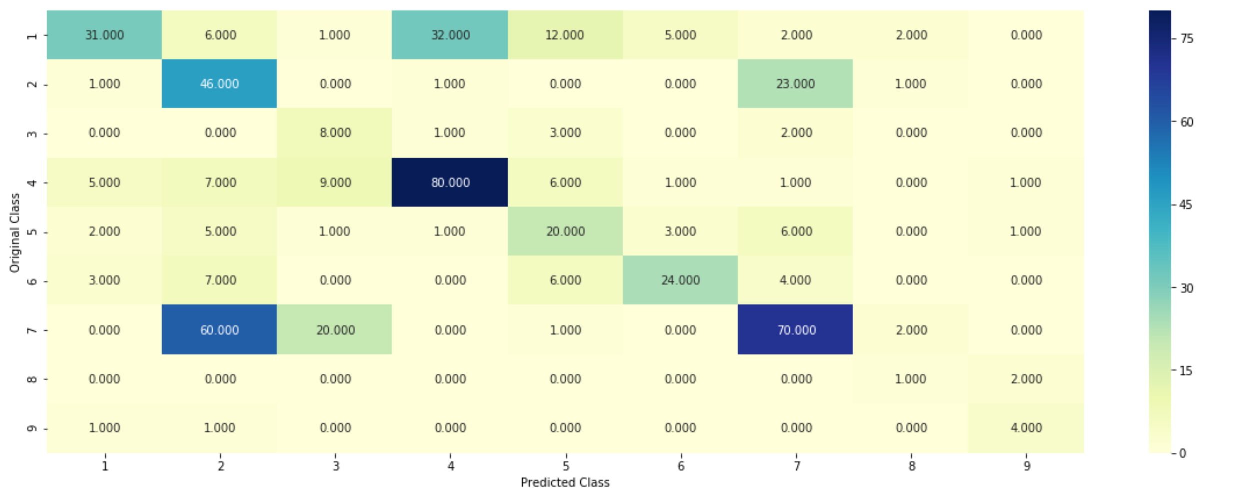
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],c
v_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 vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1630191973837753
for n_estimators = 10 and max depth =  3
Log Loss : 1.6875983918276172
for n_estimators = 10 and max depth =  5
Log Loss : 1.4735606760691977
for n_estimators = 10 and max depth =  10
Log Loss : 2.023237146328503
for n_estimators = 50 and max depth =  2
Log Loss : 1.8210499366547648
for n_estimators = 50 and max depth =  3
Log Loss : 1.465966213338451
for n_estimators = 50 and max depth =  5
Log Loss : 1.368067726634864
for n_estimators = 50 and max depth =  10
Log Loss : 1.8441678271104405
for n_estimators = 100 and max depth =  2
Log Loss : 1.6775650350943516
for n_estimators = 100 and max depth =  3
Log Loss : 1.5161748940652868
for n_estimators = 100 and max depth =  5
Log Loss : 1.2907320877465016
for n_estimators = 100 and max depth =  10
Log Loss : 1.7027327518229243
for n_estimators = 200 and max depth =  2
Log Loss : 1.7047956910819742
for n_estimators = 200 and max depth =  3
Log Loss : 1.5213074562183757
for n_estimators = 200 and max depth =  5
Log Loss : 1.3728587937355439
for n_estimators = 200 and max depth =  10
Log Loss : 1.7093837659833533
for n_estimators = 500 and max depth =  2
Log Loss : 1.7273385216723787
for n_estimators = 500 and max depth =  3
Log Loss : 1.5666375851914
for n_estimators = 500 and max depth =  5
Log Loss : 1.3803753729960229
for n_estimators = 500 and max depth =  10
Log Loss : 1.7863142383062056
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7048498167756883
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5946437082636886
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3861636271702247
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7834172689253782
For values of best alpha =  100 The train log loss is: 0.05406315192348259
For values of best alpha =  100 The cross validation log loss is: 1.290732087
7465016
For values of best alpha =  100 The test log loss is: 1.3084742824495443
```

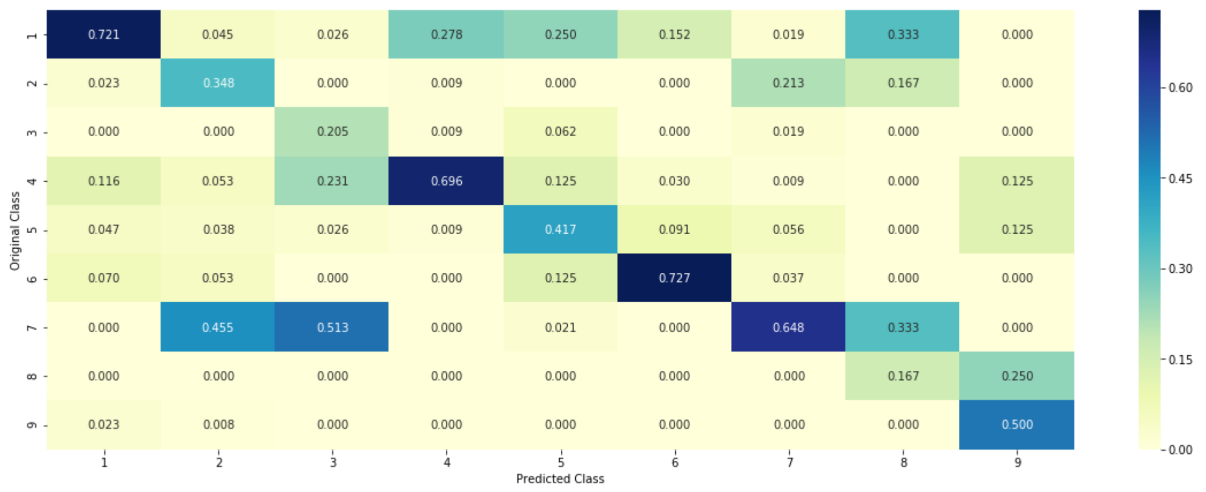
4.5.4. Testing model with best hyper parameters (Response Coding)

In [85]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimat
ors=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_sta
te=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_respons
eCoding,cv_y, clf)
```

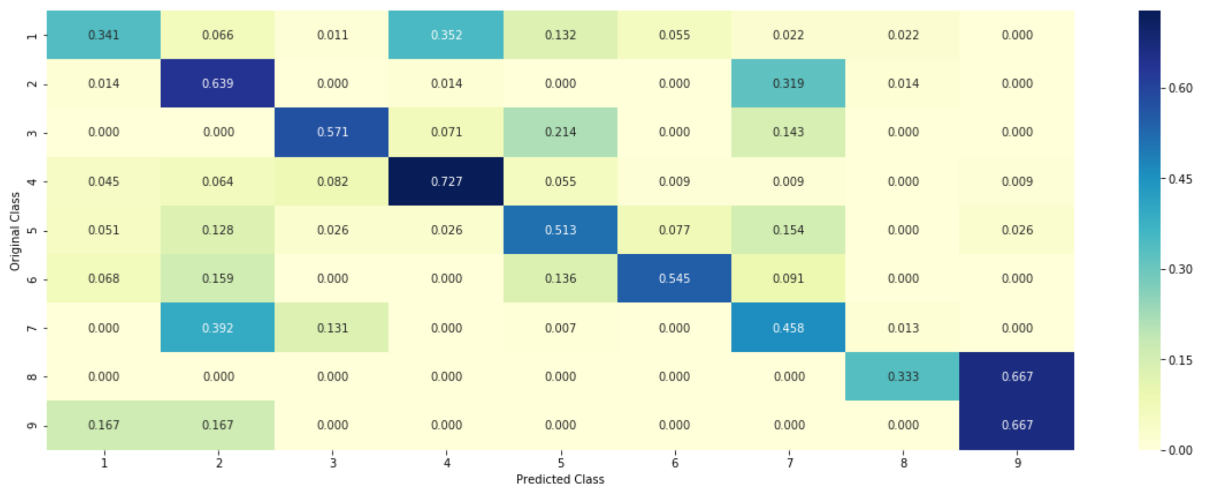
```
Log loss : 1.2907320877465016
Number of mis-classified points : 0.46616541353383456
-------------------- Confusion matrix --------------------
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [86]:
```
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].reshap
e(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: [[0.217  0.0288 0.1194 0.4592 0.0318 0.0534 0.
0092 0.0426 0.0386]]
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
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

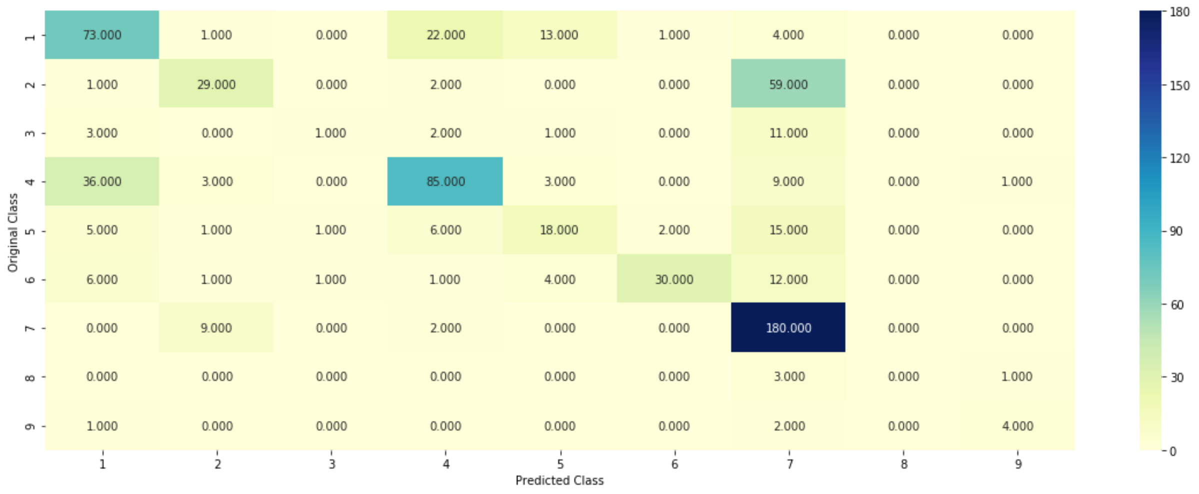
In [87]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual 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: [[0.2156 0.0156 0.1096 0.5461 0.0245 0.0466 0.
0072 0.0193 0.0155]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

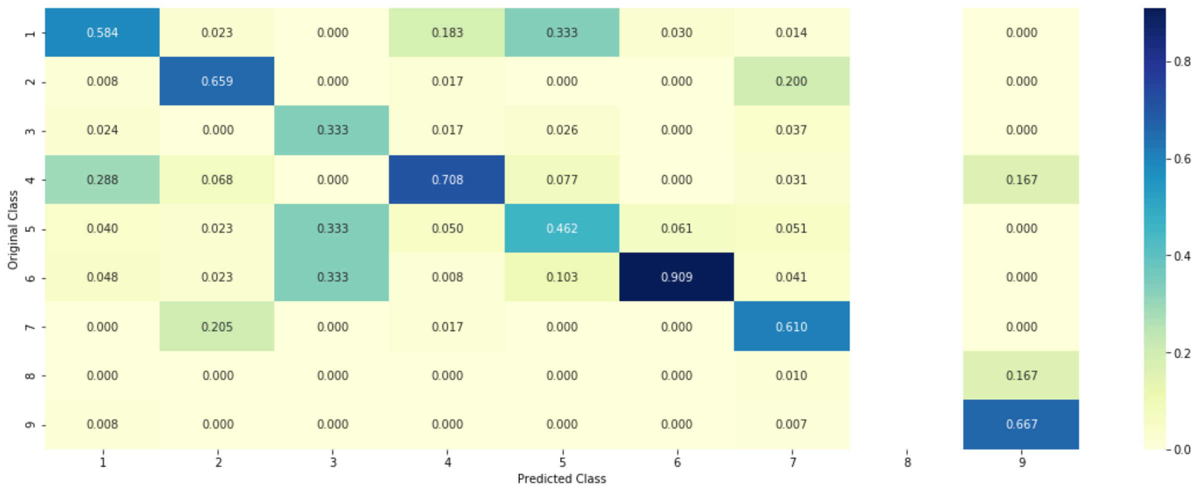
4.7.3 Maximum Voting classifier

In [89]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Votin
gClassifier.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.pre
dict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_p
roba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predi
ct_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_onehotCodin
g))
```

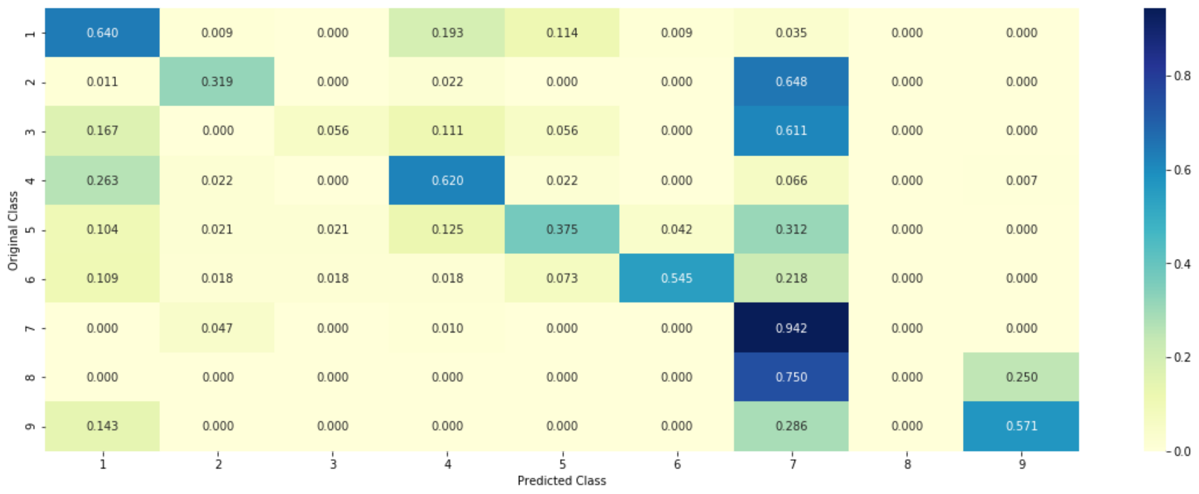
```
Log loss (train) on the VotingClassifier : 0.8342079351353936
Log loss (CV) on the VotingClassifier : 1.222666540824057
Log loss (test) on the VotingClassifier : 1.2218891661520908
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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



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



Conclusion

In [2]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names =["Models","Train","CV","Test","Misclassified(%)"]

x.add_row(["Naive Bayes (One hot coding)",0.52,1.24,1.23,0.39])
x.add_row(["KNN (Response)",0.48,1.08,1.06,0.34])
x.add_row(["LR(Class balanced) one hot coding",0.42,1.08,1.07,0.35])
x.add_row(["LR(Class unbalanced) one hot coding",0.41,1.07,1.07,0.34])
x.add_row(["Lr SVM one hot encoding",0.49,1.11,1.11,0.36])
x.add_row(["Random Forest one hot coding",0.89,1.15,1.18,0.40])
x.add_row(["Random Forest Response coding",0.54,1.29,1.30,0.46])
x.add_row(["Maximum Voting Classifier",0.83,1.22,1.22,0.36])

print(x)
print("\n")
```

```
+-------------------------------------+-------+------+------+----------------
--+
|                Models               | Train |  CV  | Test | Misclassified
(%) |
+-------------------------------------+-------+------+------+----------------
--+
|     Naive Bayes (One hot coding)    |  0.52 | 1.24 | 1.23 |       0.39
|
|            KNN (Response)           |  0.48 | 1.08 | 1.06 |       0.34
|
|  LR(Class balanced) one hot coding  |  0.42 | 1.08 | 1.07 |       0.35
|
| LR(Class unbalanced) one hot coding |  0.41 | 1.07 | 1.07 |       0.34
|
|       Lr SVM one hot encoding       |  0.49 | 1.11 | 1.11 |       0.36
|
|     Random Forest one hot coding    |  0.89 | 1.15 | 1.18 |       0.4
|
|    Random Forest Response coding    |  0.54 | 1.29 | 1.3  |       0.46
|
|      Maximum Voting Classifier      |  0.83 | 1.22 | 1.22 |       0.36
|
+-------------------------------------+-------+------+------+----------------
--+
```