PersonalizedCancerDiagnosis

November 24, 2019

Personalized cancer diagnosis

1. Business Problem

1.1. Description
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)
Download training_variants.zip and training_text.zip from Kaggle.
Context:
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462
Problem statement :
Classify the given genetic variations/mutations based on evidence from text-based clinical literature.
1.2. Source/Useful Links
Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.

- Both these data files are have a common column called ID
- Data file's information:
  training_variants (ID , Gene, Variations, Class)
  training_text (ID, Text)

2.1.2. Example Data Point
training_variants
ID,Gene,Variation,Class 0,FAM58A,Truncating Mutations,1 1,CBL,W802*,2 2,CBL,Q249E,2 ...
training_text
ID,Text 0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem
2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation
Metric(s): * Multi class log-loss * Confusion matrix
2.2.3. Machine Learing Objectives and Constraints
Objective: Predict the probability of each data-point belonging to each of the nine classes.
Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

[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

[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']
```

[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

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

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

[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

[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
```

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 135.6424032 seconds
```

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

[6]:

	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

```
                                                TEXT
0  cyclin dependent kinases cdks regulate variety...
1  abstract background non small cell lung cancer...
2  abstract background non small cell lung cancer...
3  recent evidence demonstrated acquired uniparen...
4  oncogenic mutations monomeric casitas b lineag...
```

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

[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
```

[8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +'␣
↪'+result['Variation']
```

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

[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)

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

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

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

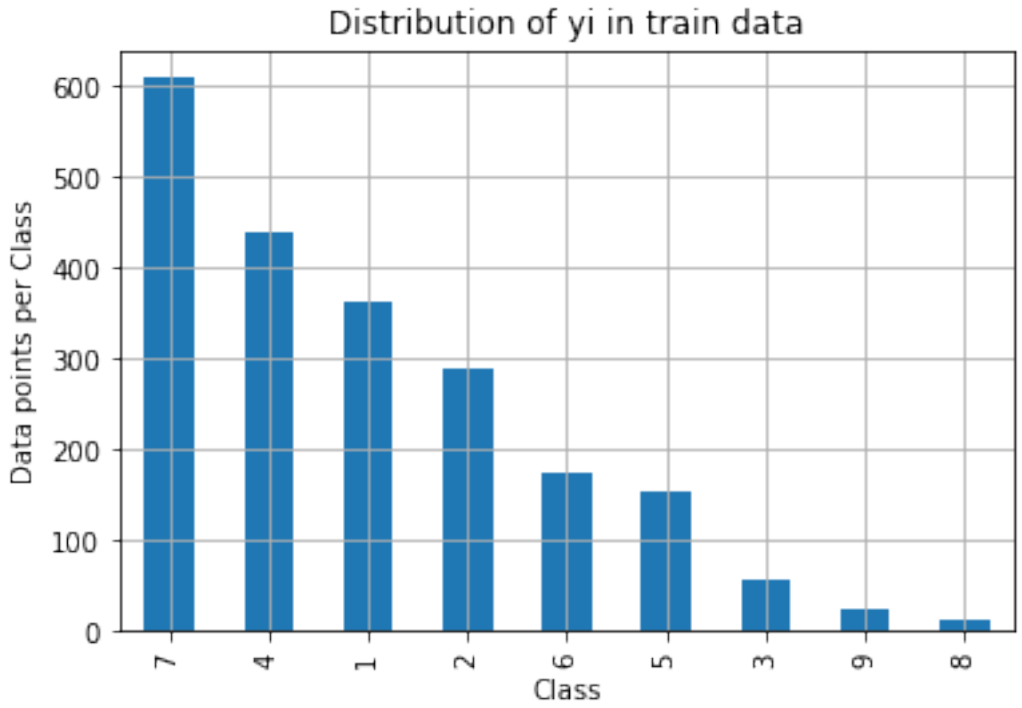
[11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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

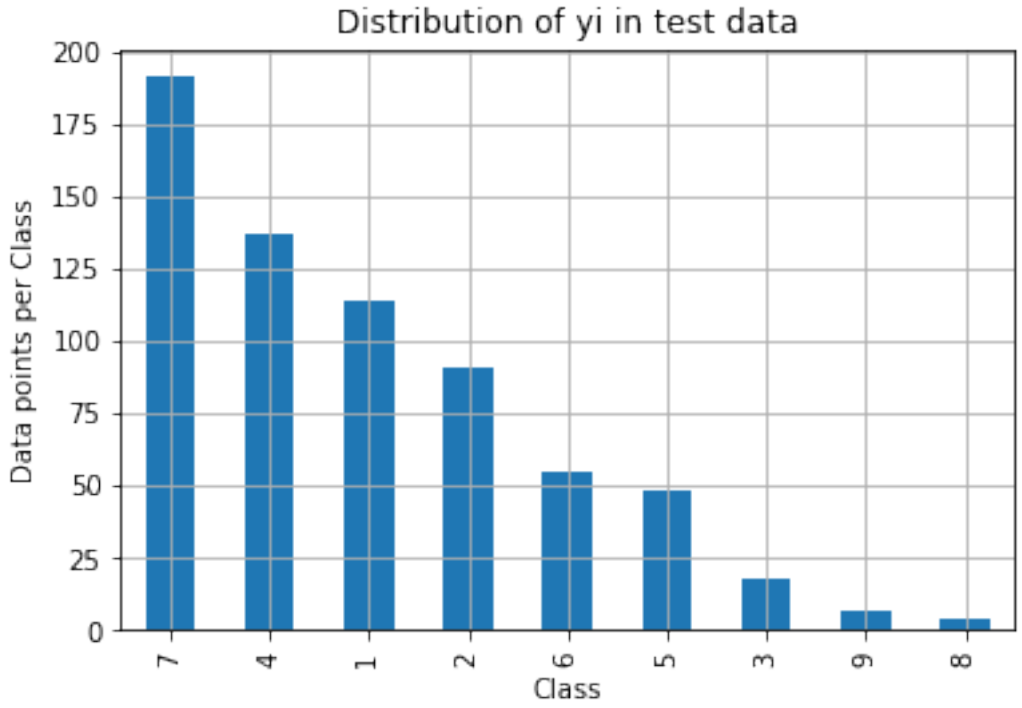
[12]:
```
# it returns a dict, keys as class labels and values as the number of data␣
 ↪points in that class
train_class_distribution = train_df['Class'].value_counts()
test_class_distribution = test_df['Class'].value_counts()
cv_class_distribution = cv_df['Class'].value_counts()

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

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

print('-'*80)
my_colors = 'rgbkymc'
```

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

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

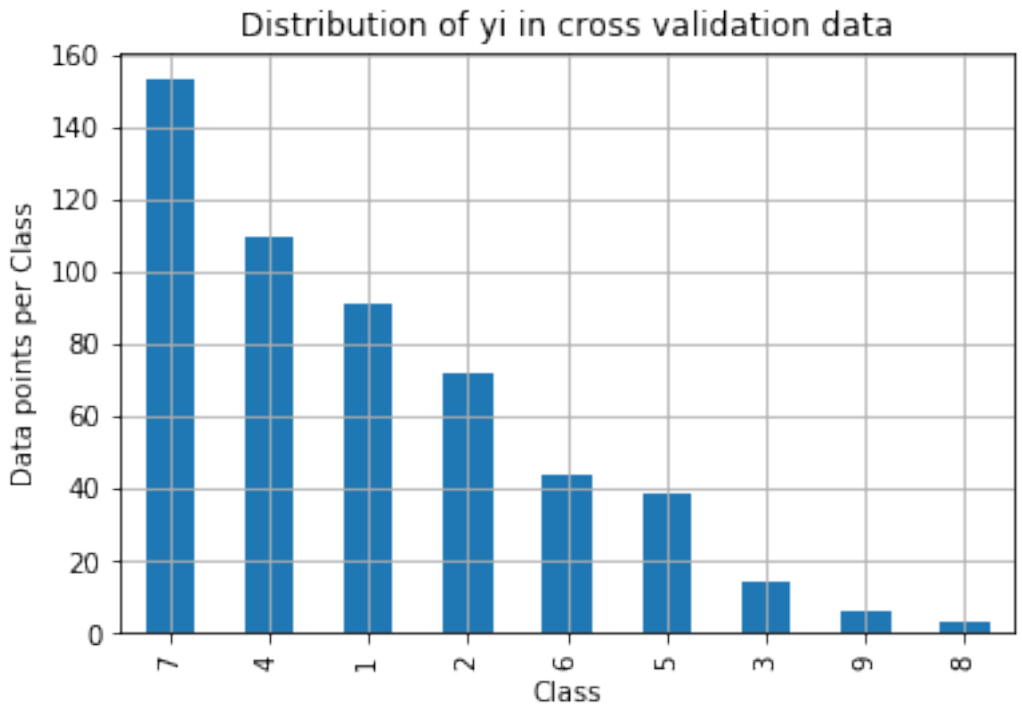



```
Number of data points in class 1 : 609 ( 28.672 %)
Number of data points in class 2 : 439 ( 20.669 %)
Number of data points in class 3 : 363 ( 17.09 %)
Number of data points in class 4 : 289 ( 13.606 %)
Number of data points in class 5 : 176 ( 8.286 %)
Number of data points in class 6 : 155 ( 7.298 %)
```

```
Number of data points in class 7 : 57 ( 2.684 %)
Number of data points in class 8 : 24 ( 1.13 %)
Number of data points in class 9 : 12 ( 0.565 %)
-------------------------------------------------------------------------------
```




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




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

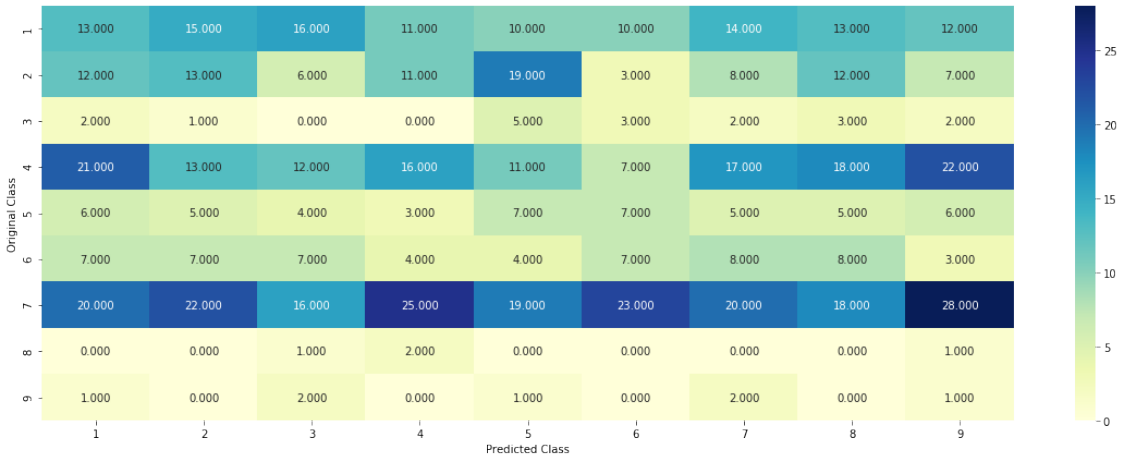
[13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i␣
 ↪are predicted class j

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

    # C = [[1, 2],
```

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

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

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

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

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

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

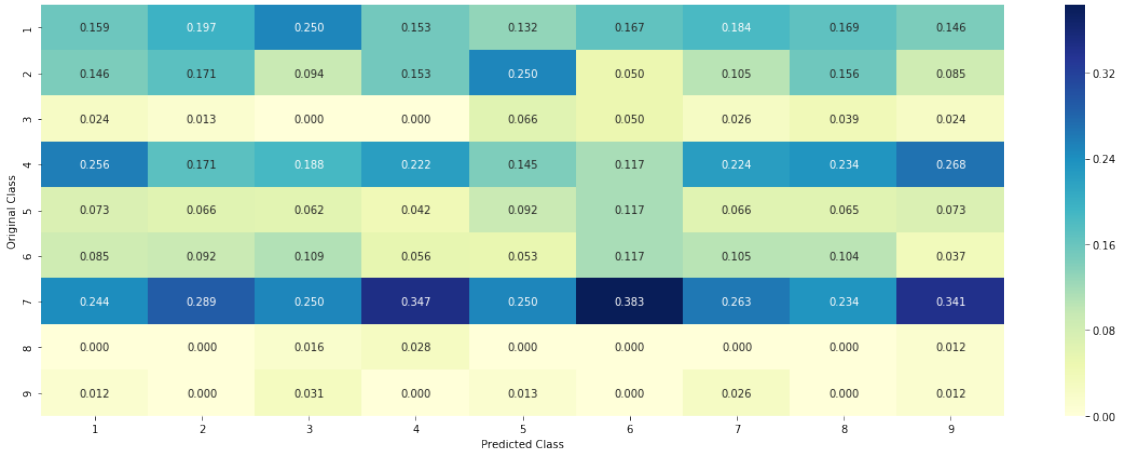
```
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

[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 their␣
 ↪sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

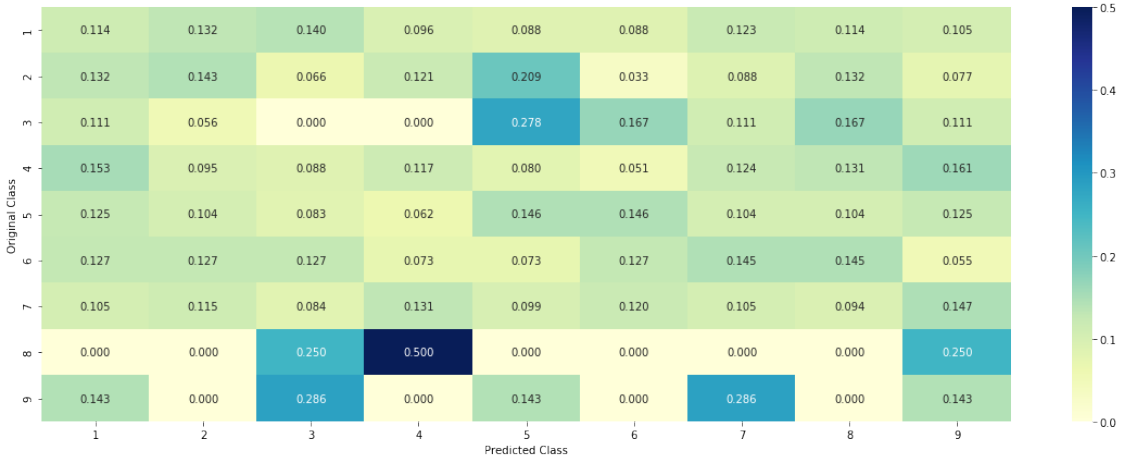
```
Log loss on Cross Validation Data using Random Model 2.478097482780709
Log loss on Test Data using Random Model 2.450563292786766
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

[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␣
 ↪class1 + 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)␣
 ↪representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

```
    # Q61L                                          3
    # S222D                                         2
    # P130S                                         2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

            # cls_cnt.shape[0](numerator) will contain the number of time that␣
↪particular feature 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': [0.200757575757575, 0.037878787878788, 0.
↪06818181818181777, 0.13636363636363635, 0.25, 0.1931818181818181, 0.
↪037878787878788, 0.037878787878788, 0.037878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.
↪061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.
↪066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.
↪056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.
↪06818181818181777, 0.06818181818181777, 0.0625, 0.3465909090909012, 0.
↪0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.
↪060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.
↪34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.
↪060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.
↪069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.
↪062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.
↪062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.
↪072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.
↪066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.
↪066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.
↪073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.
↪080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.
↪066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

when we 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?

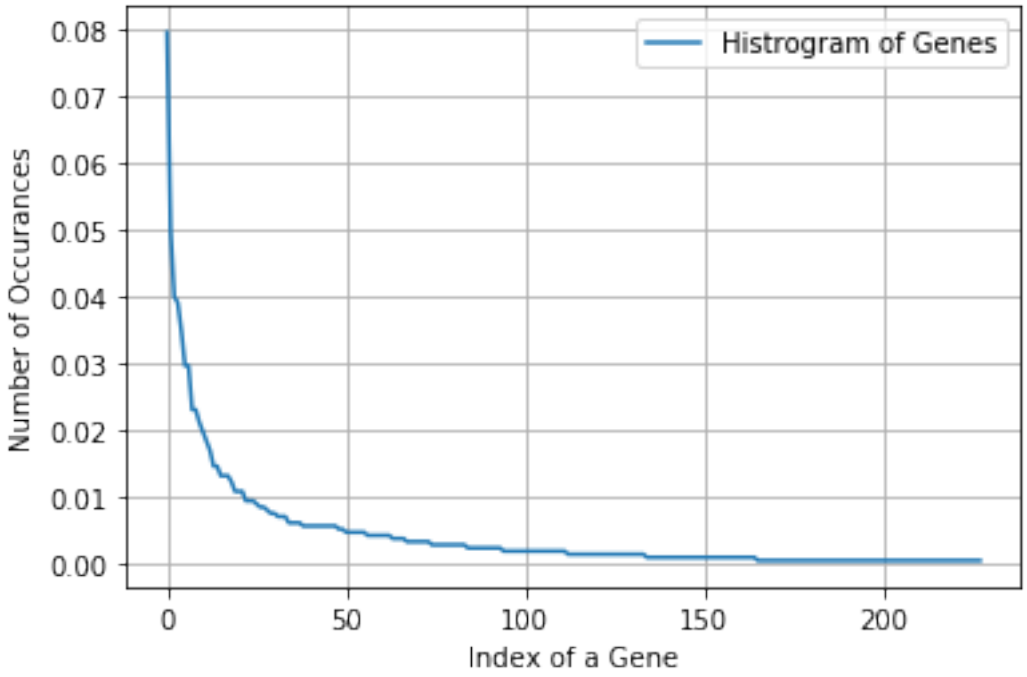
[16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 228
BRCA1     169
TP53      106
EGFR       85
PTEN       83
BRCA2      74
KIT        63
BRAF       63
ERBB2      49
ALK        49
PDGFRA     45
Name: Gene, dtype: int64
```

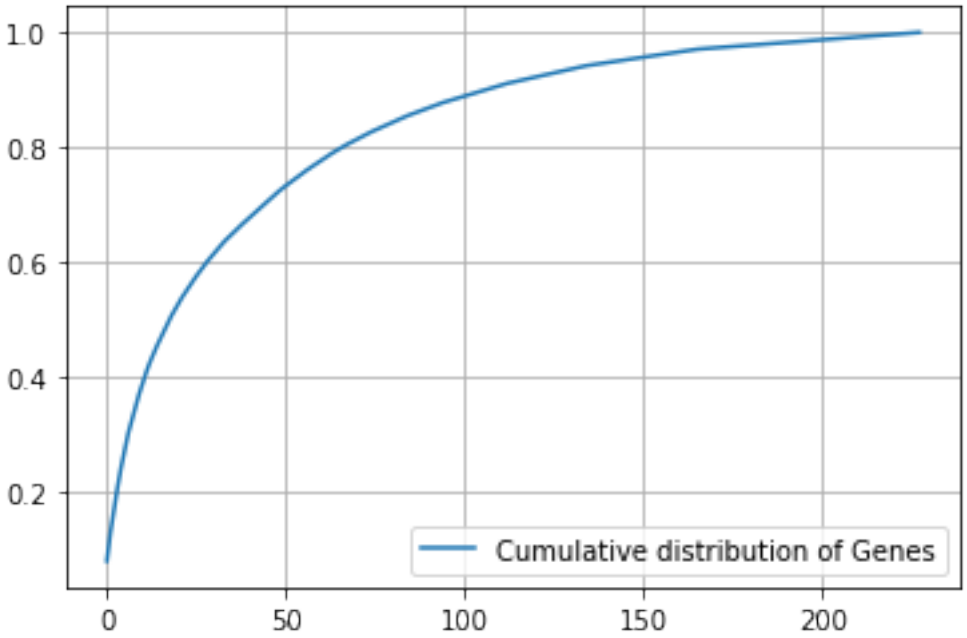
[17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes␣
 ↪in the train data, and they are distibuted as follows",)
```

```
Ans: There are 228 different categories of genes in the train data, and they are
distibuted as follows
```

[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()
```



[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/

One hot Encoding

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.

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

[21]:
```
print("train_gene_feature_responseCoding is converted feature using respone␣
 ↪coding method. The shape of gene feature:",␣
 ↪train_gene_feature_responseCoding.shape)
```

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

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

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

[23]:
```
2272     PTEN
2337     JAK2
3012      KIT
1415    FGFR3
545     SMAD2
Name: Gene, dtype: object
```

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

[25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot␣
 ↪encoding method. The shape of gene feature:",␣
 ↪train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding
method. The shape of gene feature: (2124, 227)
```

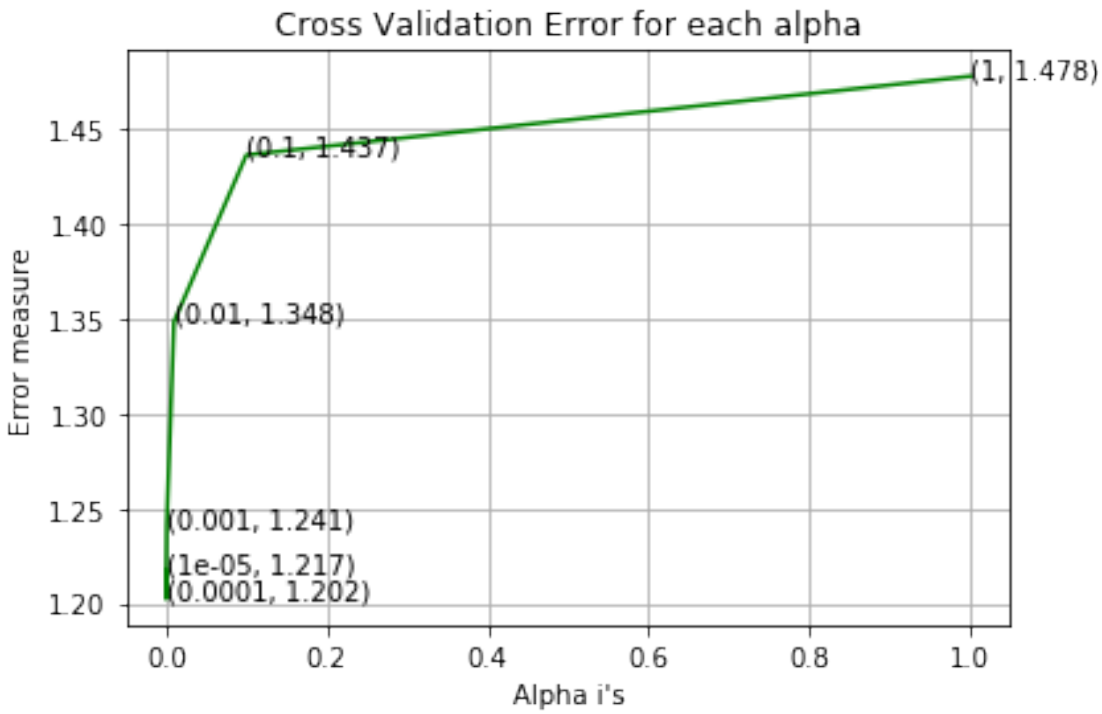
Q4. How good is this gene feature in predicting y_i?

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.

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2169788882727053
For values of alpha =  0.0001 The log loss is: 1.2022874720760017
For values of alpha =  0.001 The log loss is: 1.2409692133164814
For values of alpha =  0.01 The log loss is: 1.3484676917093859
For values of alpha =  0.1 The log loss is: 1.436541998686817
For values of alpha =  1 The log loss is: 1.4781085179805062
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0031002932732616
For values of best alpha =  0.0001 The cross validation log loss is:
1.2022874720760017
For values of best alpha =  0.0001 The test log loss is: 1.1764926863911394
```

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.

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  228  genes
in train dataset?
Ans
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 509 out of  532 : 95.67669172932331
```

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?

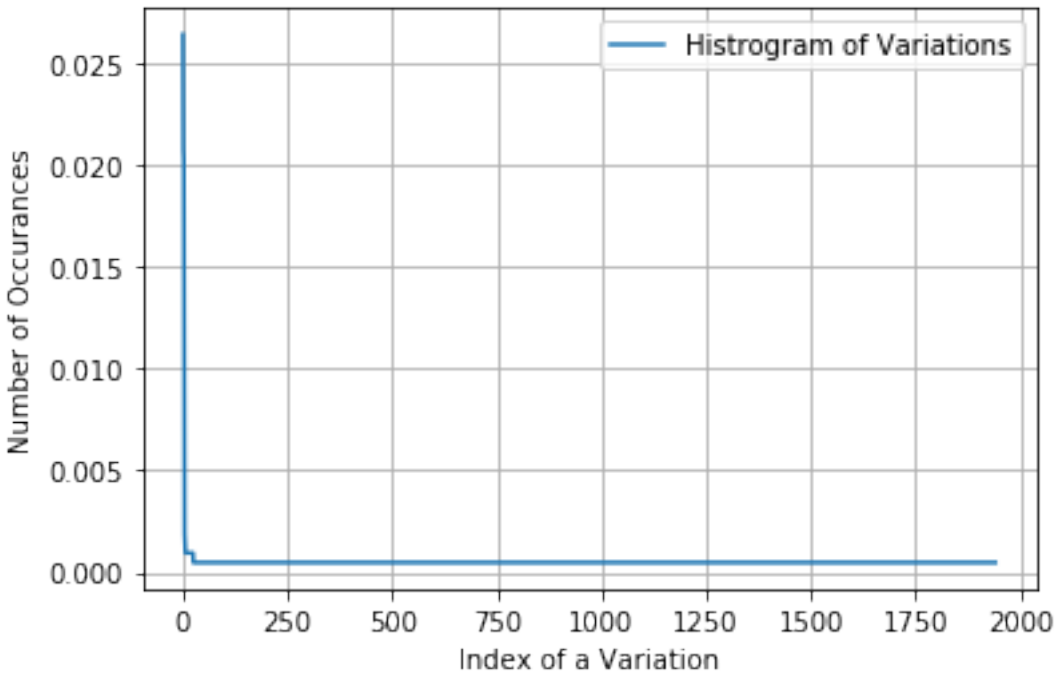
[28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1941
Truncating_Mutations    56
Deletion                44
Amplification           43
Fusions                 20
Overexpression           4
T58I                     3
E330K                    2
G13D                     2
EWSR1-ETV1_Fusion        2
E17K                     2
Name: Variation, dtype: int64
```

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

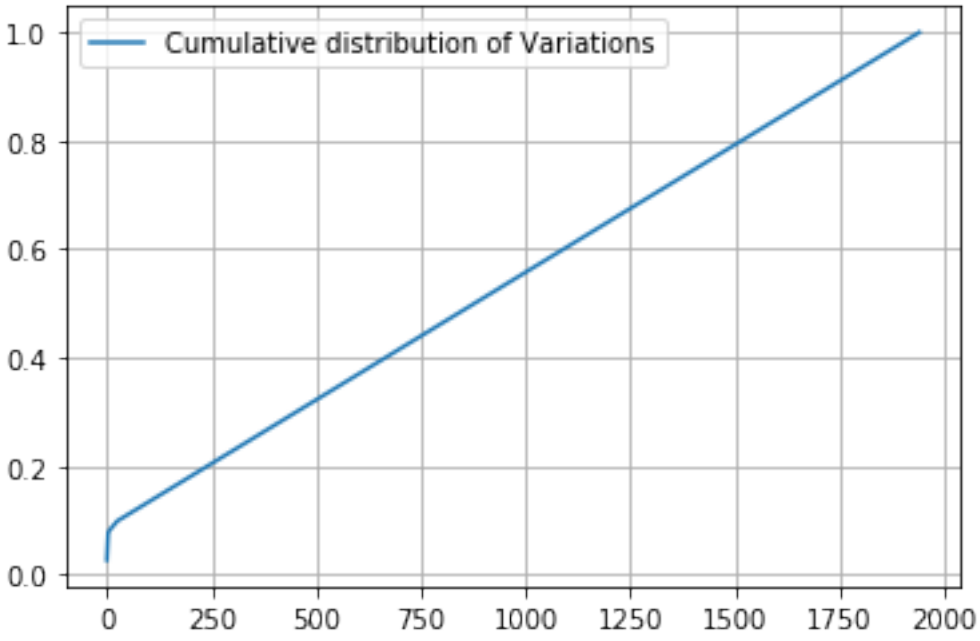
```
Ans: There are 1941 different categories of variations in the train data, and
they are distibuted as follows
```

[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()
```



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

```
[0.02636535 0.04708098 0.0673258  ... 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/

One hot Encoding

Response coding

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

[32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", cv_df))
```

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

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

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

[35]:
```
print("train_variation_feature_onehotEncoded is converted feature using the␣
 ↪onne-hot encoding method. The shape of Variation feature:",␣
 ↪train_variation_feature_onehotCoding.shape)
```

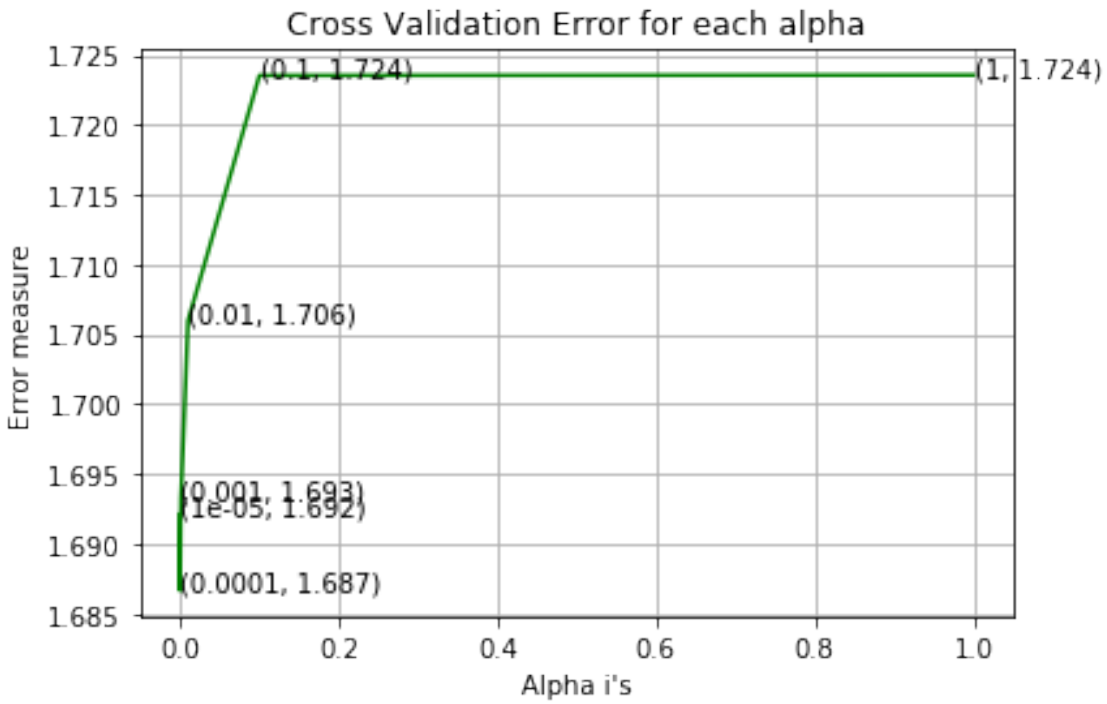
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1973)
```

Q10. How good is this Variation feature in predicting y_i?
Let's build a model just like the earlier!

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

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

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
```

```
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.6920931677358548
For values of alpha =  0.0001 The log loss is: 1.68663391913646
For values of alpha =  0.001 The log loss is: 1.6931846340084546
For values of alpha =  0.01 The log loss is: 1.7059675449528828
For values of alpha =  0.1 The log loss is: 1.7235335278546948
For values of alpha =  1 The log loss is: 1.7235606809690918
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7425329226485963
For values of best alpha =  0.0001 The cross validation log loss is:
1.68663391913646
For values of best alpha =  0.0001 The test log loss is: 1.7057995868840634
```

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.

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

```
Q12. How many data points are covered by total  1941  genes in test and cross
validation data sets?
Ans
1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 62 out of  532 : 11.654135338345863
```

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?

[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
```

[39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/
 ↪(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/
 ↪len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

[40]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer = TfidfVectorizer(max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.
 ↪fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns␣
 ↪(1*number of 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 : 1000
```

[41]:
```
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)
```

[42]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

[43]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/
 ↪train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
 ↪test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/
 ↪cv_text_feature_responseCoding.sum(axis=1)).T
```

[44]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding,␣
 ↪axis=0)
```

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

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

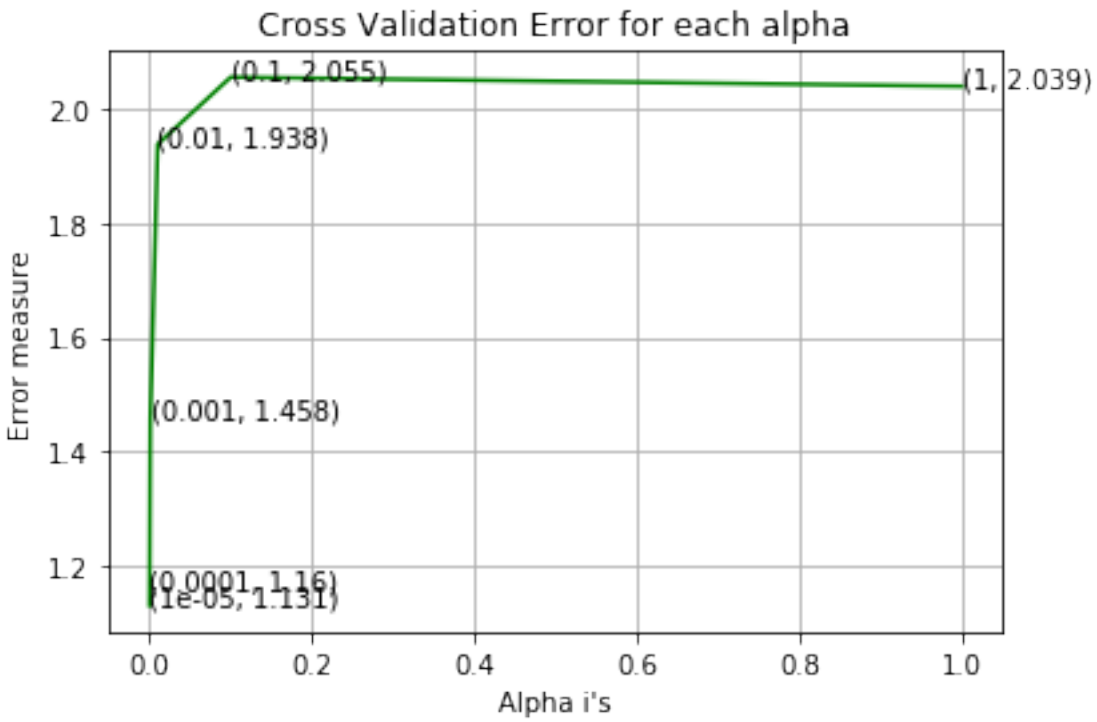
[45]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] ,␣
 ↪reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

[46]:
```
# Number of words for a given frequency.
#print(Counter(sorted_text_occur))
```

[47]:
```
# Train a Logistic regression+Calibration model using text features whicha re␣
 ↪on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1313830236123523
For values of alpha =  0.0001 The log loss is: 1.1603274917713868
For values of alpha =  0.001 The log loss is: 1.458401271824388
For values of alpha =  0.01 The log loss is: 1.9380483013306218
For values of alpha =  0.1 The log loss is: 2.055337918779392
For values of alpha =  1 The log loss is: 2.039428999236976
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7394064353243744
For values of best alpha =  1e-05 The cross validation log loss is:
1.1313830236123523
For values of best alpha =  1e-05 The test log loss is: 1.015621259073822
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it seems like!

[48]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

[49]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train␣
 ↪data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in␣
 ↪train data")
```

```
3.43 % of word of test data appeared in train data
3.66 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

[50]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

[51]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

[52]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

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

Stacking the three types of features

[53]:
```
# merging gene, variance and text features

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

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

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

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

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


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

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

[54]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",␣
 ↪train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ",␣
 ↪test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data␣
 ↪=", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3200)
(number of data points * number of features) in test data =  (665, 3200)
(number of data points * number of features) in cross validation data = (532,
3200)
```

[55]:
```
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_responseCoding.shape)
print("(number of data points * number of features) in cross validation data␣
 ↪=", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
```

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

4.1. Base Line Model

4.1.1. Naive Bayes

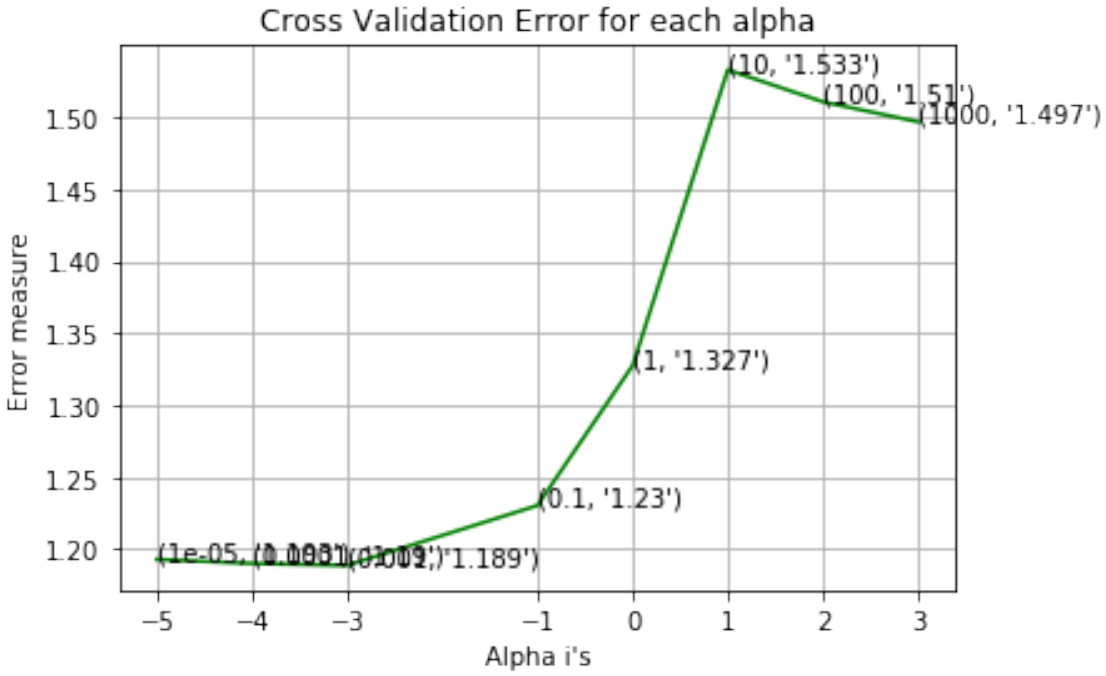
4.1.1.1. Hyper parameter tuning

[56]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/
 ↪stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True,␣
 ↪class_prior=None)

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
```

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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.1930371995624687
for alpha = 0.0001
Log Loss : 1.1902516347609935
for alpha = 0.001
Log Loss : 1.1888342115586885
```

```
for alpha = 0.1
Log Loss : 1.2304151001220163
for alpha = 1
Log Loss : 1.3274763636312443
for alpha = 10
Log Loss : 1.5326275068684156
for alpha = 100
Log Loss : 1.5104183297176046
for alpha = 1000
Log Loss : 1.4965640268890057
```




```
For values of best alpha =  0.001 The train log loss is: 0.5207816391891147
For values of best alpha =  0.001 The cross validation log loss is:
1.1888342115586885
For values of best alpha =  0.001 The test log loss is: 1.1682152289846914
```

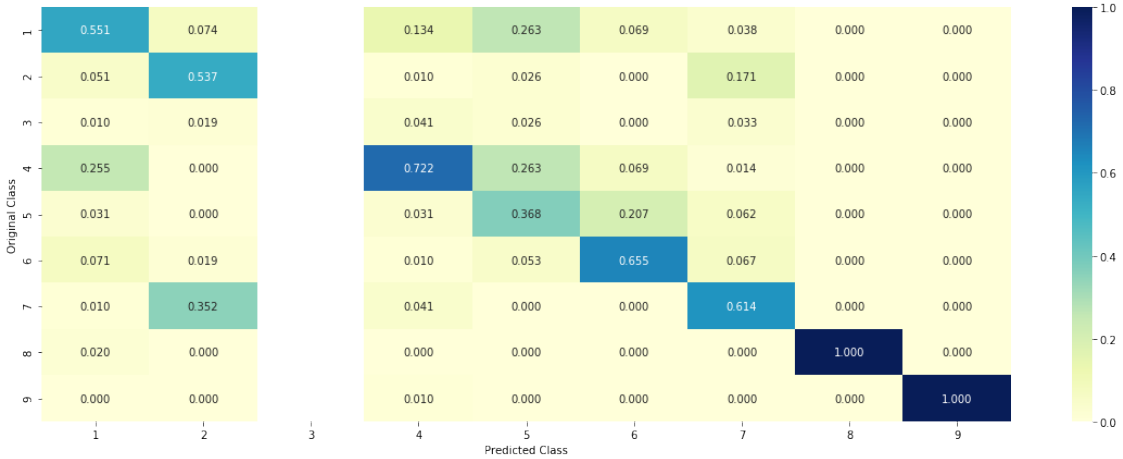
4.1.1.2. Testing the model with best hyper paramters

[57]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/
 ↪stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True,␣
 ↪class_prior=None)

# some of methods of MultinomialNB()
```

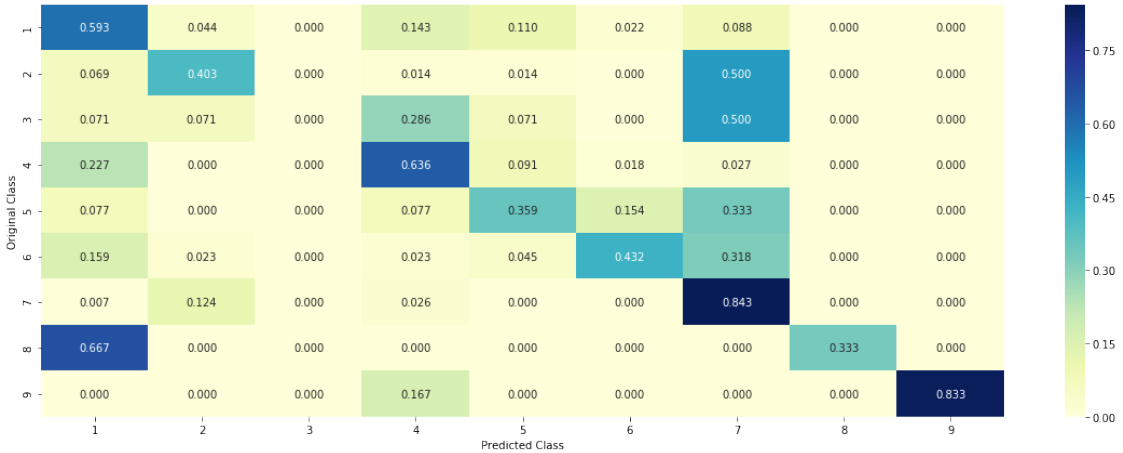
```
# fit(X, y[, sample_weight])        Fit Naive Bayes classifier according to X,␣
 ↪y
# predict(X)        Perform classification on an array of test vectors X.
# predict_log_proba(X)        Return log-probability estimates for the test␣
 ↪vector X.
# ------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/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


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

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

```
Log Loss : 1.1888342115586885
Number of missclassified point : 0.3966165413533835
--------------------- Confusion matrix --------------------
```

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



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



4.1.1.3. Feature Importance, Correctly classified point

[58]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0573 0.0432 0.0125 0.0647 0.0385 0.6926
0.0843 0.0036 0.0033]]
Actual Class : 1
--------------------------------------------------
5 Text feature [110] present in test data point [True]
13 Text feature [10] present in test data point [True]
14 Text feature [109] present in test data point [True]
24 Text feature [108table] present in test data point [True]
26 Text feature [13] present in test data point [True]
28 Text feature [00] present in test data point [True]
65 Text feature [006] present in test data point [True]
69 Text feature [005] present in test data point [True]
74 Text feature [129] present in test data point [True]
76 Text feature [107] present in test data point [True]
Out of the top  100  features  10 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

[59]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6928 0.0462 0.0132 0.0845 0.033  0.0334
0.0897 0.0038 0.0035]]
Actual Class : 1
--------------------------------------------------
32 Text feature [104] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.2. K Nearest Neighbour Classification

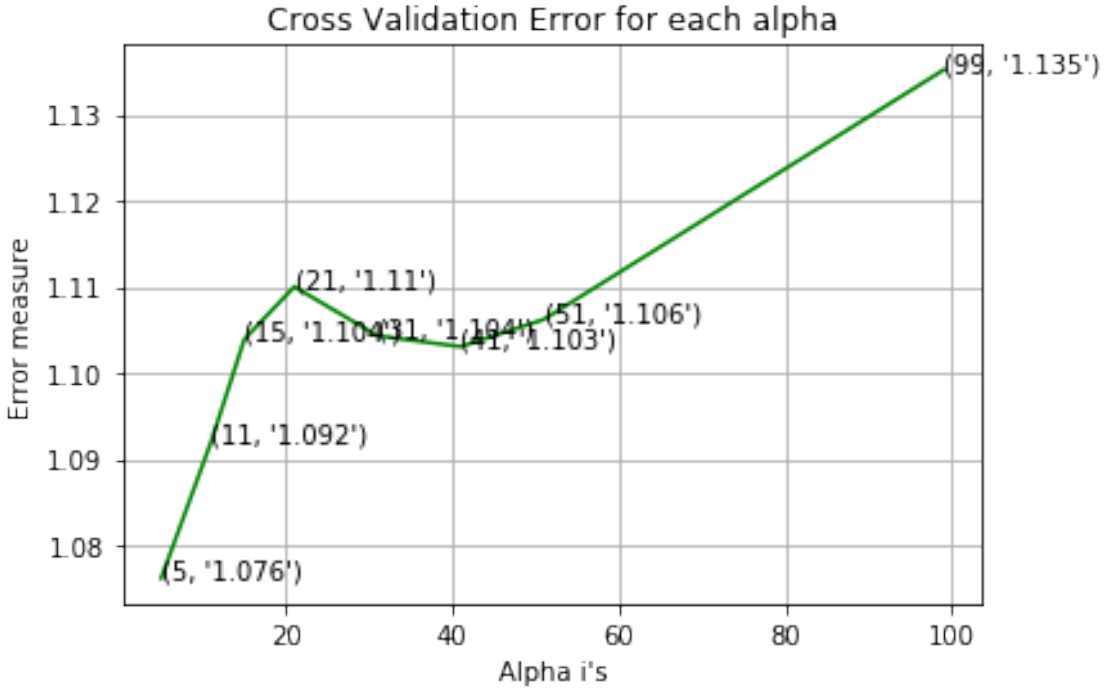
4.2.1. Hyper parameter tuning

[60]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto,␣
 ↪leaf_size=30, p=2,
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

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



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

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


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

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


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

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

```
for alpha = 5
Log Loss : 1.0761745087615866
```

```
for alpha = 11
Log Loss : 1.0920078443449122
for alpha = 15
Log Loss : 1.1039285065337823
for alpha = 21
Log Loss : 1.1100327087030168
for alpha = 31
Log Loss : 1.1043242832734752
for alpha = 41
Log Loss : 1.1030711429861957
for alpha = 51
Log Loss : 1.1062368720804299
for alpha = 99
Log Loss : 1.1351955806178928
```



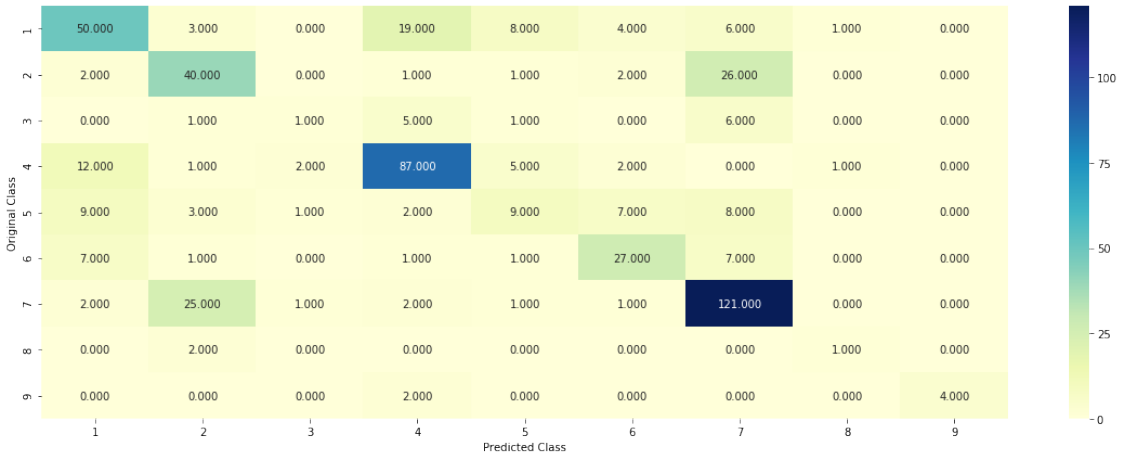

```
For values of best alpha =  5 The train log loss is: 0.4833138485026281
For values of best alpha =  5 The cross validation log loss is:
1.0761745087615866
For values of best alpha =  5 The test log loss is: 1.0325161149955246
```

4.2.2. Testing the model with best hyper paramters

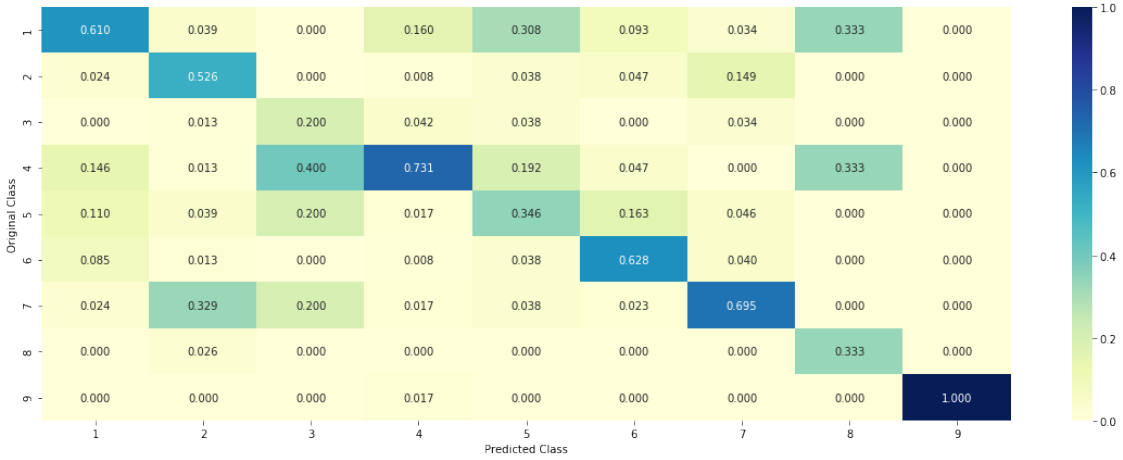
[61]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
```

```
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto,␣
 ↪leaf_size=30, p=2,
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

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

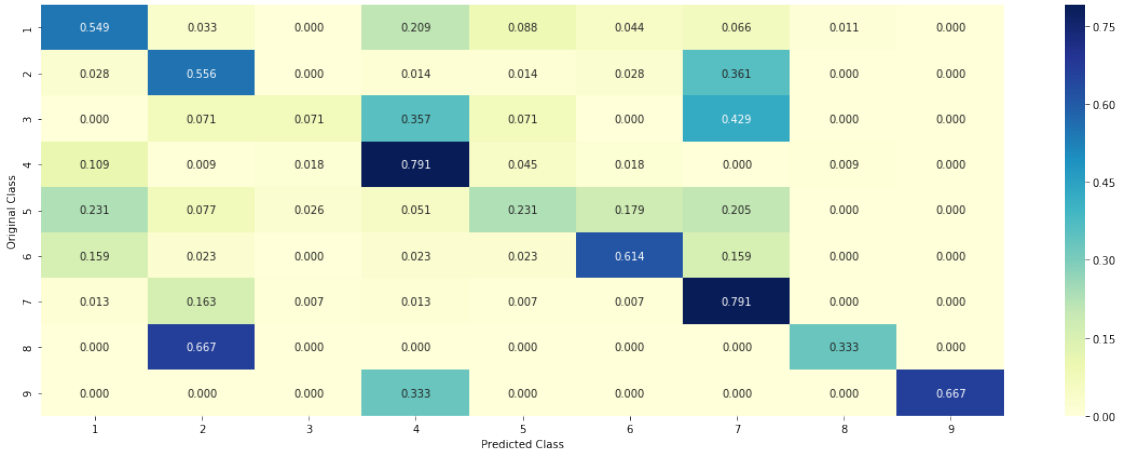
```
Log loss : 1.0761745087615866
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

[62]:
```
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 : 1
The  5  nearest neighbours of the test points belongs to classes [1 1 5 4 6]
Fequency of nearest points : Counter({1: 2, 5: 1, 4: 1, 6: 1})
```

4.2.4. Sample Query Point-2

[63]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

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

4.3. Logistic Regression

4.3.1. With Class balancing

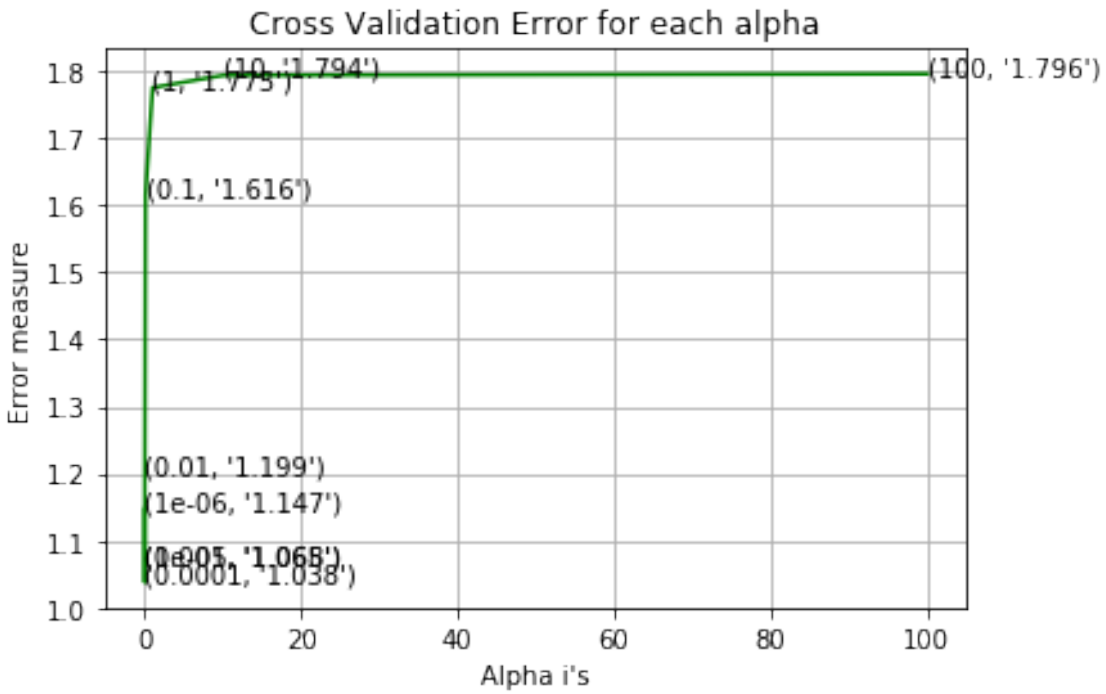
4.3.1.1. Hyper paramter tuning

[64]:
```
# 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 is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:
 ↪",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1471949632059448
for alpha = 1e-05
Log Loss : 1.0675174801651275
for alpha = 0.0001
Log Loss : 1.0378897357709995
for alpha = 0.001
Log Loss : 1.06468098835488
for alpha = 0.01
Log Loss : 1.199247781688393
for alpha = 0.1
Log Loss : 1.6156374138632983
for alpha = 1
Log Loss : 1.775292446303233
for alpha = 10
Log Loss : 1.7935756601481807
for alpha = 100
Log Loss : 1.7957311036705885
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4441910271872523
For values of best alpha =  0.0001 The cross validation log loss is:
1.0378897357709995
For values of best alpha =  0.0001 The test log loss is: 0.938551428494461
```

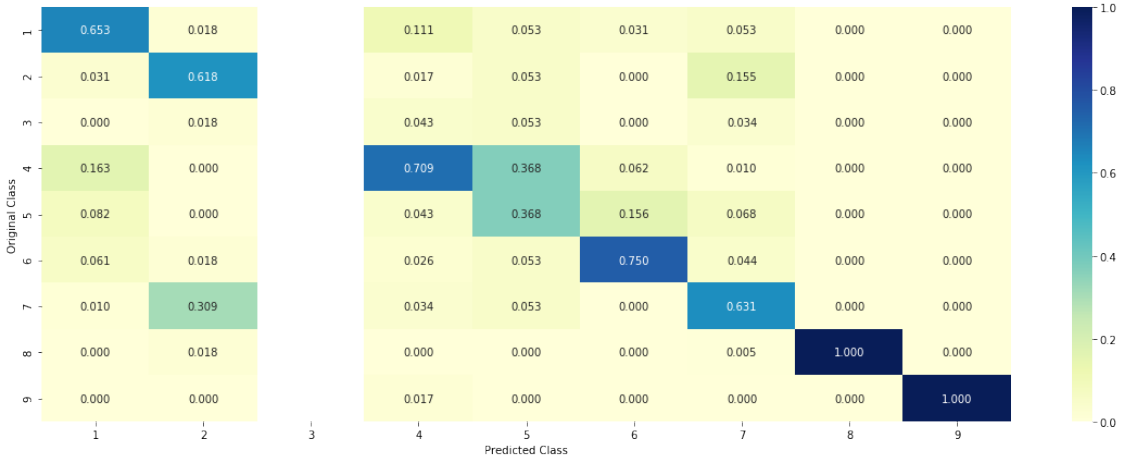
4.3.1.2. Testing the model with best hyper paramters

[65]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 1.0378897357709995
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```


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




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

4.3.1.3. Feature Importance

[66]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i],␣
 ↪yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query␣
 ↪point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0],"␣
 ↪class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or␣
 ↪Not']))
```

4.3.1.3.1. Correctly Classified point

[67]:
```
# 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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1792 0.0067 0.0028 0.0549 0.2286 0.5187
0.0041 0.0039 0.0008]]
Actual Class : 1
--------------------------------------------------
103 Text feature [01] present in test data point [True]
367 Text feature [117] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

[68]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.368e-01 6.300e-03 2.000e-04 4.370e-02
1.400e-03 3.700e-03 5.100e-03
  2.600e-03 3.000e-04]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

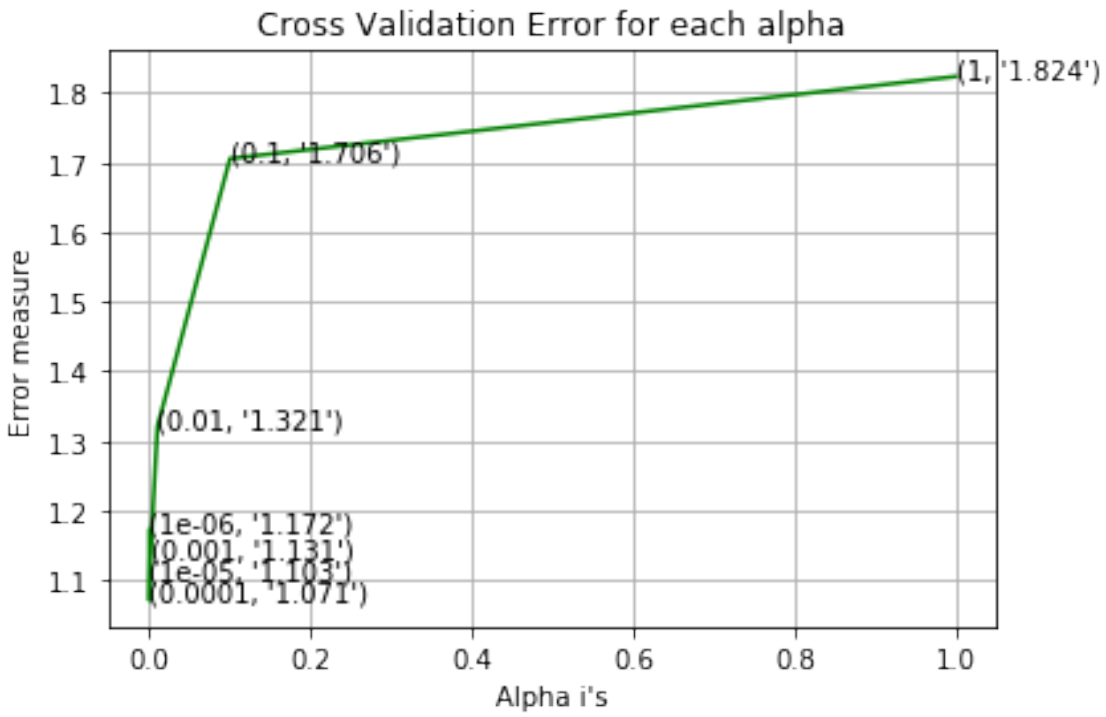
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

[69]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

```
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
 ↪classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1716604028738402
for alpha = 1e-05
Log Loss : 1.1034579518163985
for alpha = 0.0001
Log Loss : 1.0707417085389397
for alpha = 0.001
Log Loss : 1.130899449257992
for alpha = 0.01
Log Loss : 1.3210830519031889
for alpha = 0.1
Log Loss : 1.7058519897735307
for alpha = 1
Log Loss : 1.8237898147636011
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4319667790217524
For values of best alpha =  0.0001 The cross validation log loss is:
1.0707417085389397
For values of best alpha =  0.0001 The test log loss is: 0.9613490389731268
```

4.3.2.2. Testing model with best hyper parameters

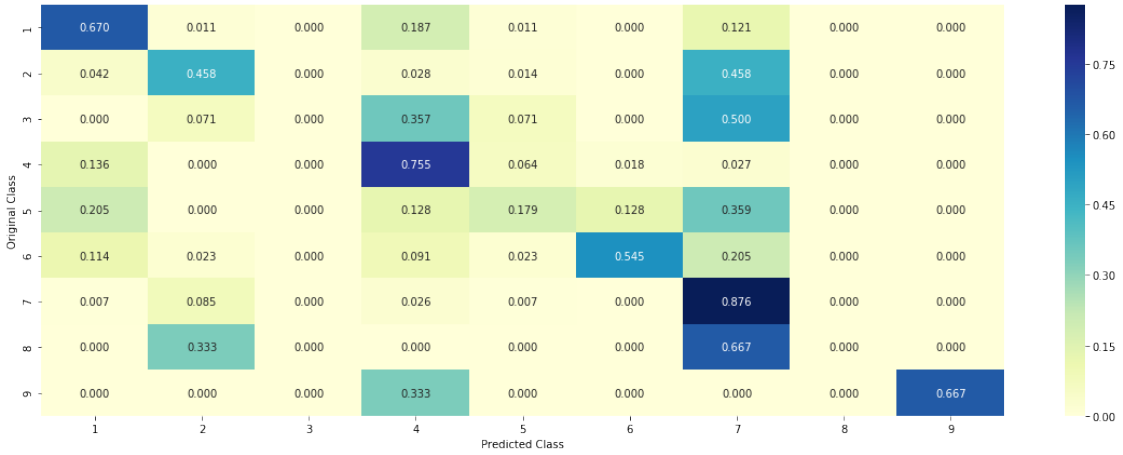
[70]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

```
Log loss : 1.0707417085389397
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

[71]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[1.934e-01 6.700e-03 2.300e-03 6.000e-02
2.056e-01 5.225e-01 6.000e-03
  3.200e-03 2.000e-04]]
Actual Class : 1
--------------------------------------------------
41 Text feature [03] present in test data point [True]
102 Text feature [01] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

[72]:
```
test_point_index = 100
no_feature = 500
```

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

```
Predicted Class : 1
Predicted Class Probabilities: [[9.352e-01 6.000e-03 3.000e-04 4.870e-02
1.300e-03 2.600e-03 4.300e-03
  1.600e-03 1.000e-04]]
Actual Class : 1
--------------------------------------------------
281 Text feature [10] present in test data point [True]
356 Text feature [030] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.4. Linear Support Vector Machines

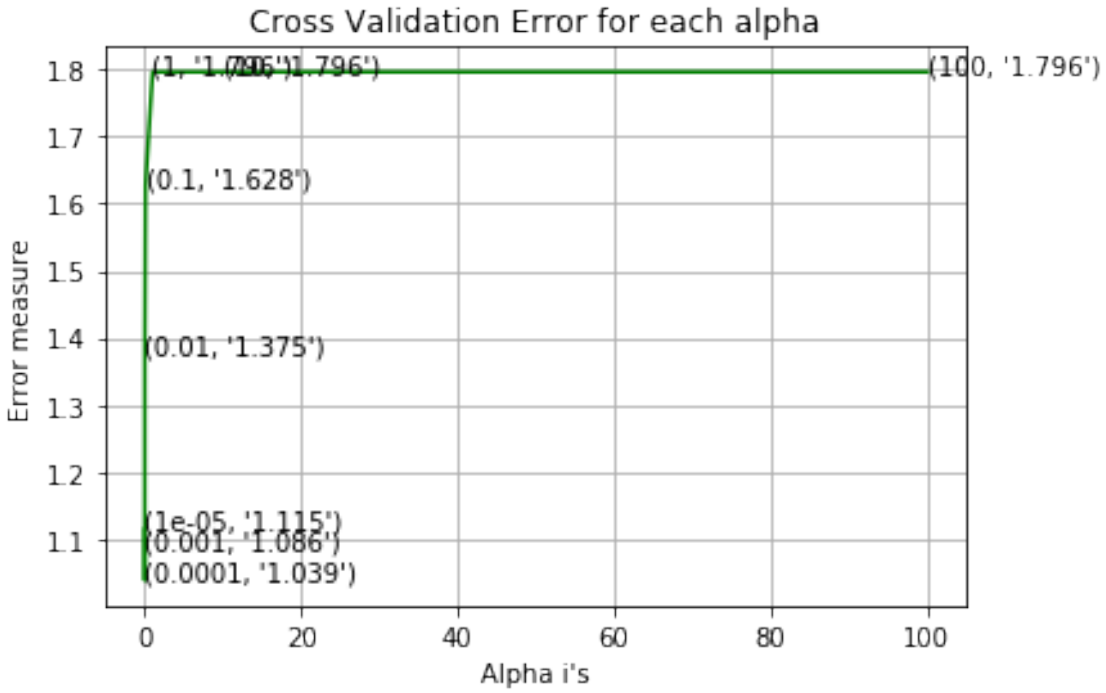
4.4.1. Hyper paramter tuning

[73]:
```
# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1,␣
 ↪decision_function_shape=ovr, random_state=None)

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



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

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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.1145843631890746
for C = 0.0001
Log Loss : 1.0385817887919162
for C = 0.001
Log Loss : 1.0855959429710396
for C = 0.01
Log Loss : 1.375086390621979
for C = 0.1
Log Loss : 1.6275442495249235
for C = 1
Log Loss : 1.7963710839492548
for C = 10
Log Loss : 1.7963710483930566
for C = 100
Log Loss : 1.7963712928524638
```



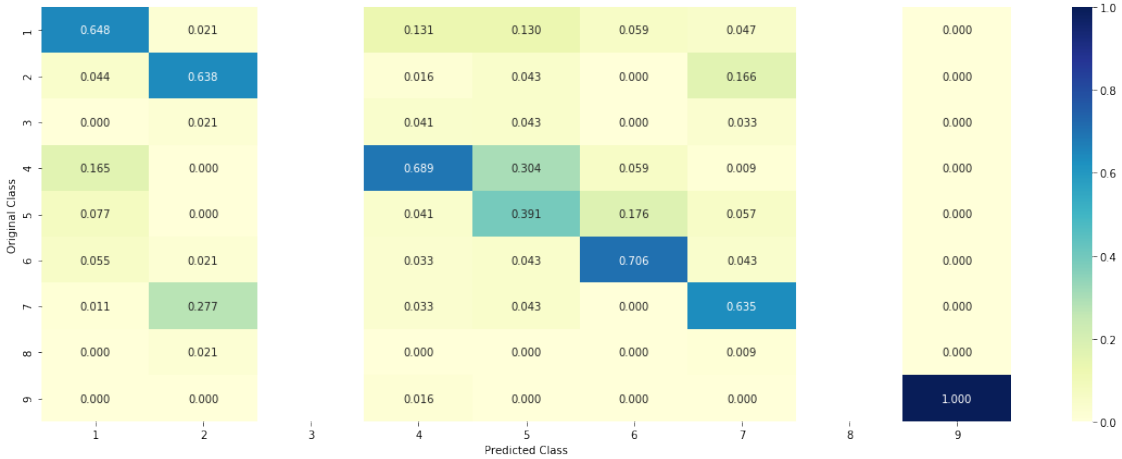
```
For values of best alpha =  0.0001 The train log loss is: 0.3886921784253098
For values of best alpha =  0.0001 The cross validation log loss is:
1.0385817887919162
For values of best alpha =  0.0001 The test log loss is: 0.9731460329849415
```

4.4.2. Testing model with best hyper parameters

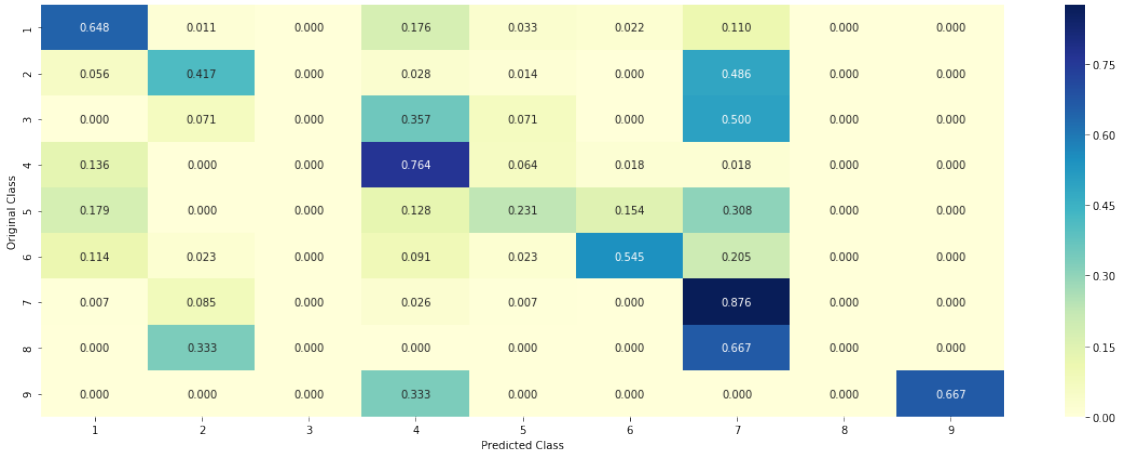
[74]:
```
# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ----------------------------------
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1,␣
 ↪decision_function_shape=ovr, random_state=None)

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


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

```
Log loss : 1.0385817887919162
Number of mis-classified points : 0.3533834586466165
-------------------- 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

[75]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.1939 0.0371 0.0041 0.0806 0.2995 0.3368
0.0443 0.0024 0.0012]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.3.2. For Incorrectly classified point

[76]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[8.554e-01 2.010e-02 5.000e-04 7.020e-02
3.300e-03 3.130e-02 1.540e-02
```

```
  2.400e-03 1.600e-03]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

[77]:
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

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

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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2480564658535527
for n_estimators = 100 and max depth =  10
Log Loss : 1.272333358559436
for n_estimators = 200 and max depth =  5
Log Loss : 1.238040743803511
for n_estimators = 200 and max depth =  10
Log Loss : 1.2604355061689345
for n_estimators = 500 and max depth =  5
Log Loss : 1.2405130454720714
for n_estimators = 500 and max depth =  10
Log Loss : 1.2544311934181707
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2376144342745536
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2531933414827798
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2380894801934184
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2524279581779028
For values of best estimator =  1000 The train log loss is: 0.8632155642056464
For values of best estimator =  1000 The cross validation log loss is:
1.2376144342745534
For values of best estimator =  1000 The test log loss is: 1.1615366126526514
```

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

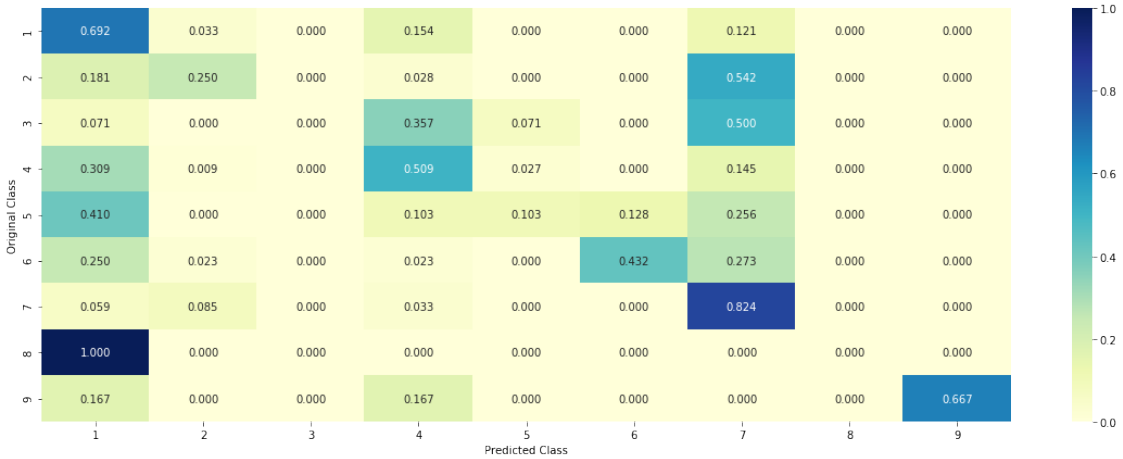
[78]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
```

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

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

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

```
Log loss : 1.2376144342745534
Number of mis-classified points : 0.4548872180451128
-------------------- Confusion matrix --------------------
```


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

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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

[79]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
 ↪n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
```

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2162 0.0053 0.0121 0.0569 0.1629 0.5305
0.0114 0.0023 0.0024]]
Actual Class : 1
--------------------------------------------------
4 Text feature [113705] present in test data point [True]
28 Text feature [12g] present in test data point [True]
59 Text feature [13] present in test data point [True]
79 Text feature [046] present in test data point [True]
99 Text feature [110] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

4.5.3.2. Inorrectly Classified point

[80]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3162 0.1058 0.0193 0.2255 0.0561 0.0609
0.1828 0.0064 0.027 ]]
Actuall Class : 1
--------------------------------------------------
22 Text feature [130] present in test data point [True]
59 Text feature [13] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

[81]:
```
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
```

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log␣
 ↪loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross␣
 ↪validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_,␣
 ↪eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log␣
 ↪loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.146129490159194
for n_estimators = 10 and max depth =  3
```

```
Log Loss : 1.803812607660948
for n_estimators = 10 and max depth =  5
Log Loss : 1.6429917143154265
for n_estimators = 10 and max depth =  10
Log Loss : 1.8803936152829723
for n_estimators = 50 and max depth =  2
Log Loss : 1.7704250600787985
for n_estimators = 50 and max depth =  3
Log Loss : 1.5024760375386246
for n_estimators = 50 and max depth =  5
Log Loss : 1.4562840391238305
for n_estimators = 50 and max depth =  10
Log Loss : 1.6802610610171416
for n_estimators = 100 and max depth =  2
Log Loss : 1.6439589675833544
for n_estimators = 100 and max depth =  3
Log Loss : 1.55250358372621
for n_estimators = 100 and max depth =  5
Log Loss : 1.4099488597947194
for n_estimators = 100 and max depth =  10
Log Loss : 1.7040322847392917
for n_estimators = 200 and max depth =  2
Log Loss : 1.674997674151384
for n_estimators = 200 and max depth =  3
Log Loss : 1.5416183040624987
for n_estimators = 200 and max depth =  5
Log Loss : 1.4578111075245568
for n_estimators = 200 and max depth =  10
Log Loss : 1.72491113253021
for n_estimators = 500 and max depth =  2
Log Loss : 1.7385169196570485
for n_estimators = 500 and max depth =  3
Log Loss : 1.6002134959970564
for n_estimators = 500 and max depth =  5
Log Loss : 1.455637047571353
for n_estimators = 500 and max depth =  10
Log Loss : 1.678098806548235
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7092013584655628
for n_estimators = 1000 and max depth =  3
Log Loss : 1.609193874977989
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4400947361785064
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6847759674419465
For values of best alpha =  100 The train log loss is: 0.05023498425226835
For values of best alpha =  100 The cross validation log loss is:
1.4099488597947192
```

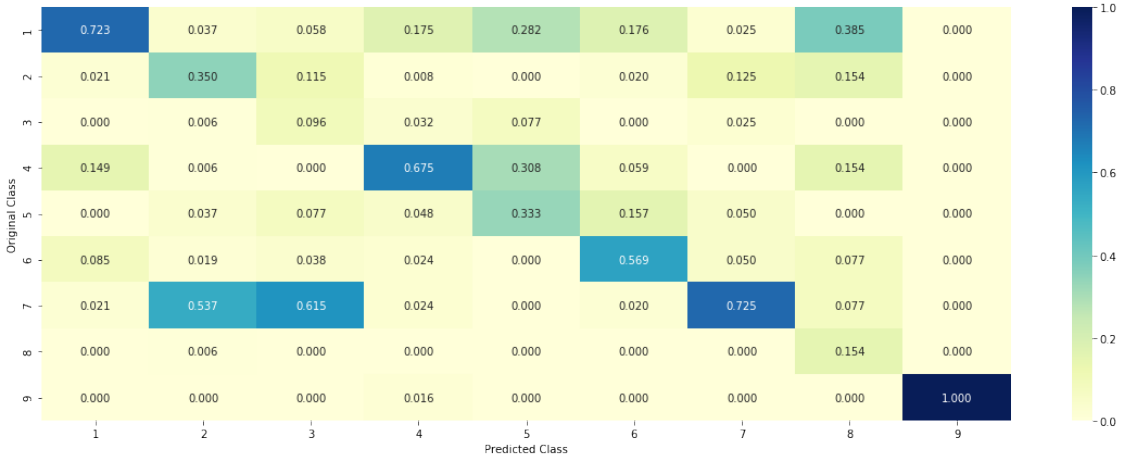
```
For values of best alpha =  100 The test log loss is: 1.33914525066316
```

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

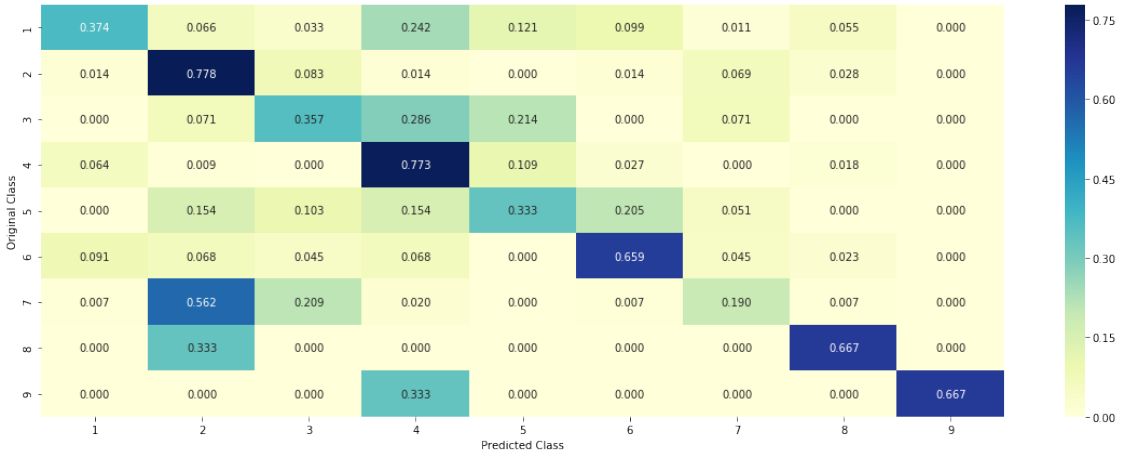
[82]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini,␣
 ↪max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto,␣
 ↪max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,␣
 ↪random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.4099488597947192
Number of mis-classified points : 0.5169172932330827
-------------------- Confusion matrix --------------------
```

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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

[83]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)],␣
↪criterion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,␣
↪n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0355 0.0042 0.0275 0.0187 0.3526 0.5486
0.0027 0.0048 0.0054]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
```

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

4.5.5.2. Incorrectly Classified point

[84]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].
 ↪reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8736 0.0076 0.0056 0.0859 0.0018 0.0125
0.0028 0.0068 0.0033]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

[85]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

```
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1,␣
 ↪decision_function_shape=ovr, random_state=None)

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


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

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

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

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


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log',␣
 ↪class_weight='balanced', random_state=0)
```

```
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

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

```
Logistic Regression :  Log Loss: 1.06
Support vector machines : Log Loss: 1.80
Naive Bayes : Log Loss: 1.19
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.030
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.500
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.182
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.400
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.883
```

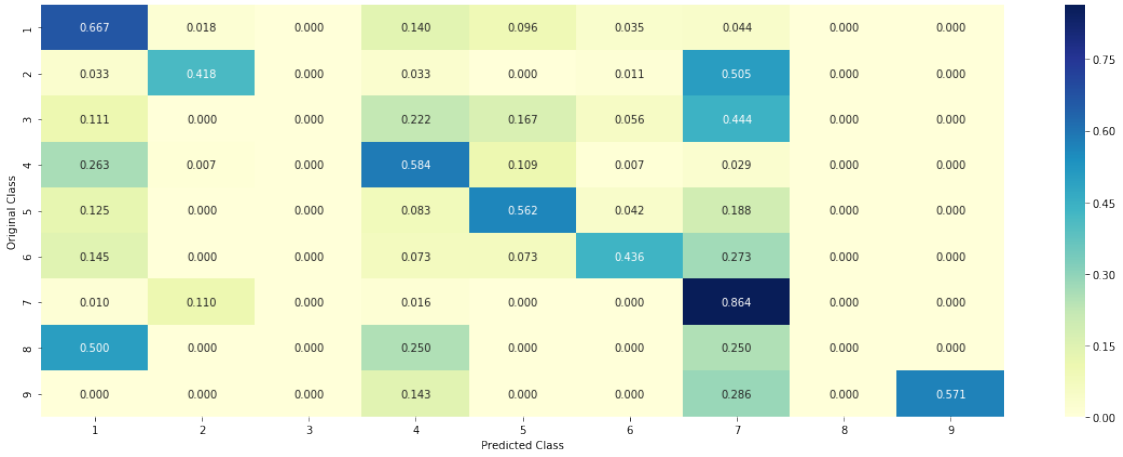
4.7.2 testing the model with the best hyper parameters

[86]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3],␣
 ↪meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5355291349061079
Log loss (CV) on the stacking classifier : 1.182205524238464
Log loss (test) on the stacking classifier : 1.1346428320352178
Number of missclassified point : 0.3774436090225564
-------------------- Confusion matrix --------------------
```


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


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



4.7.3 Maximum Voting classifier

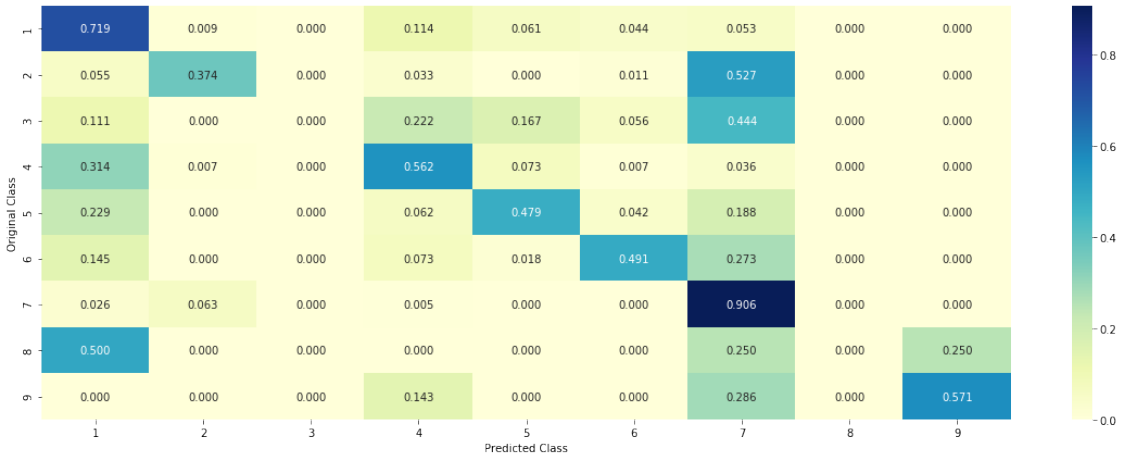
[87]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf',␣
 ↪sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.
 ↪predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.
 ↪predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.
 ↪predict_proba(test_x_onehotCoding)))
```

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

```
Log loss (train) on the VotingClassifier : 0.8312125256444053
Log loss (CV) on the VotingClassifier : 1.194563953789798
Log loss (test) on the VotingClassifier : 1.1754945381388553
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```


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


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



5. Assignments

Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)

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

Apply Logistic regression with CountVectorizer Features, including both unigrams and bi-grams

Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

[88]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer = CountVectorizer(ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.
 ↪fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of␣
 ↪times it 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 : 2334129
```

[89]:
```
# merging gene, variance and text features

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

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

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

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

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


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

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

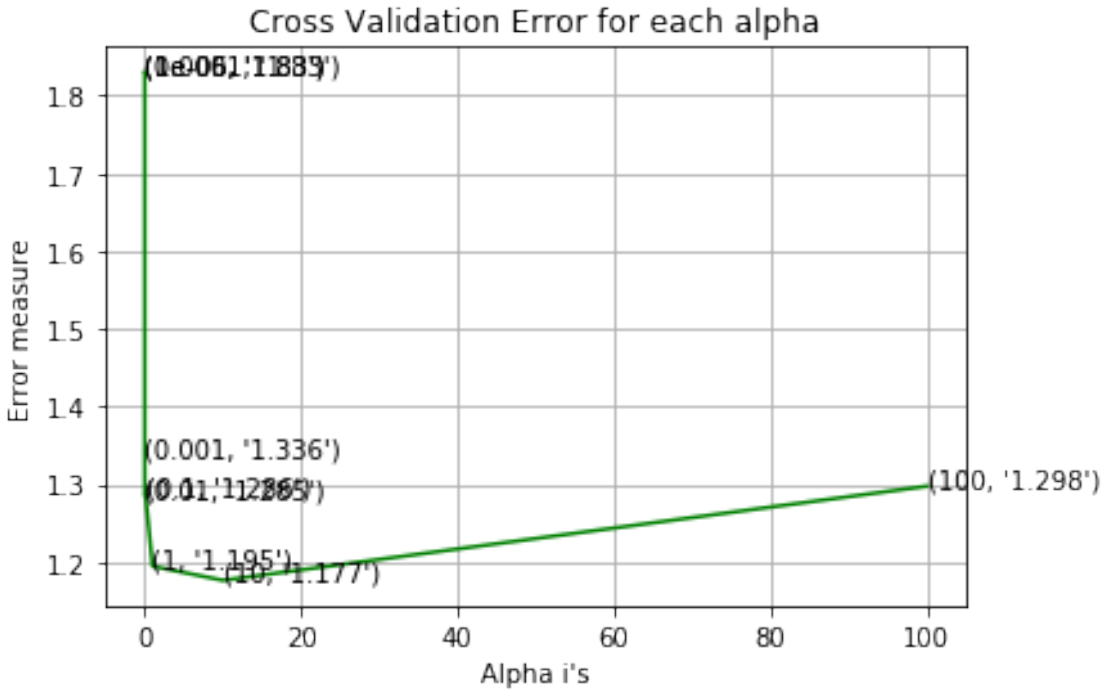
[90]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
```

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

```
for alpha = 1e-06
Log Loss : 1.8304997567764278
for alpha = 1e-05
Log Loss : 1.8304997567764278
for alpha = 0.0001
Log Loss : 1.8304997567764278
for alpha = 0.001
Log Loss : 1.3359570039593847
for alpha = 0.01
Log Loss : 1.2848582091876044
for alpha = 0.1
Log Loss : 1.2862631896832672
for alpha = 1
Log Loss : 1.194837683045458
for alpha = 10
```

```
Log Loss : 1.1766899297094717
for alpha = 100
Log Loss : 1.2977847896761399
```




```
For values of best alpha =  10 The train log loss is: 0.8747789177536139
For values of best alpha =  10 The cross validation log loss is:
1.1766899297094717
For values of best alpha =  10 The test log loss is: 1.1086758277755755
```

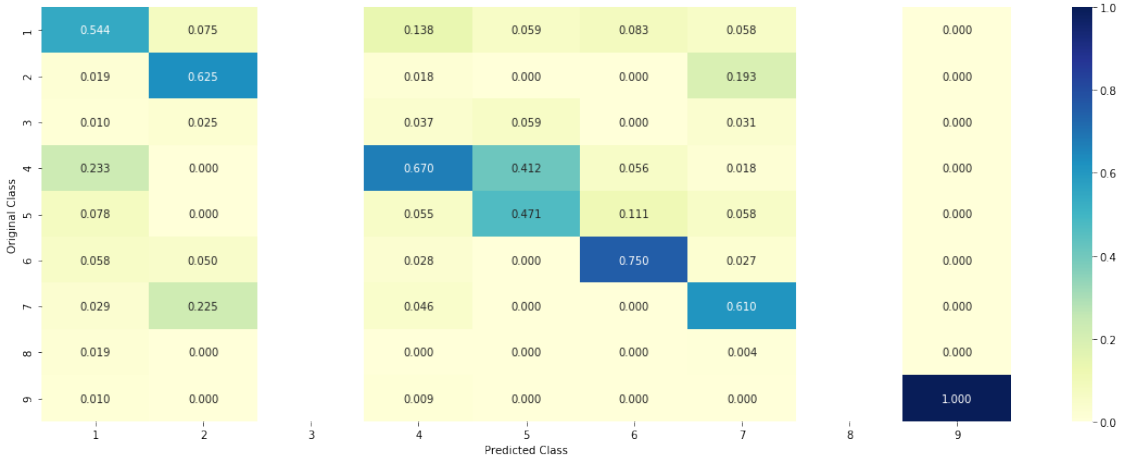
4.3.1.2. Testing the model with best hyper paramters

[91]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

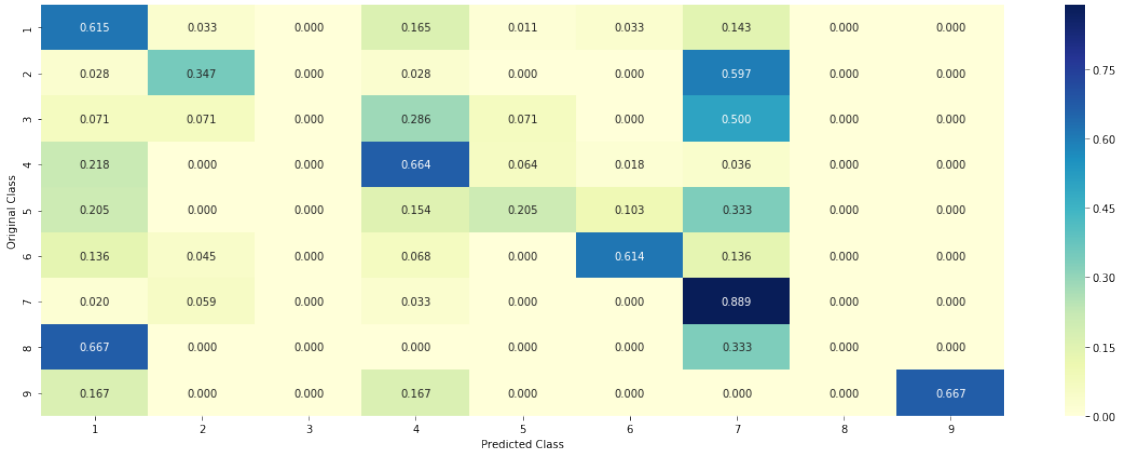
```
#------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],␣
 ↪penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding,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

```
Log loss : 1.1766899297094717
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```


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



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



4.3.1.3. Feature Importance

[92]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i],␣
 ↪yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query␣
 ↪point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0],"␣
 ↪class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or␣
 ↪Not']))
```

4.3.1.3.1. Correctly Classified point

[95]:
```
# 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(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
#get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.278  0.0046 0.0105 0.0734 0.3725 0.2458
0.0035 0.0108 0.0008]]
Actual Class : 1
--------------------------------------------------
```

4.3.1.3.2. Incorrectly Classified point

[96]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
#get_impfeature_names(indices[0], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1445 0.1469 0.0096 0.3187 0.034  0.0446
0.2883 0.0062 0.0073]]
Actual Class : 1
--------------------------------------------------
```

featureengineering

[97]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer = TfidfVectorizer(max_features=3000,ngram_range=(1,4))
```

```
train_text_feature_onehotCoding = text_vectorizer.
 ↪fit_transform(train_df['TEXT'])
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of␣
 ↪times it 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 : 3000
```

[98]:
```
# merging gene, variance and text features

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

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

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

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

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

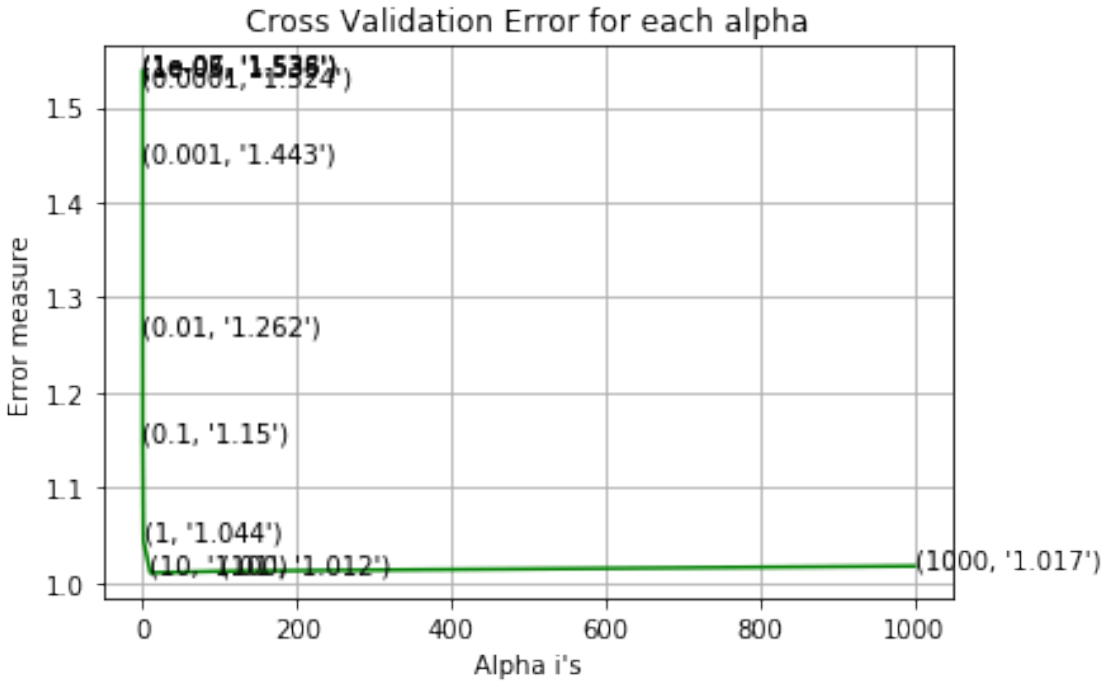
```
cv_y = np.array(list(cv_df['Class']))


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

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

[109]:
```
# 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(-7, 4)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
    clf = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    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()
```

```
for C = 1e-07
Log Loss : 1.538273332866102
for C = 1e-06
Log Loss : 1.5358760071855444
for C = 1e-05
Log Loss : 1.5347580530591214
for C = 0.0001
Log Loss : 1.5240151143298053
for C = 0.001
Log Loss : 1.4429766172901641
for C = 0.01
Log Loss : 1.261965741630454
for C = 0.1
Log Loss : 1.150244784455939
```

```
for C = 1
Log Loss : 1.0444299859998958
for C = 10
Log Loss : 1.0103288075424641
for C = 100
Log Loss : 1.0121231707628517
for C = 1000
Log Loss : 1.0174833939830819
```

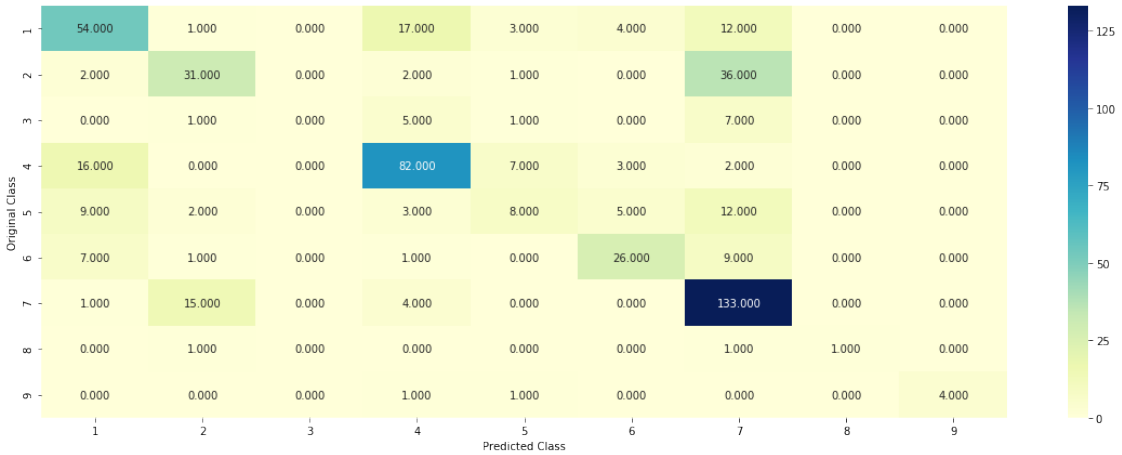


4.3.1.2. Testing the model with best hyper paramters

[112]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,␣
 ↪learning_rate=optimal, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

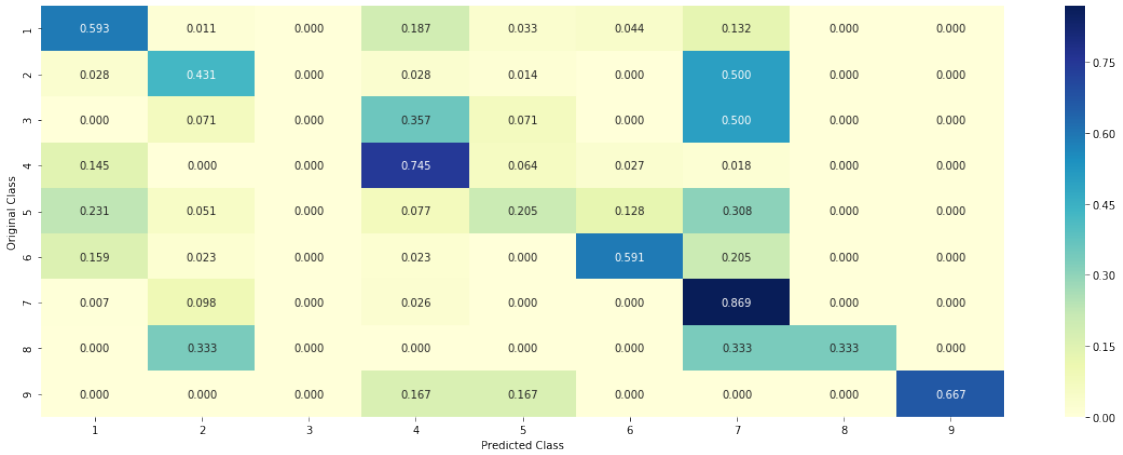
```
#------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/geometric-intuition-1/
#------------------------------


best_alpha = np.argmin(cv_log_error_array)

clf =␣
 ↪LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,␣
 ↪cv_x_onehotCoding, cv_y, clf)
```

```
For values of best alpha =  10 The train log loss is: 0.4469668457827815
For values of best alpha =  10 The cross validation log loss is:
1.0103288075424641
For values of best alpha =  10 The test log loss is: 0.9426097821260784
Log loss : 1.0103288075424641
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```



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


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



[113]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Train logloss", "Test logloss"]
x.add_row(["TFIDF",  round(0.9679107505070993,3), round(1.27,3)])
x.add_row(["Bag of words with2gram",  round( 0.834545887877024,3), round(1.
↪2320152426040365,3)])
x.add_row(["TFIDF 1-4 gram maxfeature 3000",  round(0.446966845782,3), round(0.
↪9426097821260784,3)])
```

```
x.border=True
print(x)
```

```
+--------------------------------+---------------+--------------+
|             Model              | Train logloss | Test logloss |
+--------------------------------+---------------+--------------+
|             TFIDF              |     0.968     |     1.27     |
|     Bag of words with2gram     |     0.835     |    1.232     |
| TFIDF 1-4 gram maxfeature 3000 |     0.447     |    0.943     |
+--------------------------------+---------------+--------------+
```

TFIDF model withh (1-4) gram and maxfeature 3000 is the best model with test logloss 0.934