mayankgupta9968@gmail.com_15

November 22, 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.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
```

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

```
Using TensorFlow backend.
```

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 : 255.499943 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])
```

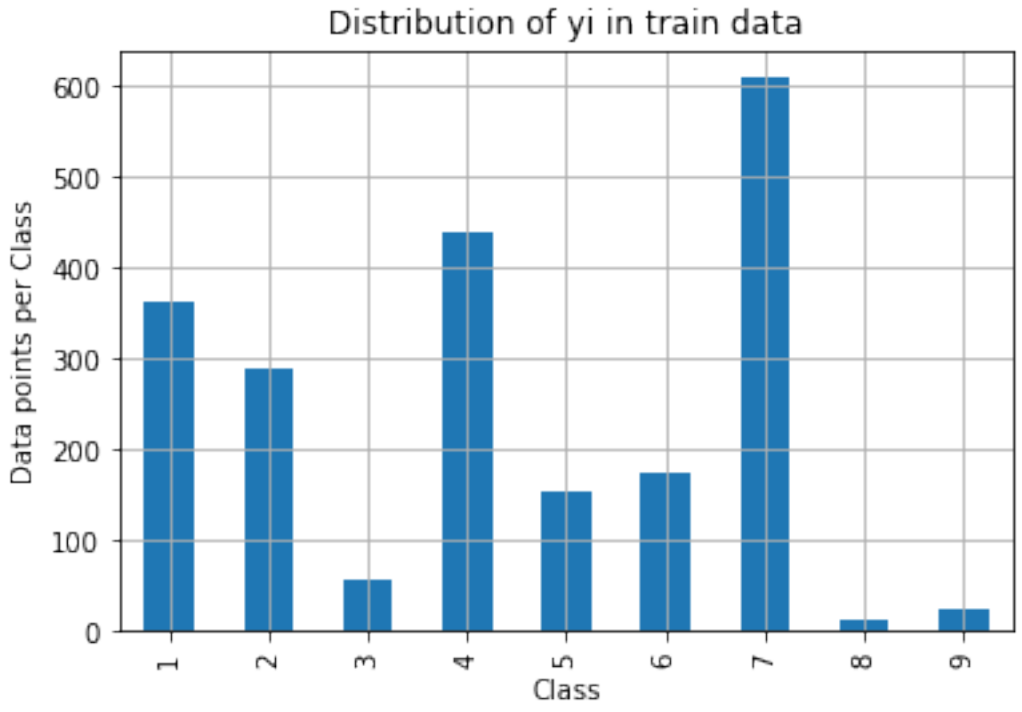
```
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

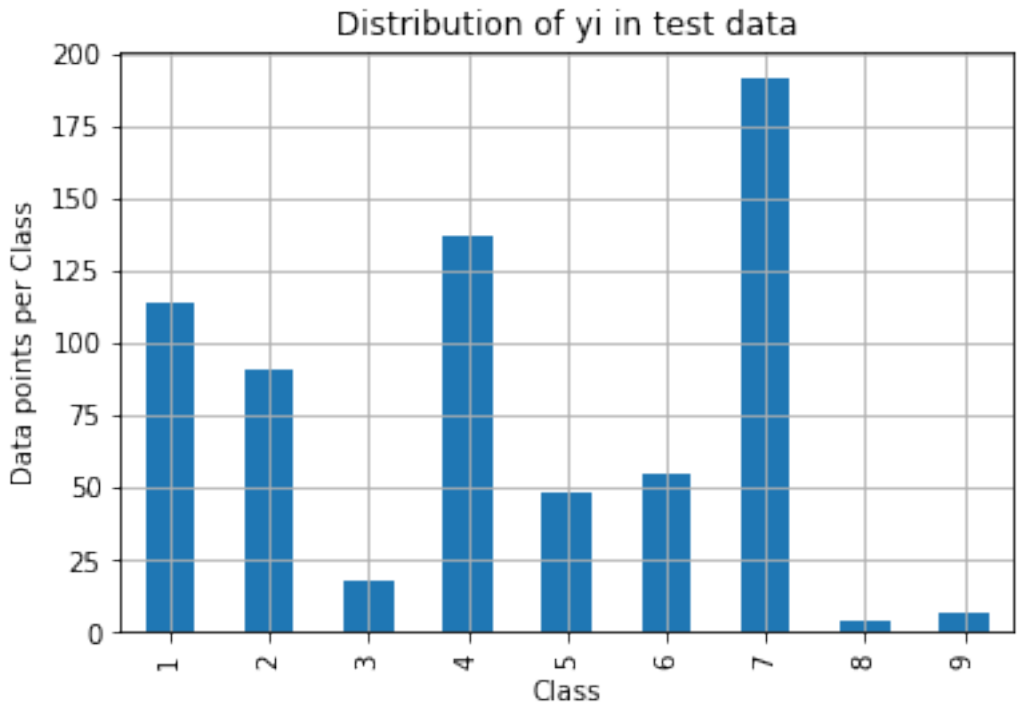
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

[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().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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

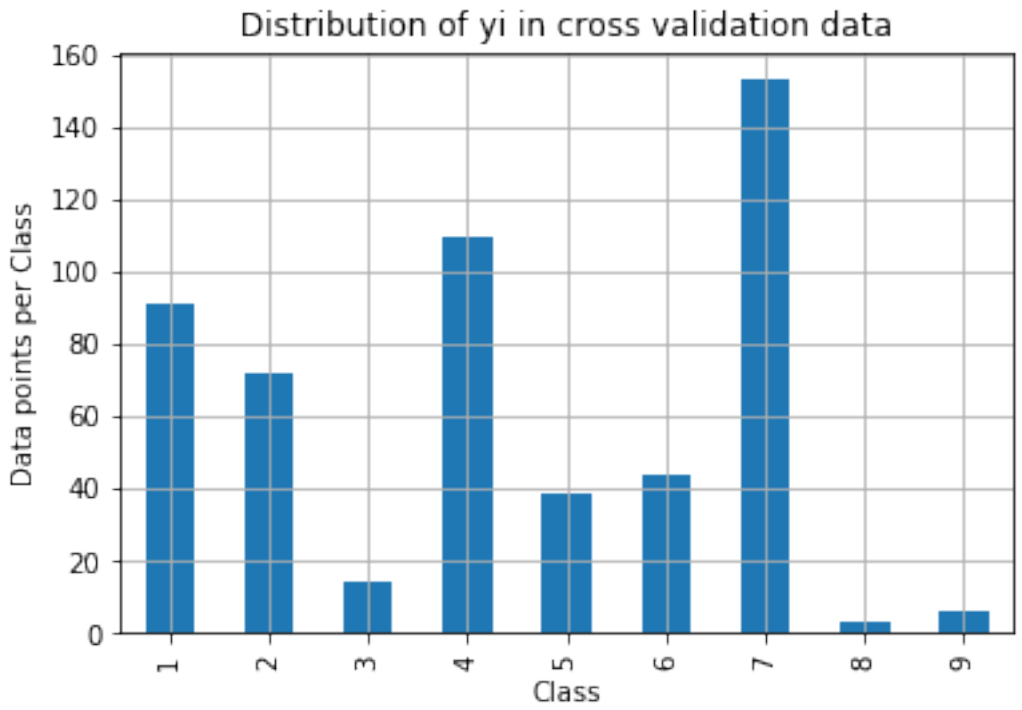
```
    print('Number of data points in class', i+1, ':',test_class_distribution.
 ↪values[i], '(', np.round((test_class_distribution.values[i]/test_df.
 ↪shape[0]*100), 3), '%)')

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




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




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

3.2 Prediction using a 'Random' Model

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

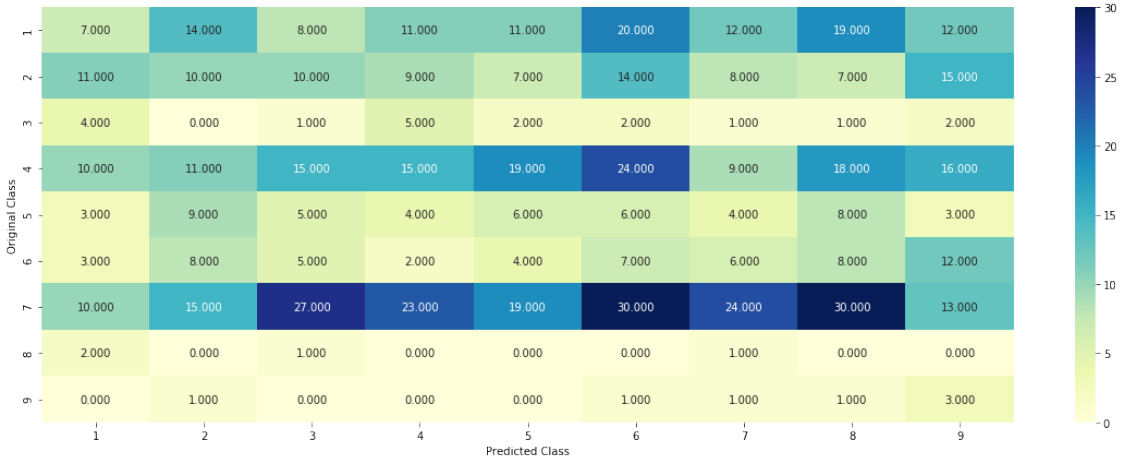
[13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i␣
 ↪are predicted class j

    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()
```

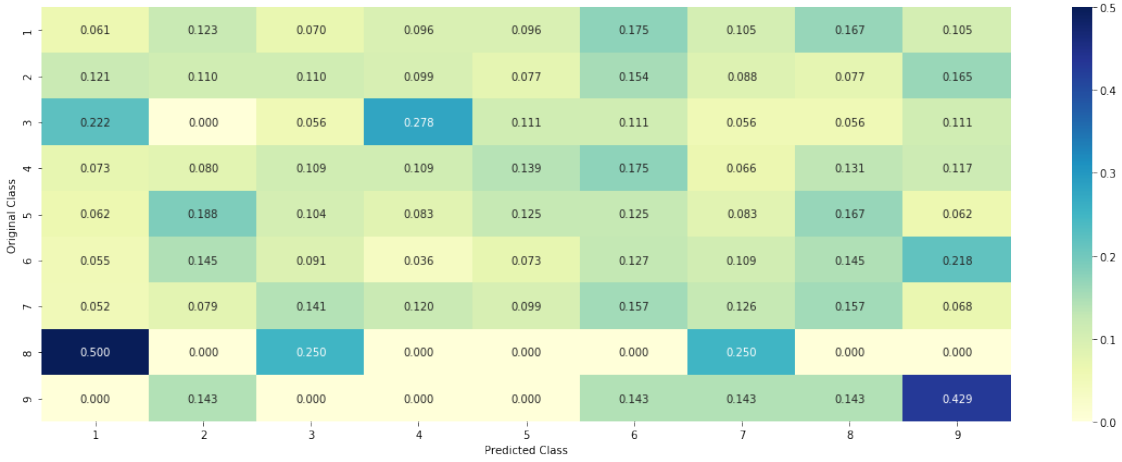
[14]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their␣
 ↪sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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 2.456463751152216
Log loss on Test Data using Random Model 2.5023429562818578
-------------------- 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, 0.34659090909090912, 0.
 ↪0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.
 ↪060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.
 ↪3454545454545454, 0.060606060606060608, 0.060606060606060608, 0.
 ↪060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.
 ↪069182389937106917, 0.46540880503144655, 0.07547169811320754, 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))
```

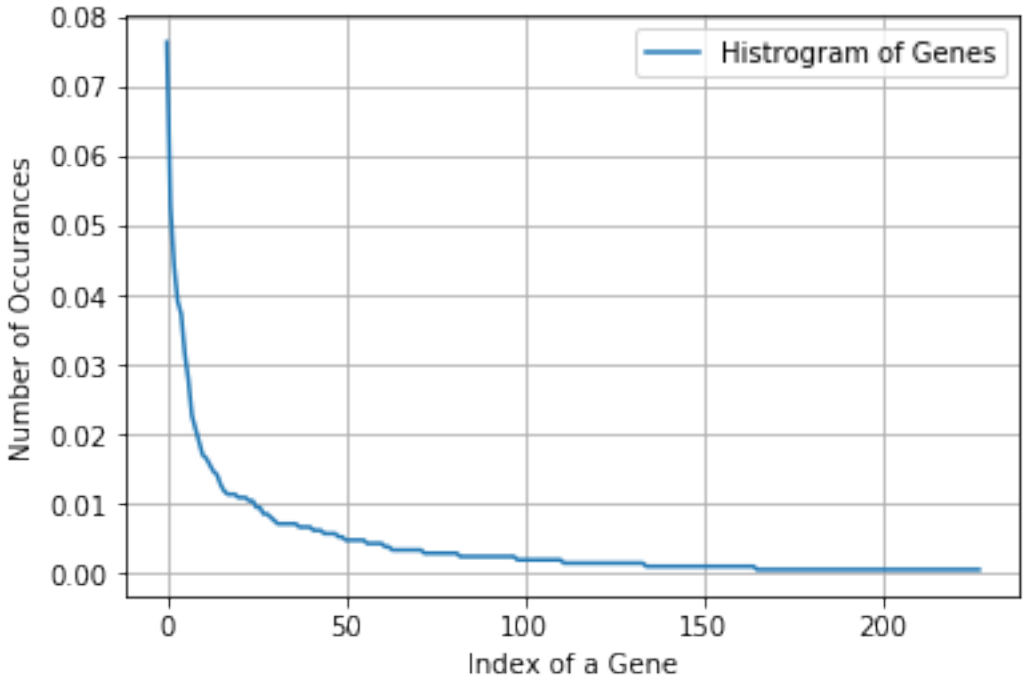
```
Number of Unique Genes : 228
BRCA1     162
TP53      112
EGFR       94
BRCA2      83
PTEN       79
KIT        67
BRAF       59
ALK        48
ERBB2      44
PDGFRA     40
Name: Gene, dtype: int64
```

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

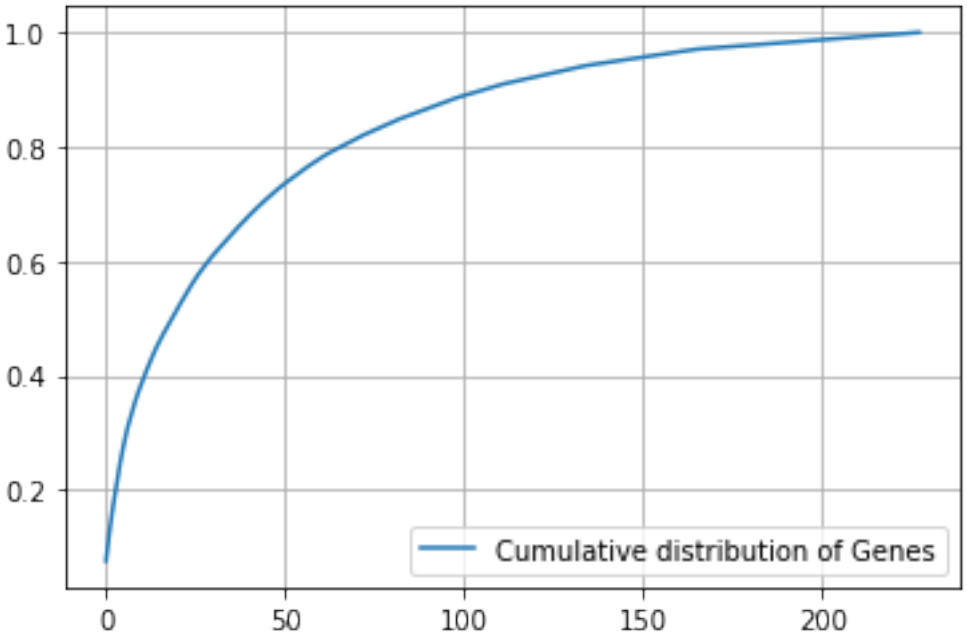
```
Ans: There are 228 different categories of genes in the train data, and they are
distibuted as follows
```

[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 = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.
 ↪fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

[23]:
```
train_df['Gene'].shape
```

[23]:
```
(2124,)
```

[24]:
```
train_df['Gene'].head()
```

[24]:
```
1432     SPOP
2741     BRAF
2882    BRCA2
1857     MTOR
2615    BRCA1
Name: Gene, dtype: object
```

[25]:
```
gene_vectorizer.get_feature_names()
```

[25]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
```

```
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd2',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fat1',
```

```
'fbxw7',
'fgf19',
'fgf3',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
```

```
'mdm4',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
```

```
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'ros1',
'rras2',
'runx1',
'rxra',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'stat3',
'stk11',
'tcf3',
'tert',
'tet1',
'tet2',
'tgfbr1',
```

```
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xrcc2',
 'yap1']
```

[26]:
```
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, 228)
```

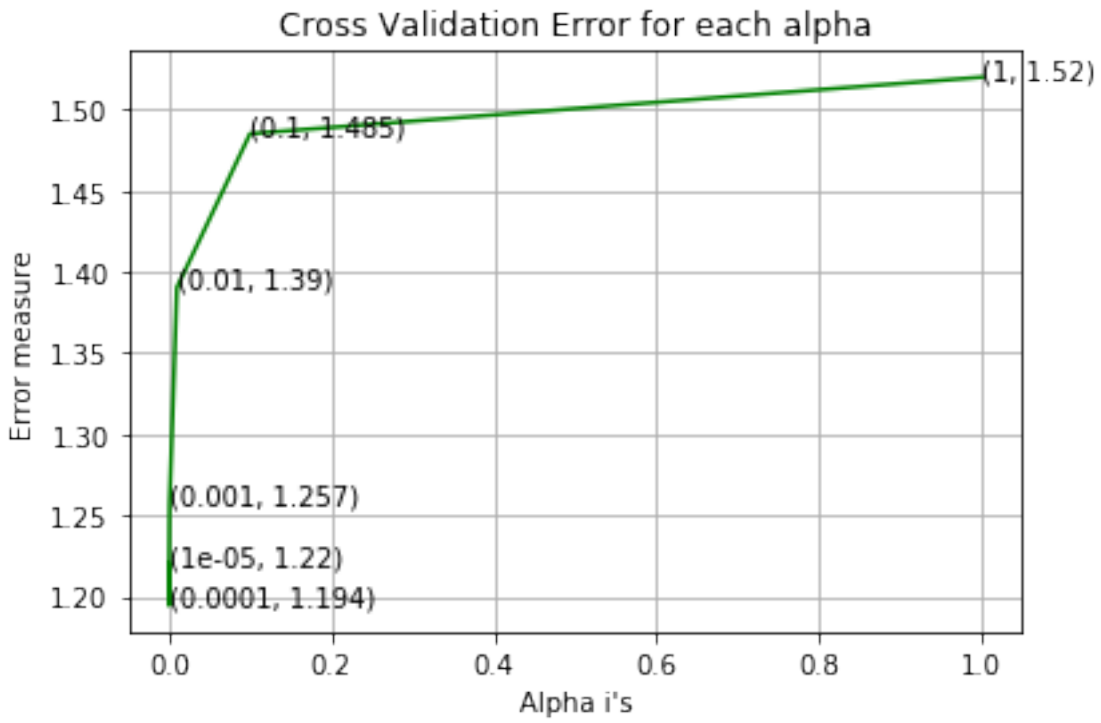
Q4. How good is this gene feature in predicting y_i?

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

[27]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_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: 1.2198067393549723
For values of alpha =  0.0001 The log loss is: 1.1944367683896815
For values of alpha =  0.001 The log loss is: 1.2566449192361553
For values of alpha =  0.01 The log loss is: 1.390321850550204
For values of alpha =  0.1 The log loss is: 1.4851427384784721
For values of alpha =  1 The log loss is: 1.5199777597538766
```




```
For values of best alpha =  0.0001 The train log loss is: 1.014596694213375
For values of best alpha =  0.0001 The cross validation log loss is:
1.1944367683896815
For values of best alpha =  0.0001 The test log loss is: 1.2051422016738564
```

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.

[28]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",␣
 ↪unique_genes.shape[0], " genes in train dataset?")

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":
 ↪",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"␣
 ↪,(cv_coverage/cv_df.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 645 out of 665 : 96.99248120300751
2. In cross validation data 513 out of  532 : 96.42857142857143
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

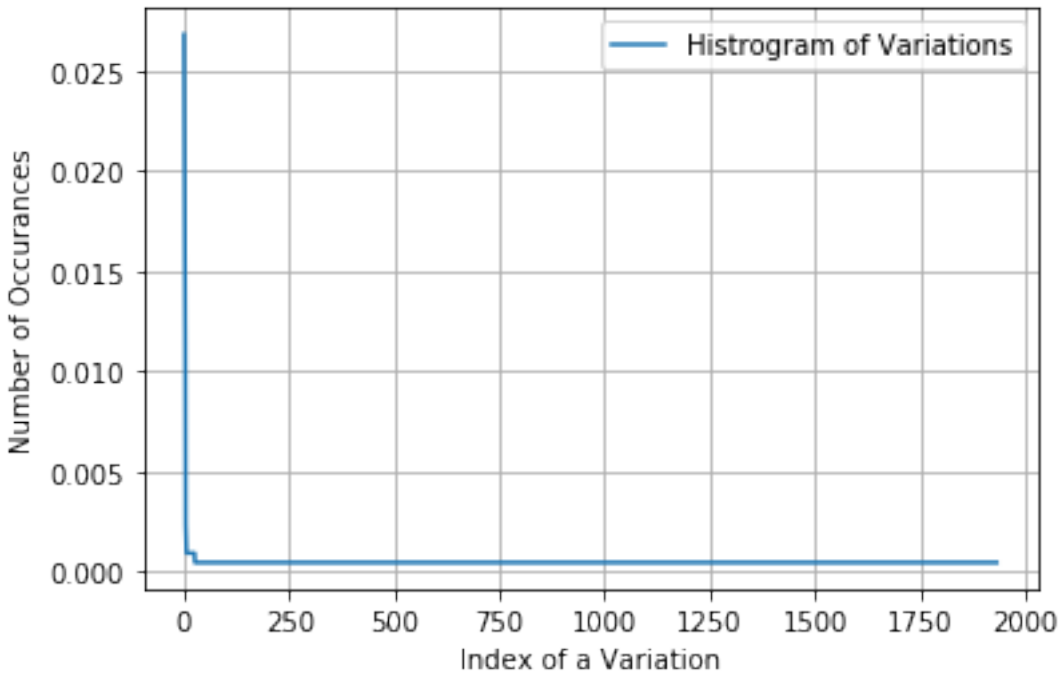
[29]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1933
Truncating_Mutations    57
Amplification           48
Deletion                40
Fusions                 24
Overexpression           5
E17K                     3
R841K                    2
F28L                     2
EWSR1-ETV1_Fusion        2
G13C                     2
Name: Variation, dtype: int64
```

[30]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of␣
 ↪variations in the train data, and they are distibuted as follows",)
```

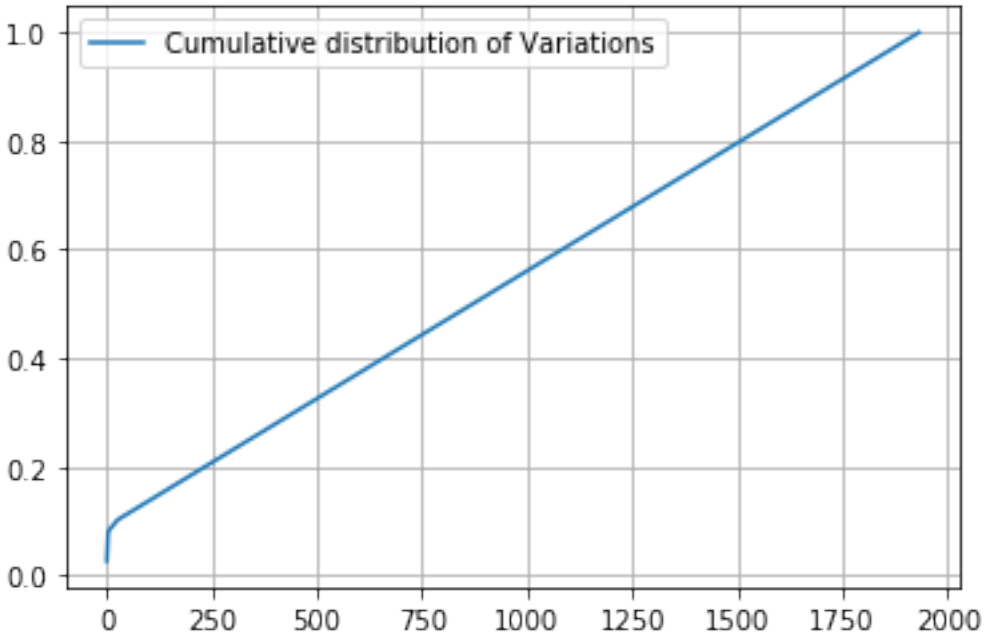
```
Ans: There are 1933 different categories of variations in the train data, and
they are distibuted as follows
```

[31]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



[32]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02683616 0.04943503 0.06826742 ... 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

[33]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha,␣
 ↪"Variation", cv_df))
```

[34]:
```
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)
```

[35]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.
 ↪fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.
 ↪transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.
 ↪transform(cv_df['Variation'])
```

[36]:
```
print("train_variation_feature_onehotEncoded is converted feature using the␣
 ↪onne-hot encoding method. The shape of Variation feature:",␣
 ↪train_variation_feature_onehotCoding.shape)
```

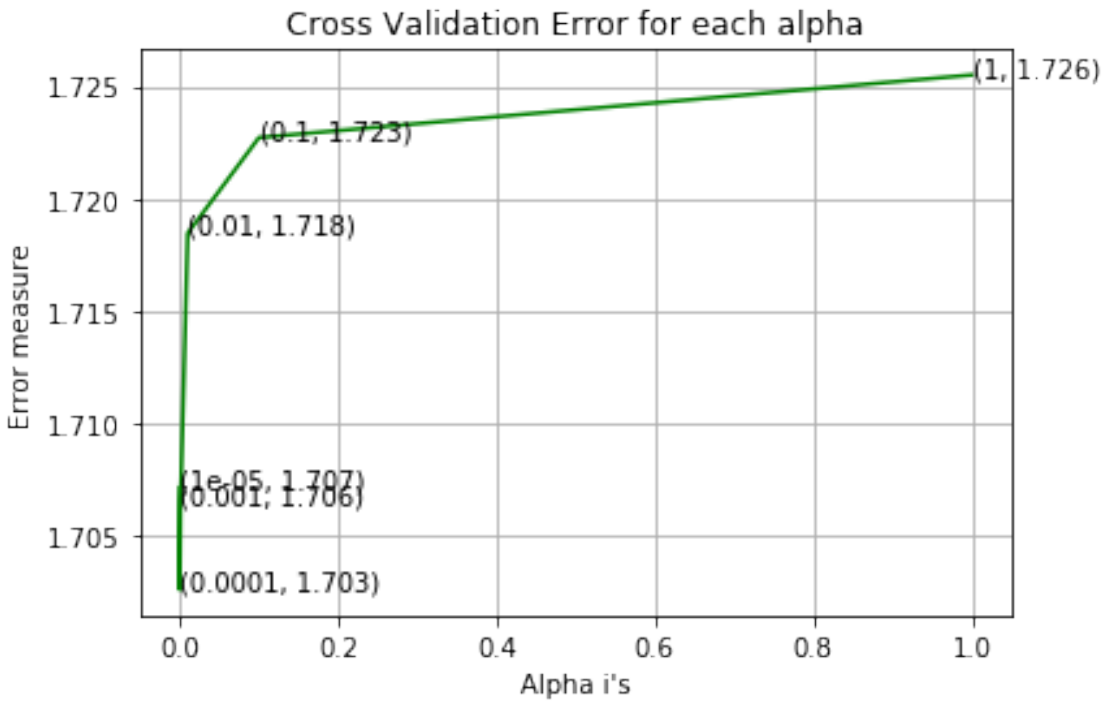
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1964)
```

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

[37]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_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: 1.7071623295500982
For values of alpha =  0.0001 The log loss is: 1.702618708351819
For values of alpha =  0.001 The log loss is: 1.70649003123691
For values of alpha =  0.01 The log loss is: 1.7184673059054407
For values of alpha =  0.1 The log loss is: 1.7227201334477416
For values of alpha =  1 The log loss is: 1.725503531573187
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7253181330622179
For values of best alpha =  0.0001 The cross validation log loss is:
1.702618708351819
For values of best alpha =  0.0001 The test log loss is: 1.7064956919066074
```

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.

[38]:
```
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  1933  genes in test and cross
validation data sets?
Ans
1. In test data 74 out of 665 : 11.12781954887218
2. In cross validation data 50 out of  532 : 9.398496240601503
```

3.2.3 Univariate Analysis on Text Feature

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

[39]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

[40]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/
 ↪(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/
 ↪len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

[41]:
```
# building a TfidfVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer = TfidfVectorizer(min_df=3, max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.
 ↪fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

[42]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

[43]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

[44]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/
 ↪train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
 ↪test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/
 ↪cv_text_feature_responseCoding.sum(axis=1)).T
```

[45]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding,␣
 ↪axis=0)
```

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

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

[46]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] ,␣
 ↪reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

[47]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({254.46916604909603: 1, 178.7519899042571: 1, 134.4055196483938: 1,
128.92767813536014: 1, 127.43989683935531: 1, 118.18091739011: 1,
117.68732619034948: 1, 116.08762058846409: 1, 112.83497391604477: 1,
106.30528521654975: 1, 105.92014884855975: 1, 90.51595280655937: 1,
90.39715817937883: 1, 87.73019009824037: 1, 80.80589617001333: 1,
79.46970825936218: 1, 79.34266207938153: 1, 79.22727879100363: 1,
77.27951770421163: 1, 77.14928235256149: 1, 76.7017582793815: 1,
73.72819252399816: 1, 71.46154619080137: 1, 71.35167612257806: 1,
69.78424530339775: 1, 68.851518399844: 1, 67.13538919774149: 1,
65.19426823763317: 1, 64.27548978746297: 1, 64.24705821839257: 1,
64.11816256455405: 1, 63.92350412917135: 1, 61.47507349334704: 1,
61.439137463975506: 1, 59.98857842441713: 1, 57.56028996310906: 1,
56.465676779049055: 1, 55.94150130662769: 1, 55.777608404661976: 1,
51.163823559134755: 1, 50.58909018997646: 1, 49.96875369157158: 1,
49.20586607779318: 1, 48.182406370320024: 1, 47.406366638606684: 1,
47.159766336978635: 1, 46.25363030982292: 1, 45.56181909477694: 1,
45.50094426401222: 1, 45.37476522049017: 1, 44.18313476349162: 1,
44.05605874546634: 1, 43.69836478760555: 1, 43.54978672167443: 1,
43.53052638869878: 1, 43.15721292878145: 1, 42.89238244821438: 1,
42.337216276992166: 1, 42.1447307166778: 1, 41.729455939325796: 1,
41.71385555720709: 1, 41.681110599055124: 1, 41.295968876583686: 1,
41.18976444601503: 1, 40.935656237551505: 1, 39.99293467093958: 1,
39.77833442996668: 1, 39.261917464385064: 1, 39.12991772314607: 1,
38.55589722342535: 1, 38.321319738326196: 1, 38.268582189868404: 1,
37.74609126751558: 1, 37.61467516257918: 1, 37.522812445494644: 1,
37.510479063048074: 1, 37.336693398556484: 1, 36.94033819575079: 1,
36.59598124697919: 1, 36.36435593195641: 1, 35.696276631904574: 1,
35.67593657564361: 1, 35.40231107571379: 1, 34.961790158414324: 1,
```

34.41868872054354: 1, 34.37886046917055: 1, 34.332445859383554: 1,
34.32794515515167: 1, 34.188519659298635: 1, 33.4614355689589: 1,
33.40066135303998: 1, 33.38975481543297: 1, 33.267365040191194: 1,
33.2226087006965: 1, 32.933665423391815: 1, 32.647282934887514: 1,
32.53368372701058: 1, 32.399666132880434: 1, 32.3146102464679: 1,
32.14777145103696: 1, 31.89739717038373: 1, 31.75217740846773: 1,
31.73511087345815: 1, 31.7041544862655: 1, 31.638256116897594: 1,
31.3257697974001: 1, 31.31890842662313: 1, 31.14936197260927: 1,
31.137764768987836: 1, 30.944040432948242: 1, 30.912702068311642: 1,
30.785378080355386: 1, 30.769077287878474: 1, 30.709217721887455: 1,
30.621726280870536: 1, 30.571633326771604: 1, 30.420468099850826: 1,
30.33588584474018: 1, 30.315461638433188: 1, 30.283250052186126: 1,
30.241513221796147: 1, 30.169273952962357: 1, 30.020940928666224: 1,
29.669240637452546: 1, 29.632445606166037: 1, 29.340563490208023: 1,
29.218157817487498: 1, 28.968916044700073: 1, 28.784357985532523: 1,
28.647297083589745: 1, 28.324728165185434: 1, 28.03591754233962: 1,
27.942694162550847: 1, 27.727492587441496: 1, 27.662217174817624: 1,
27.617372342945437: 1, 27.554003619764856: 1, 27.332917511364514: 1,
27.310036906120015: 1, 27.26843745933464: 1, 27.076641879844495: 1,
27.020879573048042: 1, 26.725753722667925: 1, 26.32587490814571: 1,
26.240788613431132: 1, 26.170003558212457: 1, 26.074854217314225: 1,
25.762985554557904: 1, 25.703787640279753: 1, 25.69308450632181: 1,
25.56989697806742: 1, 25.564832679011015: 1, 25.467163817519875: 1,
25.166824320068525: 1, 25.15385897676289: 1, 25.053363565799046: 1,
25.049103723668452: 1, 25.01875522491175: 1, 24.885382159841228: 1,
24.85580911380685: 1, 24.81493653220052: 1, 24.660508652264017: 1,
24.53814918069518: 1, 24.47240711102866: 1, 24.41151041367867: 1,
24.370870519539732: 1, 24.368576267987788: 1, 24.363426950990917: 1,
24.228835134073233: 1, 24.193731650801688: 1, 24.16638391110529: 1,
24.139056793128688: 1, 23.996242284725867: 1, 23.935138174996094: 1,
23.894435240949534: 1, 23.697134568380626: 1, 23.68914299473156: 1,
23.66772331327607: 1, 23.437729362784623: 1, 23.385486820917944: 1,
23.369935512062224: 1, 23.185869808387896: 1, 23.095127299801167: 1,
23.06192002568882: 1, 23.034554113555593: 1, 22.98891306414724: 1,
22.796760502720826: 1, 22.766309691391996: 1, 22.760232529548727: 1,
22.708726363835858: 1, 22.70146261188975: 1, 22.553211371797875: 1,
22.543799756188275: 1, 22.531855072510883: 1, 22.4802073681143: 1,
22.47351661020557: 1, 22.41178988498305: 1, 22.411707044622073: 1,
22.403462143514574: 1, 22.37914904053357: 1, 22.369803795755477: 1,
22.302271012921526: 1, 22.300632798618757: 1, 22.186605848835992: 1,
22.184071653097483: 1, 22.074352359961846: 1, 22.067084257247814: 1,
22.031377632372795: 1, 22.014957273564328: 1, 21.891521631396916: 1,
21.782167532174416: 1, 21.73427236815352: 1, 21.723109537530945: 1,
21.684190885891585: 1, 21.603639838114447: 1, 21.575162882465076: 1,
21.5751310234885: 1, 21.50881019471117: 1, 21.476094257192866: 1,
21.47169821665163: 1, 21.432781569540857: 1, 21.41972023963926: 1,
21.364314601873605: 1, 21.27957415571981: 1, 21.234866424652644: 1,
21.21803737465972: 1, 21.14201246022102: 1, 21.007208457623552: 1,

20.996749575108822: 1, 20.988633914757507: 1, 20.813329275199006: 1,
20.81029370037865: 1, 20.672546771075677: 1, 20.67057478632192: 1,
20.65890445903071: 1, 20.629612992080705: 1, 20.627689559259803: 1,
20.468885949489895: 1, 20.430051657683784: 1, 20.390573854749125: 1,
20.379632581652874: 1, 20.302652683090862: 1, 20.300768611986175: 1,
20.274907158593702: 1, 20.209494940702584: 1, 20.189409266198016: 1,
20.150695092150027: 1, 20.089381528884193: 1, 19.971454061543504: 1,
19.91489481524316: 1, 19.911823539021327: 1, 19.877220053444983: 1,
19.856440181270365: 1, 19.80318072443159: 1, 19.750565021234248: 1,
19.725854797062826: 1, 19.7097268274931: 1, 19.69733072953669: 1,
19.68571649470016: 1, 19.554558036295614: 1, 19.46940737208575: 1,
19.46642334114044: 1, 19.31246435334384: 1, 19.247934148860484: 1,
19.212217950960177: 1, 19.204251804195035: 1, 19.201991535790235: 1,
19.14830082032927: 1, 19.129790198099755: 1, 19.129197076401496: 1,
19.128138911605593: 1, 19.114469912652584: 1, 19.091793890882336: 1,
19.075345663840835: 1, 19.040518638608127: 1, 19.034992006091922: 1,
18.975717638571126: 1, 18.956191890468673: 1, 18.947021575297242: 1,
18.93756479711425: 1, 18.866027620870785: 1, 18.853313809207148: 1,
18.679493031349327: 1, 18.65307719609988: 1, 18.57595540967902: 1,
18.56612128822028: 1, 18.481353054387483: 1, 18.426203986782696: 1,
18.411192920917635: 1, 18.377055146947608: 1, 18.37091442923131: 1,
18.368809319407006: 1, 18.35405356819605: 1, 18.28804033720769: 1,
18.195297548935564: 1, 18.15517570973663: 1, 18.146444978673664: 1,
18.11915968965761: 1, 18.115381715252827: 1, 18.08671490533399: 1,
18.018964895241304: 1, 18.01387122295723: 1, 17.999794967183075: 1,
17.936118207942318: 1, 17.895457032813766: 1, 17.871431219999078: 1,
17.79443849879477: 1, 17.75595783715172: 1, 17.677738125025265: 1,
17.568694367714464: 1, 17.53660492885675: 1, 17.51319173662807: 1,
17.502473109190973: 1, 17.456500780717345: 1, 17.453553607506592: 1,
17.422389570233097: 1, 17.422326291528453: 1, 17.40159962024269: 1,
17.372819191174738: 1, 17.365454196713042: 1, 17.336381603992557: 1,
17.329741056153534: 1, 17.2940949414426: 1, 17.282682354768085: 1,
17.253964617829514: 1, 17.223187248391685: 1, 17.21220748698649: 1,
17.180725686357924: 1, 17.147723254985227: 1, 17.146634110391336: 1,
17.115879077287083: 1, 17.08923573752972: 1, 17.066275333229207: 1,
17.06204747836101: 1, 17.040516072257702: 1, 16.96757308472036: 1,
16.96740298328688: 1, 16.893733500506258: 1, 16.84397326817933: 1,
16.843352786238135: 1, 16.830022839010336: 1, 16.82605082594341: 1,
16.818107544773163: 1, 16.80511951154787: 1, 16.799346354997372: 1,
16.797945267160955: 1, 16.76740593726042: 1, 16.764222567037717: 1,
16.761883555468646: 1, 16.747592763600036: 1, 16.740243336894572: 1,
16.728882089683065: 1, 16.710158081905334: 1, 16.63146351985067: 1,
16.601053850578076: 1, 16.474661960981912: 1, 16.459913766118063: 1,
16.45790699662438: 1, 16.456237922101344: 1, 16.446109316731082: 1,
16.44325218955449: 1, 16.44030570890614: 1, 16.431981958578433: 1,
16.393075765398365: 1, 16.332467263129743: 1, 16.24748227692147: 1,
16.234076564186402: 1, 16.232914291658613: 1, 16.21551405724661: 1,
16.172684536949284: 1, 16.164622301483064: 1, 16.15968240725983: 1,

16.155652813013358: 1, 15.996086953134217: 1, 15.946659981556035: 1,
15.941991439514748: 1, 15.89881128974504: 1, 15.873991653049648: 1,
15.824281180847304: 1, 15.802185480852806: 1, 15.74705302727174: 1,
15.69175741223004: 1, 15.672884955296555: 1, 15.668396708205764: 1,
15.654694290252943: 1, 15.64448166091358: 1, 15.635622691899346: 1,
15.618206344884696: 1, 15.575532599628287: 1, 15.565016076934102: 1,
15.538201357427154: 1, 15.518498067528636: 1, 15.507271967121318: 1,
15.49735491534095: 1, 15.489209337274469: 1, 15.455838269897159: 1,
15.395700431625233: 1, 15.383863543599062: 1, 15.379388194083083: 1,
15.367579976545453: 1, 15.225950091900097: 1, 15.218785848382574: 1,
15.206723315086615: 1, 15.17511322584628: 1, 15.161297579593883: 1,
15.142855264349626: 1, 15.090645857832824: 1, 15.074080149368031: 1,
15.07263227088583: 1, 15.03544536003048: 1, 14.97950151139789: 1,
14.97232404801114: 1, 14.932087474563192: 1, 14.923684203606472: 1,
14.896624290536442: 1, 14.868308887650388: 1, 14.865054358540677: 1,
14.852724352582037: 1, 14.848426567566515: 1, 14.847145849106582: 1,
14.825838250625768: 1, 14.821135829632116: 1, 14.818463341117484: 1,
14.811313904414943: 1, 14.804812067938977: 1, 14.801728973431139: 1,
14.801627422228085: 1, 14.733387852048784: 1, 14.72815063632778: 1,
14.701614472225701: 1, 14.689891466594418: 1, 14.655899354045358: 1,
14.64618478630174: 1, 14.643524814896423: 1, 14.604843051044439: 1,
14.596663909323192: 1, 14.590008234936638: 1, 14.586613255273601: 1,
14.568351376923907: 1, 14.566942171804298: 1, 14.52774801166416: 1,
14.51307738763753: 1, 14.506095837873616: 1, 14.505593479408521: 1,
14.470359574629628: 1, 14.46435476629134: 1, 14.425831691220997: 1,
14.411649263380236: 1, 14.410328827217745: 1, 14.399278122108731: 1,
14.394485818415523: 1, 14.388198413956989: 1, 14.349838426624636: 1,
14.34152078441973: 1, 14.33803870294735: 1, 14.31400444629287: 1,
14.309087477782414: 1, 14.199461176665015: 1, 14.1318436035394: 1,
14.131750959944327: 1, 14.11092542234951: 1, 14.088603066364707: 1,
14.074887300397217: 1, 14.062165660485293: 1, 14.040006167125602: 1,
14.007455540115645: 1, 13.976129470304455: 1, 13.967383288845715: 1,
13.96682551134307: 1, 13.935443917768891: 1, 13.929188582227516: 1,
13.910266682245576: 1, 13.875726032026037: 1, 13.8683267417214: 1,
13.843268611482328: 1, 13.83295784791498: 1, 13.831015536996224: 1,
13.798691810767691: 1, 13.790510351001021: 1, 13.755548126886733: 1,
13.740116694521372: 1, 13.7345573136467: 1, 13.719562313095645: 1,
13.709265619100202: 1, 13.689329522699053: 1, 13.611008904248827: 1,
13.57809795030045: 1, 13.57259064285825: 1, 13.570645242522433: 1,
13.567954938993653: 1, 13.567256154028492: 1, 13.558894010353086: 1,
13.533211072388697: 1, 13.513458543747909: 1, 13.500274624978967: 1,
13.482755437867633: 1, 13.394340815463531: 1, 13.347202484196144: 1,
13.343994091460257: 1, 13.308789227111484: 1, 13.264510015578127: 1,
13.227782067820655: 1, 13.218554791720141: 1, 13.174228021022891: 1,
13.173780618285145: 1, 13.159731233566301: 1, 13.14140005176185: 1,
13.12346868898525: 1, 13.106264040820982: 1, 13.078768004183452: 1,
13.074056337909342: 1, 13.074018488175634: 1, 13.0642431695892: 1,
13.062483596054143: 1, 13.011243840324559: 1, 12.970140495178798: 1,

12.95814391340954: 1, 12.87144860797842: 1, 12.837890092955183: 1,
12.836885015662642: 1, 12.83085203704659: 1, 12.812326727024523: 1,
12.766951952360285: 1, 12.72413618598635: 1, 12.722666357922082: 1,
12.71975712890774: 1, 12.712263958265641: 1, 12.673121803734446: 1,
12.667789306963323: 1, 12.655183344063362: 1, 12.630853531910864: 1,
12.619328003480064: 1, 12.61175073869279: 1, 12.598644095753349: 1,
12.596370929239624: 1, 12.57850260755453: 1, 12.576700370052034: 1,
12.559115455308893: 1, 12.537324534844432: 1, 12.528697316510854: 1,
12.506910742862088: 1, 12.473321061292049: 1, 12.461910471963646: 1,
12.449750234033838: 1, 12.428667628092526: 1, 12.41893704227833: 1,
12.411979846899998: 1, 12.370033316434474: 1, 12.367912419447446: 1,
12.352754624257374: 1, 12.315100367755463: 1, 12.29885752181772: 1,
12.278699753175317: 1, 12.266911125174174: 1, 12.252642905788093: 1,
12.233702254574272: 1, 12.227499135133083: 1, 12.224733843610773: 1,
12.218007073831952: 1, 12.212740251029993: 1, 12.207391184120867: 1,
12.200954901191272: 1, 12.189562733344719: 1, 12.156807356315268: 1,
12.1313159710294: 1, 12.10849031516078: 1, 12.095128785652426: 1,
12.093188906668484: 1, 12.08287129152766: 1, 12.070749883862074: 1,
12.02253350174091: 1, 12.018929545078068: 1, 11.984163216837073: 1,
11.969324540838915: 1, 11.96471458630835: 1, 11.952594833033949: 1,
11.939039600539568: 1, 11.91291874340603: 1, 11.888116973120976: 1,
11.811442884674953: 1, 11.792540375319813: 1, 11.790383575767759: 1,
11.760879485958295: 1, 11.757990802866924: 1, 11.753796004768155: 1,
11.746280125381679: 1, 11.741772878126952: 1, 11.731640858087367: 1,
11.687154026316765: 1, 11.628454174978831: 1, 11.616281395224028: 1,
11.609098716123261: 1, 11.601270798144029: 1, 11.599366193787086: 1,
11.591173094130603: 1, 11.568619442401761: 1, 11.540593549498052: 1,
11.526247570386039: 1, 11.517779403686031: 1, 11.480425460470862: 1,
11.474333861518135: 1, 11.460752820516978: 1, 11.458154127777163: 1,
11.451067208007736: 1, 11.430167494482298: 1, 11.41566886795135: 1,
11.411087979034534: 1, 11.409177022331768: 1, 11.396668847964142: 1,
11.393217887118082: 1, 11.359424977710743: 1, 11.356646362364751: 1,
11.350502345855167: 1, 11.338964760436166: 1, 11.282626943748499: 1,
11.265666335332597: 1, 11.263352197463353: 1, 11.25407935928936: 1,
11.252449473403798: 1, 11.242669146879486: 1, 11.242382659323464: 1,
11.241465802338386: 1, 11.190075486746126: 1, 11.187518160341991: 1,
11.178451785143654: 1, 11.175241785694269: 1, 11.174746338857688: 1,
11.1599610320699: 1, 11.150913325579822: 1, 11.145643047654069: 1,
11.13677421726689: 1, 11.126877512868013: 1, 11.125661539928613: 1,
11.12077996847826: 1, 11.111353117023642: 1, 11.03181242829906: 1,
11.026880291978133: 1, 11.024604242277512: 1, 10.978799799797399: 1,
10.978755987525666: 1, 10.975218193743045: 1, 10.962968037249391: 1,
10.959814577213013: 1, 10.9546800460824: 1, 10.924852623831594: 1,
10.905575025853933: 1, 10.869412600274973: 1, 10.854376771232173: 1,
10.850653175488352: 1, 10.836594946760245: 1, 10.831506953850935: 1,
10.821078356047044: 1, 10.819439285284131: 1, 10.811336439785318: 1,
10.809734928634766: 1, 10.804431404595796: 1, 10.802735475283017: 1,
10.79870790205351: 1, 10.790249338765149: 1, 10.768796629992048: 1,

10.761382784965075: 1, 10.758044277871514: 1, 10.738680177346884: 1,
10.704984014088389: 1, 10.702508818813408: 1, 10.693472585218306: 1,
10.686524092624175: 1, 10.679194472088403: 1, 10.66372701109951: 1,
10.602065269231185: 1, 10.587159508149352: 1, 10.573753320246578: 1,
10.57299240853384: 1, 10.540272091118236: 1, 10.536668340536647: 1,
10.527375456969684: 1, 10.5209164758871: 1, 10.51943268483582: 1,
10.512902896509107: 1, 10.512888476642074: 1, 10.511028816485426: 1,
10.508091748571722: 1, 10.506213291952022: 1, 10.505096428520153: 1,
10.489682579625995: 1, 10.466175882809774: 1, 10.460630147533523: 1,
10.438651926592422: 1, 10.43388463893079: 1, 10.414922160163627: 1,
10.413884262396815: 1, 10.412871516531741: 1, 10.406389366502856: 1,
10.379545612192853: 1, 10.359012578824979: 1, 10.345343232405677: 1,
10.336188318797614: 1, 10.312759877275028: 1, 10.301314205291753: 1,
10.297623328347923: 1, 10.291857716362511: 1, 10.275059956390313: 1,
10.274170016162225: 1, 10.24586644344629: 1, 10.238681851841118: 1,
10.201719505254518: 1, 10.19555713129644: 1, 10.18772091695346: 1,
10.126982003052962: 1, 10.121299190857636: 1, 10.115270395090285: 1,
10.111309646493192: 1, 10.096591092425228: 1, 10.094511207958968: 1,
10.066884426499286: 1, 10.044057472489769: 1, 10.042274617606994: 1,
10.040477419486452: 1, 10.03647092458834: 1, 10.035859803640577: 1,
10.031581208640475: 1, 10.028151035120603: 1, 10.023255633682881: 1,
10.018228866558896: 1, 10.015226173619242: 1, 10.002234361995718: 1,
9.95390255598759: 1, 9.95034156313269: 1, 9.935344031844465: 1,
9.933166049458196: 1, 9.931276877976131: 1, 9.904885668532305: 1,
9.862441253299869: 1, 9.861277003944327: 1, 9.854638878554587: 1,
9.852890184170008: 1, 9.838180281565483: 1, 9.83717219230565: 1,
9.831945203301135: 1, 9.826877049910669: 1, 9.806409253735096: 1,
9.796903798980416: 1, 9.794544403579408: 1, 9.788773831424905: 1,
9.74849636882641: 1, 9.740197718693064: 1, 9.739893427162098: 1,
9.734315049441502: 1, 9.723169050506943: 1, 9.711481703695906: 1,
9.708338797870786: 1, 9.7066456036728: 1, 9.702391828213862: 1,
9.702136559360353: 1, 9.691762095968318: 1, 9.678177207010581: 1,
9.677589702754485: 1, 9.664004530780689: 1, 9.653582443956099: 1,
9.622384930724996: 1, 9.621617519408625: 1, 9.602191444668808: 1,
9.587680623194629: 1, 9.5559534307209: 1, 9.545049382709541: 1,
9.543233466410722: 1, 9.506903174011278: 1, 9.50634082375761: 1,
9.502751388424095: 1, 9.502277650557874: 1, 9.489439242339428: 1,
9.488628165376374: 1, 9.487218572816024: 1, 9.471476086870599: 1,
9.449238046968555: 1, 9.448789749106847: 1, 9.400447817605764: 1,
9.378869571860458: 1, 9.378148376820304: 1, 9.37489581635006: 1,
9.37387235553753: 1, 9.369659192529424: 1, 9.35866467984035: 1,
9.358034429966551: 1, 9.336929840586945: 1, 9.31294153313196: 1,
9.291792367049325: 1, 9.284320872499952: 1, 9.274925186133684: 1,
9.270035593463284: 1, 9.26384065335728: 1, 9.257481555875332: 1,
9.248282787414732: 1, 9.245833016012963: 1, 9.237068439939954: 1,
9.233134891945632: 1, 9.232972973121496: 1, 9.211421148913429: 1,
9.209386154499434: 1, 9.202602233136819: 1, 9.202186815003632: 1,
9.184990156201003: 1, 9.164774772717797: 1, 9.14463445882601: 1,

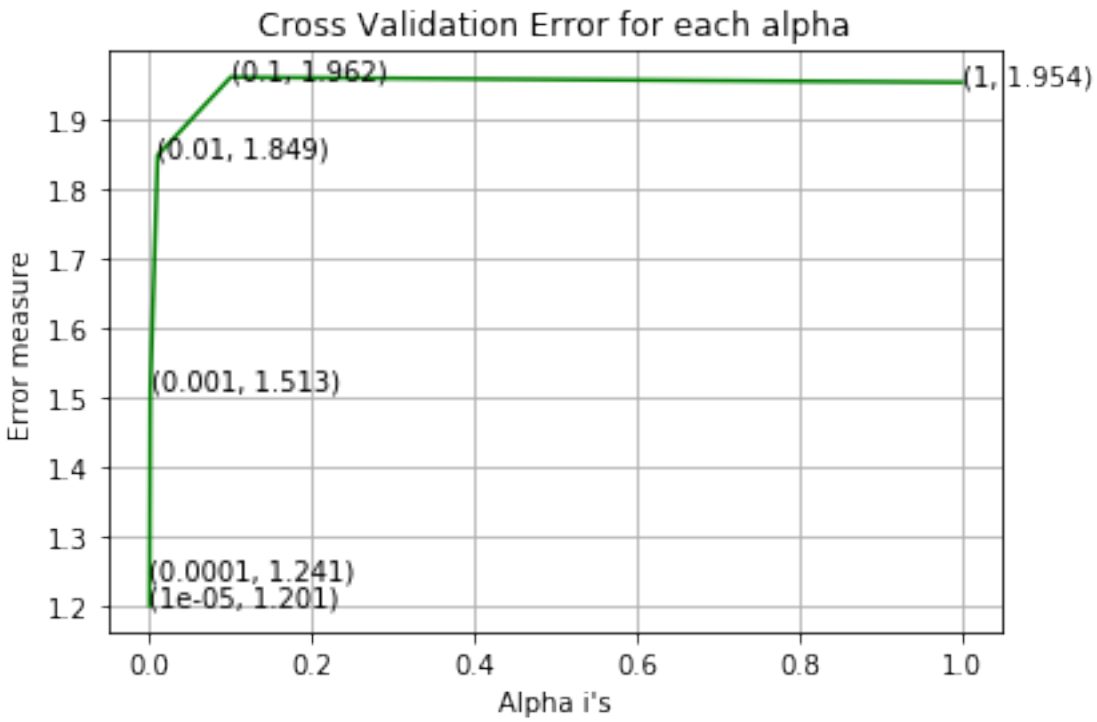
9.138120047387462: 1, 9.133470492656379: 1, 9.122963110876983: 1,
9.11687991967198: 1, 9.114980574814528: 1, 9.114483369930474: 1,
9.098396698999402: 1, 9.097690940380938: 1, 9.089301265857538: 1,
9.079102947582458: 1, 9.055009196080366: 1, 9.048012495877646: 1,
9.043929953882824: 1, 9.03778963736592: 1, 9.035809379513472: 1,
9.030769525216968: 1, 9.02383529683877: 1, 9.022889662988442: 1,
9.002108605637183: 1, 8.964260077060512: 1, 8.964172064732017: 1,
8.953219763427105: 1, 8.95257603877519: 1, 8.947318507029708: 1,
8.946844749679945: 1, 8.942967628628859: 1, 8.93946758369138: 1,
8.933838577026565: 1, 8.915455879146396: 1, 8.910885813424379: 1,
8.902221784148685: 1, 8.898227534167733: 1, 8.886799747472404: 1,
8.876550759819407: 1, 8.871291676630726: 1, 8.859204588728346: 1,
8.857281595926143: 1, 8.854342228639826: 1, 8.851913732908564: 1,
8.848479804330179: 1, 8.844907378675572: 1, 8.839848473394769: 1,
8.831218826131904: 1, 8.808783139920955: 1, 8.800632161430276: 1,
8.787813285766438: 1, 8.780442224875454: 1, 8.767893457109492: 1,
8.76316970607573: 1, 8.744662632764816: 1, 8.734206717137875: 1,
8.730392640275747: 1, 8.699098450122307: 1, 8.698055121802525: 1,
8.6898826331375: 1, 8.671758372856925: 1, 8.664701372354537: 1,
8.631042277116178: 1, 8.627605133463344: 1, 8.619192449385622: 1,
8.601188358607281: 1, 8.596003624118175: 1, 8.582464474703098: 1,
8.574955558848421: 1, 8.555040459500974: 1, 8.553166785854522: 1,
8.546050644275349: 1, 8.543426959236212: 1, 8.535740185672731: 1,
8.50952553362842: 1, 8.508752637395279: 1, 8.494404299955582: 1,
8.450864304403348: 1, 8.41875682751001: 1, 8.413656943341723: 1,
8.411762502556032: 1, 8.397424784822052: 1, 8.369651305780934: 1,
8.349172238853962: 1, 8.348809493182058: 1, 8.339737928644118: 1,
8.314510823445634: 1, 8.314058245765695: 1, 8.302538673725774: 1,
8.29992307860964: 1, 8.27904637818819: 1, 8.253160691665524: 1,
8.250830871152713: 1, 8.231186106304628: 1, 8.228002351639471: 1,
8.214663279117588: 1, 8.209967606123916: 1, 8.203274926218196: 1,
8.19833862935691: 1, 8.197200561165852: 1, 8.19065390186709: 1,
8.187858296703755: 1, 8.179617591678653: 1, 8.174979932827513: 1,
8.16208256332374: 1, 8.15841846195087: 1, 8.155714778711332: 1,
8.150993587010095: 1, 8.147786653415533: 1, 8.105800340690175: 1,
8.105330106338862: 1, 8.093963172176839: 1, 8.08738449171252: 1,
8.085753920612353: 1, 8.084320603857151: 1, 8.05636224053171: 1,
8.046074435332915: 1, 8.041023011149408: 1, 8.00575206299067: 1,
7.9958589611407485: 1, 7.995099185275508: 1, 7.97814493888041: 1,
7.974034494777867: 1, 7.973209203773578: 1, 7.970118271300056: 1,
7.9445827602806505: 1, 7.929689129388855: 1, 7.919556280190104: 1,
7.913508506336837: 1, 7.897260231071464: 1, 7.884146640079265: 1,
7.878969586547057: 1, 7.8780238488891055: 1, 7.861474783474172: 1,
7.854947573527355: 1, 7.853101482474474: 1, 7.824456207818423: 1,
7.811164373894561: 1, 7.804138283159543: 1, 7.801033064023153: 1,
7.776878415070246: 1, 7.771149633687276: 1, 7.767231741034737: 1,
7.749489407399401: 1, 7.7358757596504635: 1, 7.6957971706596995: 1,
7.689601132396992: 1, 7.681253167157278: 1, 7.661948491269241: 1,

```
7.633826489297405: 1, 7.629473680361035: 1, 7.625718402778314: 1,
7.623700658396205: 1, 7.618374228774808: 1, 7.61136425418471: 1,
7.60410912031012: 1, 7.58938451028133: 1, 7.544795571246535: 1,
7.540031779778488: 1, 7.496004101294014: 1, 7.486257272204843: 1,
7.482158124646504: 1, 7.478441370011276: 1, 7.476280262264832: 1,
7.463837446773283: 1, 7.4380958830907415: 1, 7.436278463949515: 1,
7.427854491492638: 1, 7.415995403614553: 1, 7.406258457161479: 1,
7.375815093841288: 1, 7.349437926887106: 1, 7.34697607376379: 1,
7.346310325689485: 1, 7.327269434547441: 1, 7.307822550623611: 1,
7.301838185706084: 1, 7.296557488528226: 1, 7.263226799131885: 1,
7.2412017520622145: 1, 7.217496931246552: 1, 7.1896011173647905: 1,
7.173341890762831: 1, 7.155111303366431: 1, 7.1465832617789395: 1,
7.117197986630979: 1, 7.07716128498563: 1, 7.072141802731959: 1,
7.043026002900158: 1, 7.019818294978651: 1, 6.995086505240966: 1,
6.947384412950663: 1, 6.941393035074871: 1, 6.880494669309788: 1,
6.795565068359985: 1, 6.762466526777929: 1, 6.734696220143809: 1,
6.674585628424647: 1, 6.673448863876622: 1, 6.330412948113397: 1,
6.257731101067216: 1})
```

[48]:
```
# Train a Logistic regression+Calibration model using text features whicha re␣
 ↪on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_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: 1.2010303295199924
For values of alpha =  0.0001 The log loss is: 1.2410505990063514
For values of alpha =  0.001 The log loss is: 1.5129912417005666
For values of alpha =  0.01 The log loss is: 1.8488897861077873
For values of alpha =  0.1 The log loss is: 1.9615671781812782
For values of alpha =  1 The log loss is: 1.9541286740994612
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7236087127065252
For values of best alpha =  1e-05 The cross validation log loss is:
1.2010303295199924
For values of best alpha =  1e-05 The test log loss is: 1.0930109903927585
```

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

[49]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3, max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

[50]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train␣
 ↪data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in␣
 ↪train data")
```

```
93.9 % of word of test data appeared in train data
93.9 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

[51]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

[52]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

[53]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3, max_features=1000)

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

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

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

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

[54]:
```
# merging gene, variance and text features

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

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

train_x_onehotCoding = hstack((train_gene_var_onehotCoding,␣
 ↪train_text_feature_onehotCoding)).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))
```

[55]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",␣
 ↪train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ",␣
 ↪test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data␣
 ↪=", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3192)
(number of data points * number of features) in test data =  (665, 3192)
(number of data points * number of features) in cross validation data = (532,
3192)
```

[56]:
```
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)
```

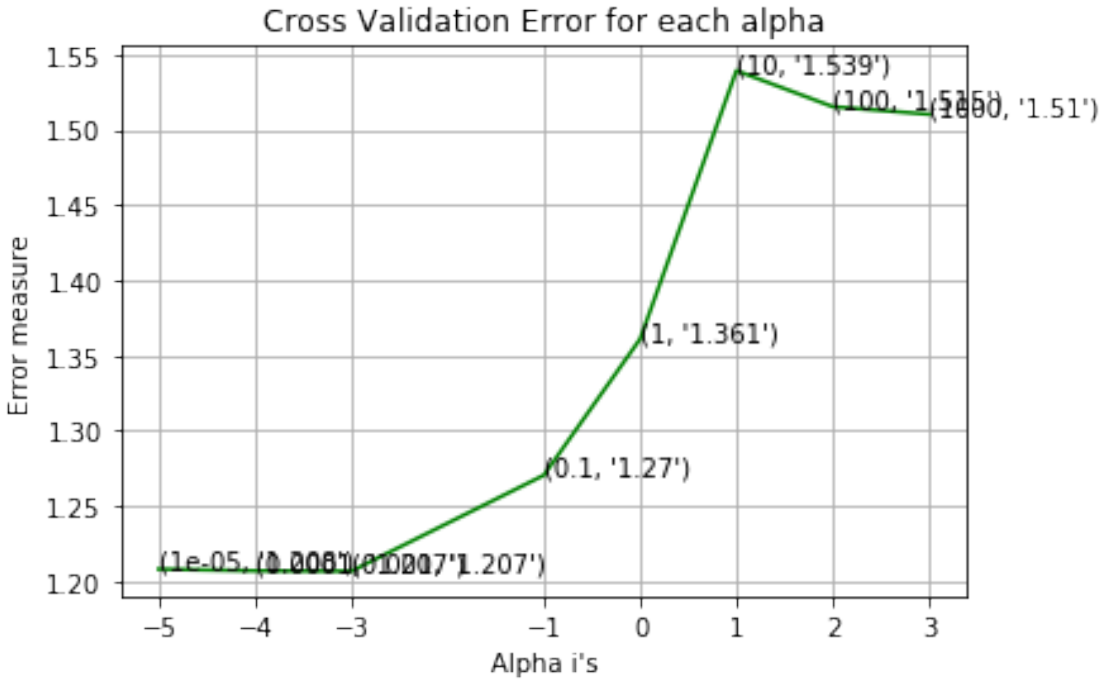
4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

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

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


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


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

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

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:
 ↪",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation␣
 ↪log loss is:",log_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 : 1.208017330731779
for alpha = 0.0001
Log Loss : 1.2069060241672867
for alpha = 0.001
Log Loss : 1.2066017622733376
```

```
for alpha = 0.1
Log Loss : 1.270380946431673
for alpha = 1
Log Loss : 1.360889251849853
for alpha = 10
Log Loss : 1.5391923175484779
for alpha = 100
Log Loss : 1.5152558008822332
for alpha = 1000
Log Loss : 1.5102170499012277
```




```
For values of best alpha =  0.001 The train log loss is: 0.5300705364381803
For values of best alpha =  0.001 The cross validation log loss is:
1.2066017622733376
For values of best alpha =  0.001 The test log loss is: 1.1849806631299795
```

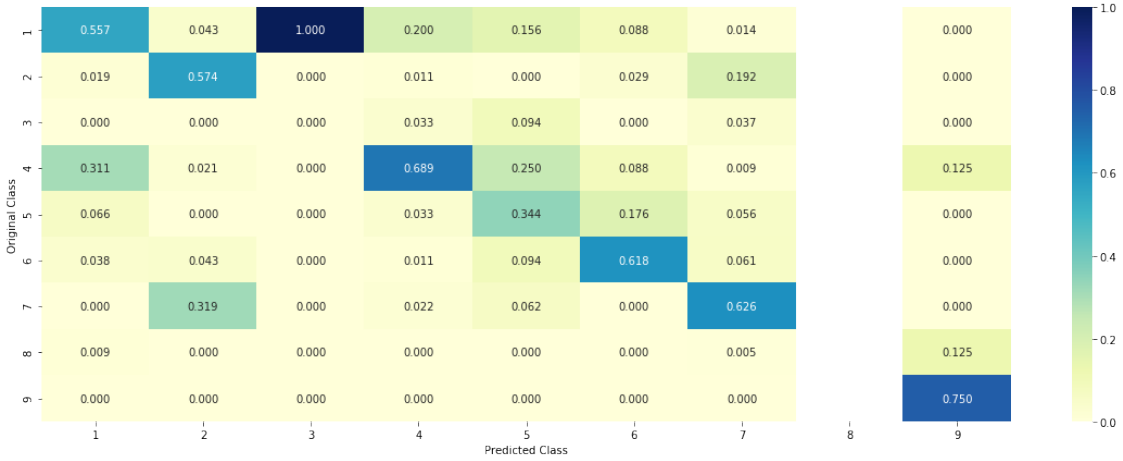
4.1.1.2. Testing the model with best hyper paramters

[58]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/
 ↪stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True,␣
 ↪class_prior=None)
```

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


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

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

```
Log Loss : 1.2066017622733376
Number of missclassified point : 0.39849624060150374
-------------------- Confusion matrix --------------------
```

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



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

4.1.1.3. Feature Importance, Correctly classified point

[59]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[0.0561 0.0432 0.0135 0.735  0.0294 0.0298
0.0859 0.0045 0.0026]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

[60]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[0.6975 0.048  0.0137 0.0721 0.0325 0.0331
0.0953 0.005  0.0029]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

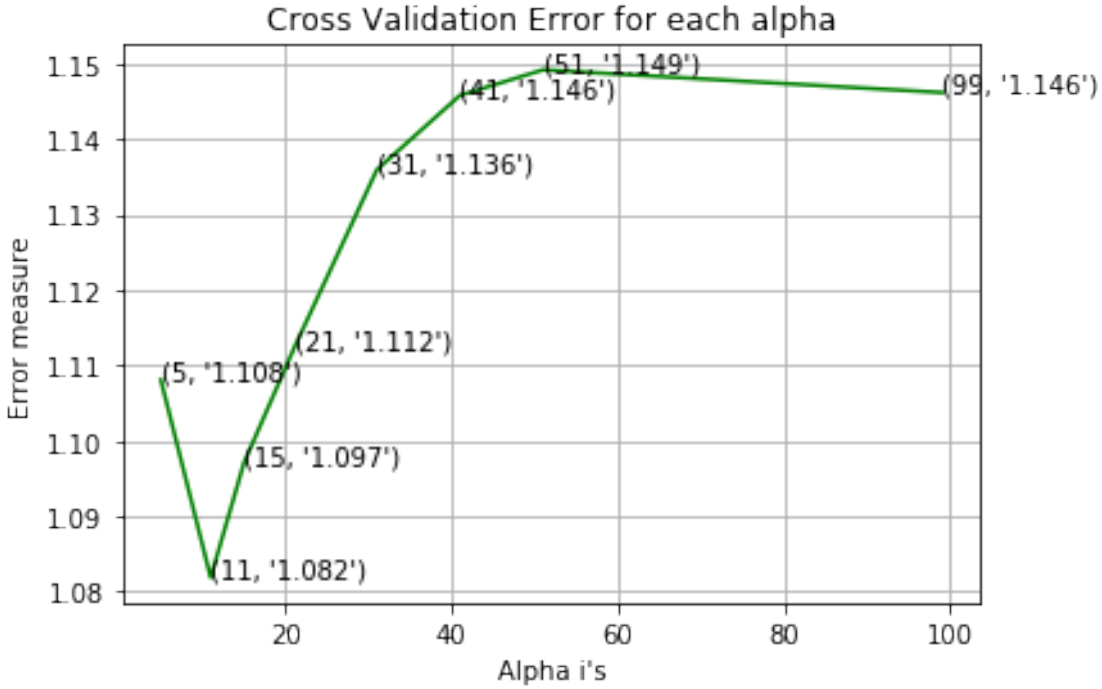
4.2.1. Hyper parameter tuning

[61]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto,␣
 ↪leaf_size=30, p=2,
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

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



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

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


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

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


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

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

```
for alpha = 5
Log Loss : 1.1080891215542723
```

```
for alpha = 11
Log Loss : 1.081876583678065
for alpha = 15
Log Loss : 1.0970542209153071
for alpha = 21
Log Loss : 1.1122114714677316
for alpha = 31
Log Loss : 1.1359300205650424
for alpha = 41
Log Loss : 1.145800609883368
for alpha = 51
Log Loss : 1.1491902561077783
for alpha = 99
Log Loss : 1.1461787275392614
```




```
For values of best alpha =  11 The train log loss is: 0.6614654615284454
For values of best alpha =  11 The cross validation log loss is:
1.081876583678065
For values of best alpha =  11 The test log loss is: 1.0549918201026147
```

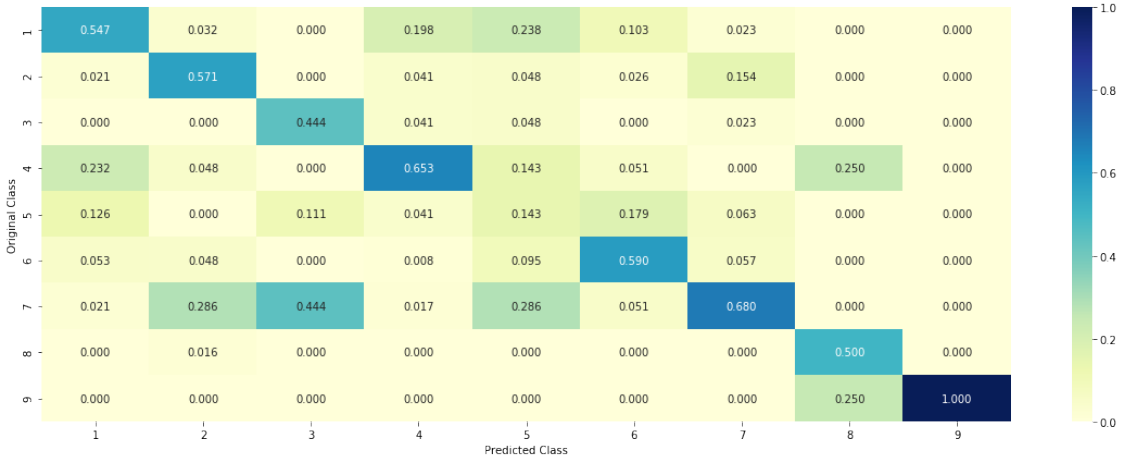
4.2.2. Testing the model with best hyper paramters

[62]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/
 ↪modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
```

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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/
 ↪lessons/k-nearest-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 : 1.081876583678065
Number of mis-classified points : 0.39285714285714285
-------------------- Confusion matrix --------------------
```


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




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

4.2.3.Sample Query point -1

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1,␣
 ↪-1), alpha[best_alpha])
```

```
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs␣
 ↪to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 4
The  11  nearest neighbours of the test points belongs to classes [4 4 4 3 4 3 4
4 4 4 4]
Fequency of nearest points : Counter({4: 9, 3: 2})
```

4.2.4. Sample Query Point-2

[64]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].
 ↪reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1,␣
 ↪-1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of␣
 ↪the test points 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 11 and the nearest neighbours of the test points belongs
to classes [1 1 1 1 1 1 1 6 1 1 1]
Fequency of nearest points : Counter({1: 10, 6: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

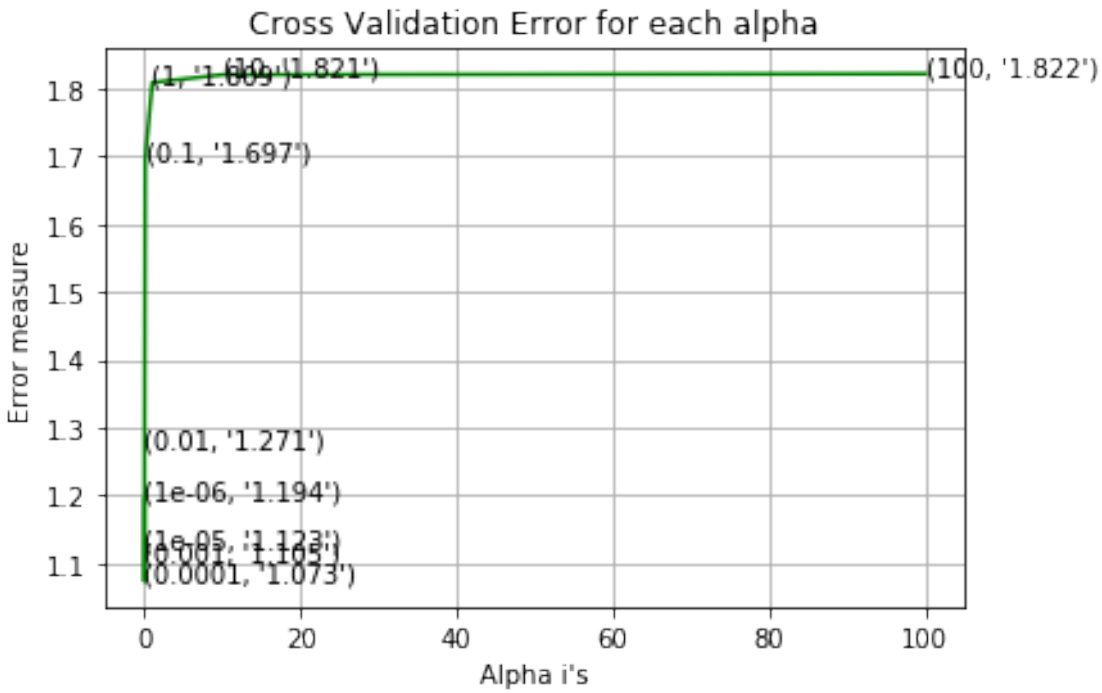
4.3.1.1. Hyper paramter tuning

[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/
#-------------------------------


# 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 : 1.1938772627283112
for alpha = 1e-05
Log Loss : 1.1227997319850649
for alpha = 0.0001
Log Loss : 1.0731158684634137
for alpha = 0.001
Log Loss : 1.1049028526263631
for alpha = 0.01
Log Loss : 1.27065332754716
for alpha = 0.1
Log Loss : 1.6969639155472531
for alpha = 1
Log Loss : 1.809330953743888
for alpha = 10
Log Loss : 1.8209016776754539
for alpha = 100
Log Loss : 1.8222154386008214
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44990620266555553
For values of best alpha =  0.0001 The cross validation log loss is:
1.0731158684634137
For values of best alpha =  0.0001 The test log loss is: 1.0076025100349446
```

4.3.1.2. Testing the model with best hyper paramters

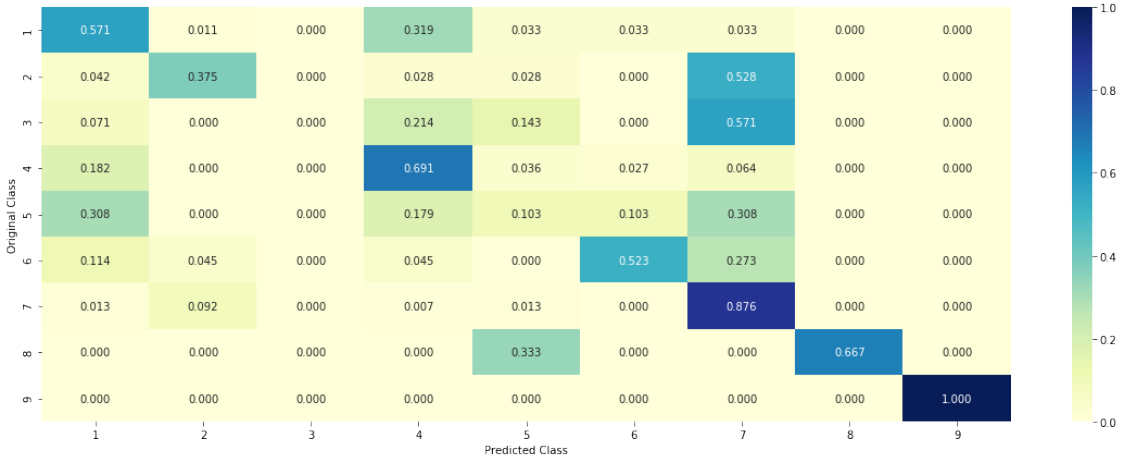
[66]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, tol=None,
# 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 : 1.0731158684634137
Number of mis-classified points : 0.39097744360902253
-------------------- Confusion matrix --------------------
```


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


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



4.3.1.3. Feature Importance

[67]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            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

[68]:
```
# 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 : 4
Predicted Class Probabilities: [[0.0174 0.007  0.0458 0.9026 0.0112 0.0049
0.0051 0.0042 0.0018]]
Actual Class : 4
--------------------------------------------------
33 Text feature [suppressor] present in test data point [True]
154 Text feature [missense] present in test data point [True]
156 Text feature [mice] present in test data point [True]
166 Text feature [would] present in test data point [True]
195 Text feature [see] present in test data point [True]
196 Text feature [representative] present in test data point [True]
200 Text feature [confer] present in test data point [True]
228 Text feature [www] present in test data point [True]
229 Text feature [due] present in test data point [True]
245 Text feature [five] present in test data point [True]
266 Text feature [46] present in test data point [True]
282 Text feature [free] present in test data point [True]
301 Text feature [inactivation] present in test data point [True]
304 Text feature [localization] present in test data point [True]
311 Text feature [resulting] present in test data point [True]
312 Text feature [activating] present in test data point [True]
317 Text feature [reduced] present in test data point [True]
325 Text feature [ability] present in test data point [True]
341 Text feature [tagged] present in test data point [True]
352 Text feature [protein] present in test data point [True]
363 Text feature [17] present in test data point [True]
365 Text feature [mammalian] present in test data point [True]
368 Text feature [nuclear] present in test data point [True]
371 Text feature [vitro] present in test data point [True]
376 Text feature [motif] present in test data point [True]
378 Text feature [previous] present in test data point [True]
401 Text feature [conditions] present in test data point [True]
412 Text feature [41] present in test data point [True]
```

```
428 Text feature [altered] present in test data point [True]
435 Text feature [rather] present in test data point [True]
438 Text feature [transfected] present in test data point [True]
439 Text feature [defective] present in test data point [True]
444 Text feature [binding] present in test data point [True]
448 Text feature [54] present in test data point [True]
451 Text feature [driven] present in test data point [True]
459 Text feature [changes] present in test data point [True]
464 Text feature [show] present in test data point [True]
475 Text feature [suggesting] present in test data point [True]
477 Text feature [19] present in test data point [True]
485 Text feature [patients] present in test data point [True]
487 Text feature [activation] present in test data point [True]
499 Text feature [germline] present in test data point [True]
Out of the top  500  features  42 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

[69]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[9.536e-01 7.400e-03 4.000e-04 2.790e-02
7.000e-04 1.000e-03 6.800e-03
  2.200e-03 1.000e-04]]
Actual Class : 1
--------------------------------------------------
37 Text feature [panel] present in test data point [True]
95 Text feature [surface] present in test data point [True]
128 Text feature [fraction] present in test data point [True]
170 Text feature [repeats] present in test data point [True]
171 Text feature [activated] present in test data point [True]
183 Text feature [page] present in test data point [True]
185 Text feature [specificity] present in test data point [True]
197 Text feature [elevated] present in test data point [True]
220 Text feature [pa] present in test data point [True]
252 Text feature [hotspot] present in test data point [True]
```

257 Text feature [american] present in test data point [True]
260 Text feature [blue] present in test data point [True]
271 Text feature [substitutions] present in test data point [True]
273 Text feature [region] present in test data point [True]
275 Text feature [sequenced] present in test data point [True]
283 Text feature [mutational] present in test data point [True]
284 Text feature [rather] present in test data point [True]
285 Text feature [melanomas] present in test data point [True]
294 Text feature [models] present in test data point [True]
301 Text feature [approximately] present in test data point [True]
306 Text feature [colorectal] present in test data point [True]
312 Text feature [classified] present in test data point [True]
316 Text feature [e2] present in test data point [True]
328 Text feature [screening] present in test data point [True]
330 Text feature [pathways] present in test data point [True]
335 Text feature [loss] present in test data point [True]
336 Text feature [12] present in test data point [True]
343 Text feature [activity] present in test data point [True]
348 Text feature [various] present in test data point [True]
352 Text feature [peptide] present in test data point [True]
353 Text feature [mm] present in test data point [True]
356 Text feature [21] present in test data point [True]
358 Text feature [intermediate] present in test data point [True]
359 Text feature [clinically] present in test data point [True]
360 Text feature [deletion] present in test data point [True]
369 Text feature [suppressor] present in test data point [True]
370 Text feature [factor] present in test data point [True]
382 Text feature [population] present in test data point [True]
385 Text feature [author] present in test data point [True]
387 Text feature [null] present in test data point [True]
391 Text feature [3t3] present in test data point [True]
394 Text feature [oncogenic] present in test data point [True]
395 Text feature [active] present in test data point [True]
398 Text feature [function] present in test data point [True]
399 Text feature [effect] present in test data point [True]
416 Text feature [trials] present in test data point [True]
417 Text feature [constitutive] present in test data point [True]
420 Text feature [express] present in test data point [True]
423 Text feature [observation] present in test data point [True]
426 Text feature [therefore] present in test data point [True]
428 Text feature [copy] present in test data point [True]
431 Text feature [functions] present in test data point [True]
432 Text feature [score] present in test data point [True]
433 Text feature [therapy] present in test data point [True]
447 Text feature [downstream] present in test data point [True]
452 Text feature [hypothesis] present in test data point [True]
458 Text feature [discovery] present in test data point [True]
464 Text feature [wild] present in test data point [True]

```
469 Text feature [binding] present in test data point [True]
472 Text feature [deficient] present in test data point [True]
475 Text feature [therapeutic] present in test data point [True]
476 Text feature [clinical] present in test data point [True]
478 Text feature [defined] present in test data point [True]
480 Text feature [factors] present in test data point [True]
482 Text feature [structure] present in test data point [True]
486 Text feature [mice] present in test data point [True]
487 Text feature [serum] present in test data point [True]
492 Text feature [chain] present in test data point [True]
493 Text feature [splicing] present in test data point [True]
499 Text feature [free] present in test data point [True]
Out of the top  500  features  70 are present in query point
```

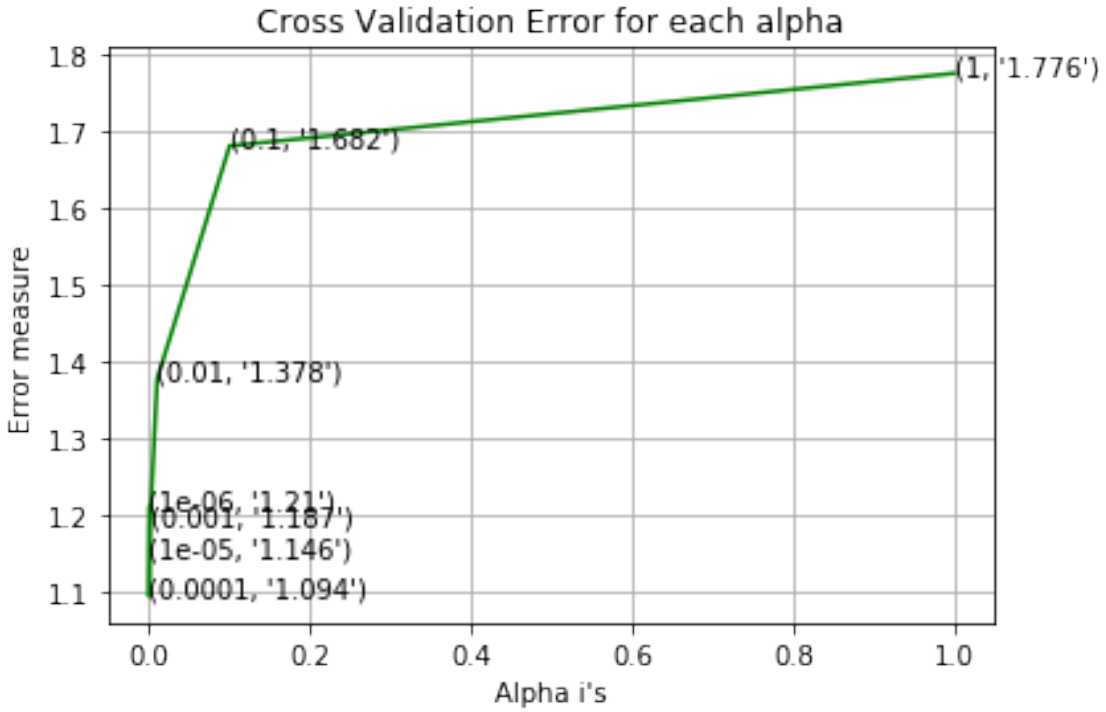
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

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

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

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



# 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 : 1.2101873892520216
for alpha = 1e-05
Log Loss : 1.1463096979061547
for alpha = 0.0001
Log Loss : 1.0944256930516638
for alpha = 0.001
Log Loss : 1.1873122368481226
for alpha = 0.01
Log Loss : 1.378245084144183
for alpha = 0.1
Log Loss : 1.6817577360052844
for alpha = 1
Log Loss : 1.7762559611918385
```



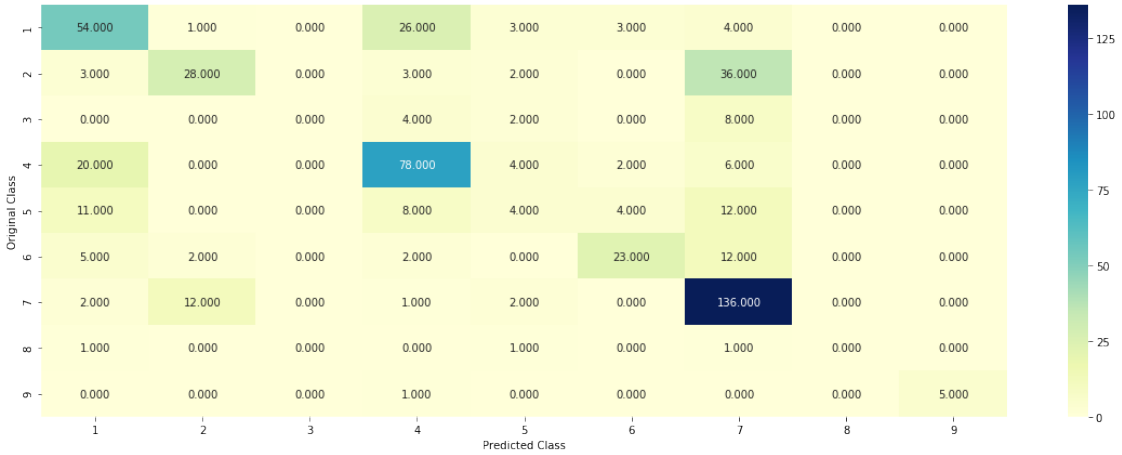

```
For values of best alpha =  0.0001 The train log loss is: 0.44617585063269427
For values of best alpha =  0.0001 The cross validation log loss is:
1.0944256930516638
For values of best alpha =  0.0001 The test log loss is: 1.015678967155341
```

4.3.2.2. Testing model with best hyper parameters

[71]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
```

```
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_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 : 1.0944256930516638
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```




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


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

4.3.2.3. Feature Importance, Correctly Classified point

[72]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[2.170e-02 7.100e-03 3.620e-02 9.083e-01
1.110e-02 4.700e-03 6.100e-03
  4.000e-03 8.000e-04]]
Actual Class : 4
--------------------------------------------------
50 Text feature [suppressor] present in test data point [True]
158 Text feature [mice] present in test data point [True]
174 Text feature [would] present in test data point [True]
189 Text feature [missense] present in test data point [True]
193 Text feature [representative] present in test data point [True]
195 Text feature [see] present in test data point [True]
213 Text feature [www] present in test data point [True]
224 Text feature [due] present in test data point [True]
248 Text feature [confer] present in test data point [True]
251 Text feature [five] present in test data point [True]
268 Text feature [free] present in test data point [True]
280 Text feature [46] present in test data point [True]
292 Text feature [resulting] present in test data point [True]
312 Text feature [localization] present in test data point [True]
319 Text feature [reduced] present in test data point [True]
350 Text feature [ability] present in test data point [True]
360 Text feature [motif] present in test data point [True]
362 Text feature [protein] present in test data point [True]
363 Text feature [inactivation] present in test data point [True]
369 Text feature [tagged] present in test data point [True]
379 Text feature [mammalian] present in test data point [True]
387 Text feature [vitro] present in test data point [True]
390 Text feature [nuclear] present in test data point [True]
401 Text feature [conditions] present in test data point [True]
409 Text feature [17] present in test data point [True]
411 Text feature [activating] present in test data point [True]
415 Text feature [previous] present in test data point [True]
433 Text feature [transfected] present in test data point [True]
455 Text feature [show] present in test data point [True]
459 Text feature [binding] present in test data point [True]
465 Text feature [suggesting] present in test data point [True]
473 Text feature [defective] present in test data point [True]
475 Text feature [rather] present in test data point [True]
479 Text feature [express] present in test data point [True]
480 Text feature [41] present in test data point [True]
499 Text feature [changes] present in test data point [True]
```

```
Out of the top  500  features  36 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

[73]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[9.393e-01 8.200e-03 3.000e-04 3.730e-02
6.000e-04 1.000e-03 8.000e-03
  5.300e-03 0.000e+00]]
Actual Class : 1
--------------------------------------------------
33 Text feature [panel] present in test data point [True]
108 Text feature [surface] present in test data point [True]
147 Text feature [fraction] present in test data point [True]
183 Text feature [page] present in test data point [True]
193 Text feature [specificity] present in test data point [True]
203 Text feature [pa] present in test data point [True]
206 Text feature [elevated] present in test data point [True]
207 Text feature [activated] present in test data point [True]
208 Text feature [repeats] present in test data point [True]
214 Text feature [21] present in test data point [True]
260 Text feature [blue] present in test data point [True]
271 Text feature [melanomas] present in test data point [True]
283 Text feature [sequenced] present in test data point [True]
284 Text feature [rather] present in test data point [True]
292 Text feature [hotspot] present in test data point [True]
295 Text feature [models] present in test data point [True]
302 Text feature [region] present in test data point [True]
307 Text feature [mutational] present in test data point [True]
312 Text feature [colorectal] present in test data point [True]
317 Text feature [substitutions] present in test data point [True]
318 Text feature [loss] present in test data point [True]
321 Text feature [american] present in test data point [True]
324 Text feature [mm] present in test data point [True]
326 Text feature [screening] present in test data point [True]
```

```
327 Text feature [approximately] present in test data point [True]
333 Text feature [classified] present in test data point [True]
352 Text feature [suppressor] present in test data point [True]
355 Text feature [various] present in test data point [True]
357 Text feature [pathways] present in test data point [True]
360 Text feature [peptide] present in test data point [True]
361 Text feature [author] present in test data point [True]
364 Text feature [null] present in test data point [True]
365 Text feature [activity] present in test data point [True]
368 Text feature [e2] present in test data point [True]
369 Text feature [12] present in test data point [True]
371 Text feature [effect] present in test data point [True]
381 Text feature [intermediate] present in test data point [True]
394 Text feature [function] present in test data point [True]
396 Text feature [3t3] present in test data point [True]
397 Text feature [factor] present in test data point [True]
401 Text feature [deletion] present in test data point [True]
404 Text feature [clinically] present in test data point [True]
405 Text feature [population] present in test data point [True]
411 Text feature [score] present in test data point [True]
414 Text feature [hypothesis] present in test data point [True]
416 Text feature [therefore] present in test data point [True]
419 Text feature [observation] present in test data point [True]
420 Text feature [copy] present in test data point [True]
433 Text feature [wild] present in test data point [True]
437 Text feature [previous] present in test data point [True]
439 Text feature [trials] present in test data point [True]
440 Text feature [therapy] present in test data point [True]
451 Text feature [functions] present in test data point [True]
452 Text feature [mice] present in test data point [True]
458 Text feature [active] present in test data point [True]
459 Text feature [encoding] present in test data point [True]
464 Text feature [express] present in test data point [True]
466 Text feature [oncogenic] present in test data point [True]
471 Text feature [splicing] present in test data point [True]
474 Text feature [factors] present in test data point [True]
479 Text feature [www] present in test data point [True]
485 Text feature [chain] present in test data point [True]
488 Text feature [constitutive] present in test data point [True]
496 Text feature [rt] present in test data point [True]
497 Text feature [binding] present in test data point [True]
498 Text feature [free] present in test data point [True]
Out of the top  500  features  66 are present in query point
```

4.4. Linear Support Vector Machines

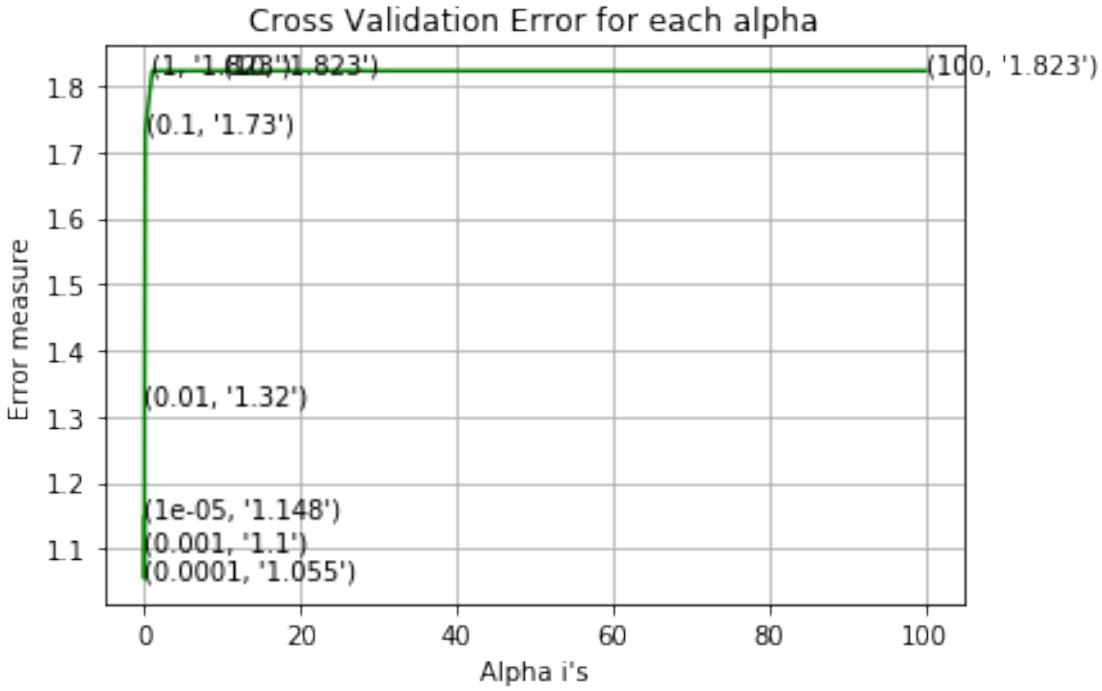
4.4.1. Hyper paramter tuning

[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/
# --------------------------------



# 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 : 1.1477075759705713
for C = 0.0001
Log Loss : 1.0546829663555854
for C = 0.001
Log Loss : 1.0996675274613494
for C = 0.01
Log Loss : 1.3200449767970004
for C = 0.1
Log Loss : 1.7302780961799578
for C = 1
Log Loss : 1.8225496054989652
```

```
for C = 10
Log Loss : 1.8225507322474714
for C = 100
Log Loss : 1.8225506411054362
```




```
For values of best alpha =  0.0001 The train log loss is: 0.39303472996775923
For values of best alpha =  0.0001 The cross validation log loss is:
1.0546829663555854
For values of best alpha =  0.0001 The test log loss is: 1.0386026548677745
```

4.4.2. Testing model with best hyper parameters

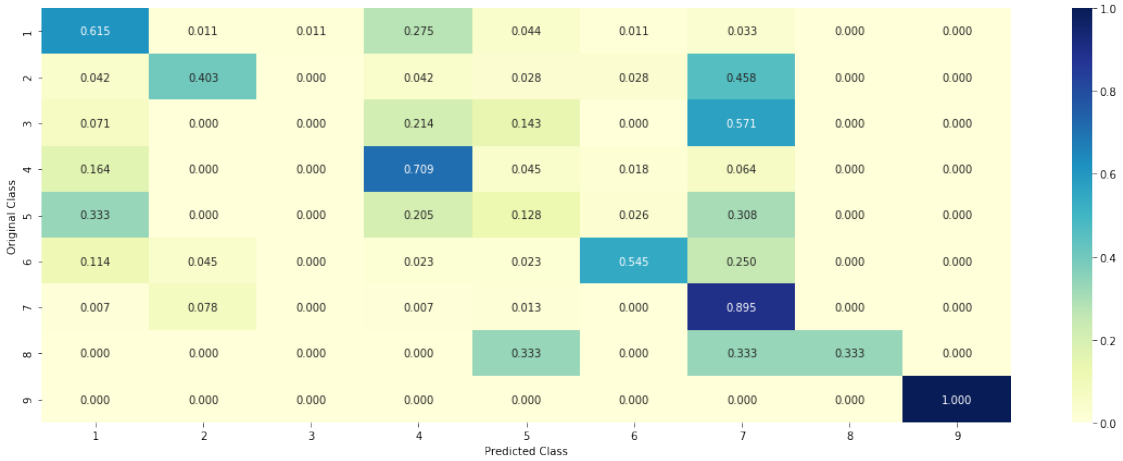
[75]:
```
# read more about support vector machines with linear kernals here http://
 ↪scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# ------------------------------
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True,␣
 ↪probability=False, 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 : 1.0546829663555854
Number of mis-classified points : 0.3684210526315789
--------------------- 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

[76]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',␣
 ↪random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
```

```
indices = np.argsort(-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: [[0.0282 0.0214 0.0512 0.8119 0.0297 0.0257
0.0242 0.0042 0.0034]]
Actual Class : 4
--------------------------------------------------
135 Text feature [mice] present in test data point [True]
136 Text feature [suppressor] present in test data point [True]
203 Text feature [due] present in test data point [True]
204 Text feature [representative] present in test data point [True]
205 Text feature [resulting] present in test data point [True]
206 Text feature [free] present in test data point [True]
208 Text feature [see] present in test data point [True]
210 Text feature [www] present in test data point [True]
215 Text feature [would] present in test data point [True]
385 Text feature [46] present in test data point [True]
386 Text feature [regions] present in test data point [True]
387 Text feature [show] present in test data point [True]
389 Text feature [vitro] present in test data point [True]
390 Text feature [five] present in test data point [True]
393 Text feature [panel] present in test data point [True]
397 Text feature [conditions] present in test data point [True]
402 Text feature [sporadic] present in test data point [True]
404 Text feature [missense] present in test data point [True]
405 Text feature [ability] present in test data point [True]
407 Text feature [binding] present in test data point [True]
Out of the top  500  features  20 are present in query point
```

4.3.3.2. For Incorrectly classified point

[77]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[9.177e-01 2.700e-02 2.000e-04 3.500e-02
9.000e-04 4.500e-03 1.260e-02
  1.800e-03 3.000e-04]]
Actual Class : 1
--------------------------------------------------
22 Text feature [21] present in test data point [True]
55 Text feature [panel] present in test data point [True]
143 Text feature [copy] present in test data point [True]
144 Text feature [melanomas] present in test data point [True]
237 Text feature [previous] present in test data point [True]
330 Text feature [page] present in test data point [True]
333 Text feature [blue] present in test data point [True]
335 Text feature [sequenced] present in test data point [True]
336 Text feature [screening] present in test data point [True]
337 Text feature [colorectal] present in test data point [True]
338 Text feature [surface] present in test data point [True]
339 Text feature [elevated] present in test data point [True]
340 Text feature [vitro] present in test data point [True]
341 Text feature [frame] present in test data point [True]
342 Text feature [rates] present in test data point [True]
343 Text feature [like] present in test data point [True]
344 Text feature [rather] present in test data point [True]
345 Text feature [loss] present in test data point [True]
346 Text feature [suppressor] present in test data point [True]
347 Text feature [trials] present in test data point [True]
348 Text feature [repeats] present in test data point [True]
349 Text feature [fraction] present in test data point [True]
350 Text feature [encoding] present in test data point [True]
351 Text feature [observation] present in test data point [True]
352 Text feature [intermediate] present in test data point [True]
473 Text feature [mutational] present in test data point [True]
474 Text feature [www] present in test data point [True]
475 Text feature [score] present in test data point [True]
476 Text feature [discovery] present in test data point [True]
478 Text feature [hypothesis] present in test data point [True]
479 Text feature [wild] present in test data point [True]
480 Text feature [activated] present in test data point [True]
481 Text feature [pa] present in test data point [True]
482 Text feature [various] present in test data point [True]
484 Text feature [play] present in test data point [True]
485 Text feature [hotspot] present in test data point [True]
```

```
486 Text feature [mouse] present in test data point [True]
487 Text feature [approximately] present in test data point [True]
488 Text feature [low] present in test data point [True]
489 Text feature [mice] present in test data point [True]
490 Text feature [myeloid] present in test data point [True]
491 Text feature [mm] present in test data point [True]
492 Text feature [taken] present in test data point [True]
493 Text feature [experimental] present in test data point [True]
494 Text feature [presence] present in test data point [True]
495 Text feature [splicing] present in test data point [True]
496 Text feature [regions] present in test data point [True]
497 Text feature [percentage] present in test data point [True]
499 Text feature [nucleotide] present in test data point [True]
Out of the top  500  features  49 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With 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/
# -------------------------------


# 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 : 1.2429213385472317
for n_estimators = 100 and max depth =  10
Log Loss : 1.2630834170939251
for n_estimators = 200 and max depth =  5
Log Loss : 1.2228233601329714
for n_estimators = 200 and max depth =  10
Log Loss : 1.2536116477998918
for n_estimators = 500 and max depth =  5
Log Loss : 1.2149007708854878
for n_estimators = 500 and max depth =  10
Log Loss : 1.2410844123224185
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2137016653694468
for n_estimators = 1000 and max depth =  10
Log Loss : 1.236825339764516
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2139861830937282
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2345558219512567
For values of best estimator =  1000 The train log loss is: 0.8616975262581904
For values of best estimator =  1000 The cross validation log loss is:
1.2137016653694468
For values of best estimator =  1000 The test log loss is: 1.166201560913679
```

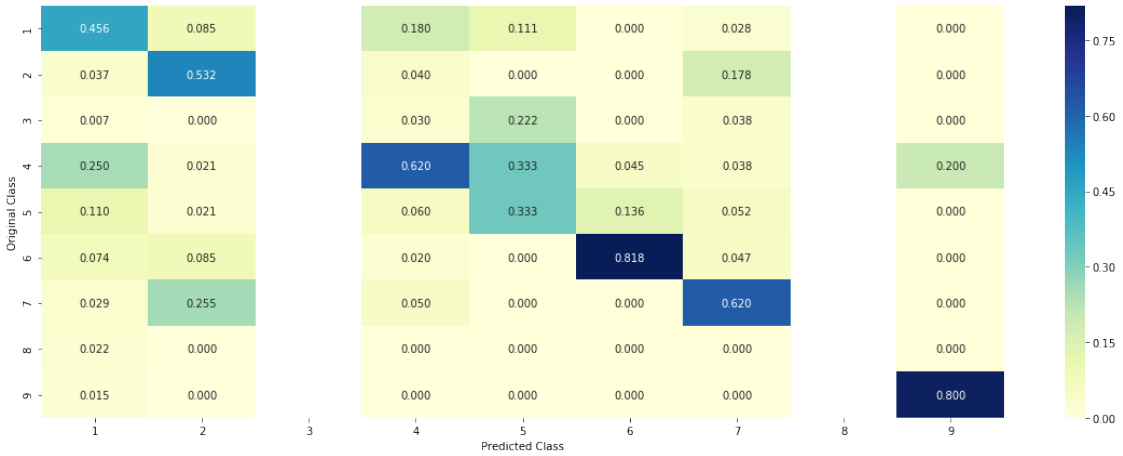
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

[79]:
```
# ------------------------------
# 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 : 1.2137016653694468
Number of mis-classified points : 0.424812030075188
-------------------- Confusion matrix --------------------
```


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



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

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

[80]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],␣
 ↪criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,␣
 ↪n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0872 0.0132 0.0454 0.7559 0.035  0.0284
0.0289 0.0029 0.003 ]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
8 Text feature [suppressor] present in test data point [True]
9 Text feature [loss] present in test data point [True]
12 Text feature [missense] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
18 Text feature [growth] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
20 Text feature [cells] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
23 Text feature [pathogenic] present in test data point [True]
25 Text feature [cell] present in test data point [True]
26 Text feature [functional] present in test data point [True]
28 Text feature [kinases] present in test data point [True]
30 Text feature [akt] present in test data point [True]
31 Text feature [pten] present in test data point [True]
33 Text feature [yeast] present in test data point [True]
34 Text feature [deleterious] present in test data point [True]
37 Text feature [protein] present in test data point [True]
40 Text feature [defective] present in test data point [True]
46 Text feature [functions] present in test data point [True]
47 Text feature [expression] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
55 Text feature [inhibition] present in test data point [True]
57 Text feature [patients] present in test data point [True]
62 Text feature [stability] present in test data point [True]
65 Text feature [downstream] present in test data point [True]
```

```
68 Text feature [57] present in test data point [True]
70 Text feature [phosphatase] present in test data point [True]
72 Text feature [predicted] present in test data point [True]
76 Text feature [conserved] present in test data point [True]
77 Text feature [phosphorylated] present in test data point [True]
78 Text feature [assays] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
82 Text feature [expected] present in test data point [True]
85 Text feature [assay] present in test data point [True]
86 Text feature [expressing] present in test data point [True]
87 Text feature [use] present in test data point [True]
89 Text feature [lines] present in test data point [True]
91 Text feature [inactivation] present in test data point [True]
92 Text feature [information] present in test data point [True]
93 Text feature [phospho] present in test data point [True]
96 Text feature [dna] present in test data point [True]
97 Text feature [p53] present in test data point [True]
99 Text feature [activity] present in test data point [True]
Out of the top  100  features  49 are present in query point
```

4.5.3.2. Inorrectly Classified point

[81]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].
 ↪iloc[test_point_index],test_df['Gene'].
 ↪iloc[test_point_index],test_df['Variation'].iloc[test_point_index],␣
 ↪no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4651 0.1202 0.0132 0.0791 0.0437 0.0382
0.1612 0.0115 0.0677]]
Actuall Class : 1
--------------------------------------------------
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
5 Text feature [function] present in test data point [True]
7 Text feature [constitutive] present in test data point [True]
8 Text feature [suppressor] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [activated] present in test data point [True]
```

```
11 Text feature [inhibitor] present in test data point [True]
12 Text feature [missense] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
16 Text feature [therapeutic] present in test data point [True]
17 Text feature [constitutively] present in test data point [True]
18 Text feature [growth] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
20 Text feature [cells] present in test data point [True]
21 Text feature [variants] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
23 Text feature [pathogenic] present in test data point [True]
24 Text feature [classified] present in test data point [True]
25 Text feature [cell] present in test data point [True]
26 Text feature [functional] present in test data point [True]
29 Text feature [receptor] present in test data point [True]
35 Text feature [drug] present in test data point [True]
37 Text feature [protein] present in test data point [True]
38 Text feature [months] present in test data point [True]
39 Text feature [treated] present in test data point [True]
40 Text feature [defective] present in test data point [True]
41 Text feature [trials] present in test data point [True]
42 Text feature [resistance] present in test data point [True]
46 Text feature [functions] present in test data point [True]
47 Text feature [expression] present in test data point [True]
48 Text feature [3t3] present in test data point [True]
52 Text feature [extracellular] present in test data point [True]
53 Text feature [transforming] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
55 Text feature [inhibition] present in test data point [True]
57 Text feature [patients] present in test data point [True]
58 Text feature [ovarian] present in test data point [True]
60 Text feature [repair] present in test data point [True]
61 Text feature [dose] present in test data point [True]
62 Text feature [stability] present in test data point [True]
65 Text feature [downstream] present in test data point [True]
67 Text feature [inhibited] present in test data point [True]
68 Text feature [57] present in test data point [True]
69 Text feature [activate] present in test data point [True]
71 Text feature [variant] present in test data point [True]
72 Text feature [predicted] present in test data point [True]
73 Text feature [null] present in test data point [True]
75 Text feature [survival] present in test data point [True]
76 Text feature [conserved] present in test data point [True]
78 Text feature [assays] present in test data point [True]
79 Text feature [proliferation] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
82 Text feature [expected] present in test data point [True]
```

```
84 Text feature [splice] present in test data point [True]
85 Text feature [assay] present in test data point [True]
86 Text feature [expressing] present in test data point [True]
87 Text feature [use] present in test data point [True]
88 Text feature [response] present in test data point [True]
89 Text feature [lines] present in test data point [True]
90 Text feature [efficacy] present in test data point [True]
91 Text feature [inactivation] present in test data point [True]
92 Text feature [information] present in test data point [True]
93 Text feature [phospho] present in test data point [True]
94 Text feature [stimulation] present in test data point [True]
95 Text feature [factor] present in test data point [True]
96 Text feature [dna] present in test data point [True]
97 Text feature [p53] present in test data point [True]
98 Text feature [risk] present in test data point [True]
99 Text feature [activity] present in test data point [True]
Out of the top  100  features  71 are present in query point
```

4.5.3. Hyper paramter tuning (With 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/
# --------------------------------


# 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 : 2.198623563152252
for n_estimators = 10 and max depth =  3
Log Loss : 1.8255993766972878
for n_estimators = 10 and max depth =  5
Log Loss : 1.6779828212897279
for n_estimators = 10 and max depth =  10
Log Loss : 2.3570006506736734
for n_estimators = 50 and max depth =  2
Log Loss : 1.7180278894683874
for n_estimators = 50 and max depth =  3
Log Loss : 1.4692336808767528
for n_estimators = 50 and max depth =  5
Log Loss : 1.499256822643227
for n_estimators = 50 and max depth =  10
Log Loss : 1.8341973976386283
for n_estimators = 100 and max depth =  2
Log Loss : 1.683115990879601
for n_estimators = 100 and max depth =  3
Log Loss : 1.4988201615139438
for n_estimators = 100 and max depth =  5
Log Loss : 1.4885166553361648
for n_estimators = 100 and max depth =  10
Log Loss : 1.856591002707938
for n_estimators = 200 and max depth =  2
Log Loss : 1.676980612002197
for n_estimators = 200 and max depth =  3
Log Loss : 1.5463169171247966
for n_estimators = 200 and max depth =  5
Log Loss : 1.5100784185910365
for n_estimators = 200 and max depth =  10
Log Loss : 1.8450785504875358
```

```
for n_estimators = 500 and max depth =  2
Log Loss : 1.730868334084814
for n_estimators = 500 and max depth =  3
Log Loss : 1.620353830196566
for n_estimators = 500 and max depth =  5
Log Loss : 1.5220988522730374
for n_estimators = 500 and max depth =  10
Log Loss : 1.8806855595639282
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6934416345031578
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6206173494154374
for n_estimators = 1000 and max depth =  5
Log Loss : 1.5105531984713638
for n_estimators = 1000 and max depth =  10
Log Loss : 1.9126475327636374
For values of best alpha =  50 The train log loss is: 0.1646172462655441
For values of best alpha =  50 The cross validation log loss is:
1.4692336808767528
For values of best alpha =  50 The test log loss is: 1.4439738118863732
```

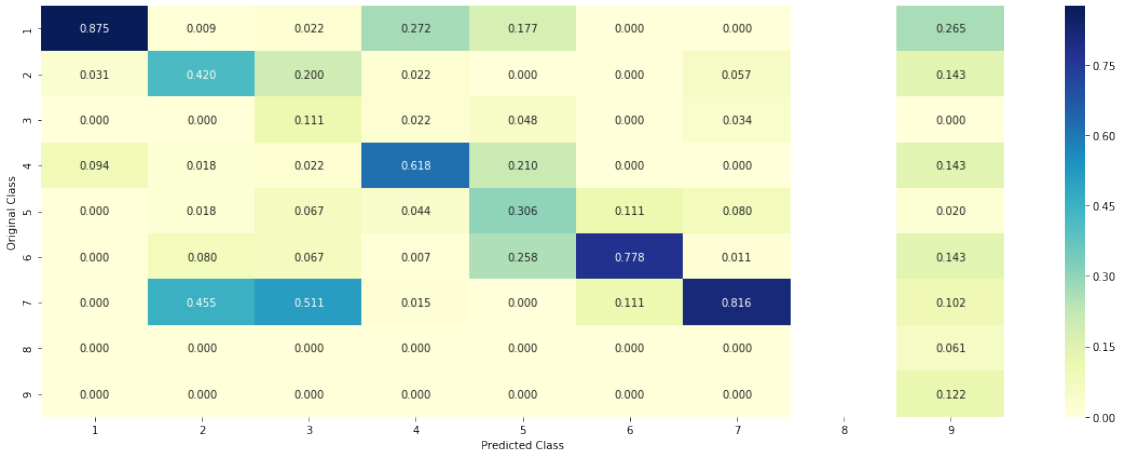
4.5.4. Testing model with best hyper parameters (Response Coding)

[83]:
```
# -------------------------------
# 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 : 1.4692336808767528
Number of mis-classified points : 0.4981203007518797
-------------------- Confusion matrix --------------------
```


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




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

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

[84]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)],␣
↪criterion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,␣
↪n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0152 0.0051 0.0358 0.8465 0.0218 0.0156
0.0015 0.0187 0.0399]]
```

```
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation 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
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

[85]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].
 ↪reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.
 ↪predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.61e-01 1.70e-03 4.20e-03 1.40e-02 7.00e-04
6.00e-03 7.00e-04 4.60e-03
  7.10e-03]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation 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
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

[86]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_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
```

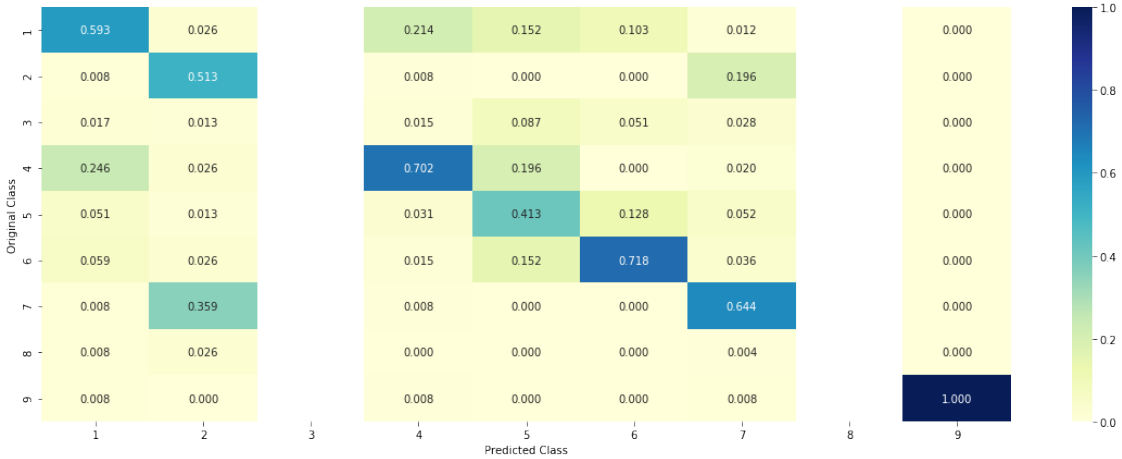
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.82
Naive Bayes : Log Loss: 1.21
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.511
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.207
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.422
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.864
```

4.7.2 testing the model with the best hyper parameters

[87]:
```
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 : 0.5481814766225364
Log loss (CV) on the stacking classifier : 1.206768541363904
Log loss (test) on the stacking classifier : 1.1571333133793873
Number of missclassified point : 0.37894736842105264
-------------------- Confusion matrix --------------------
```

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



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

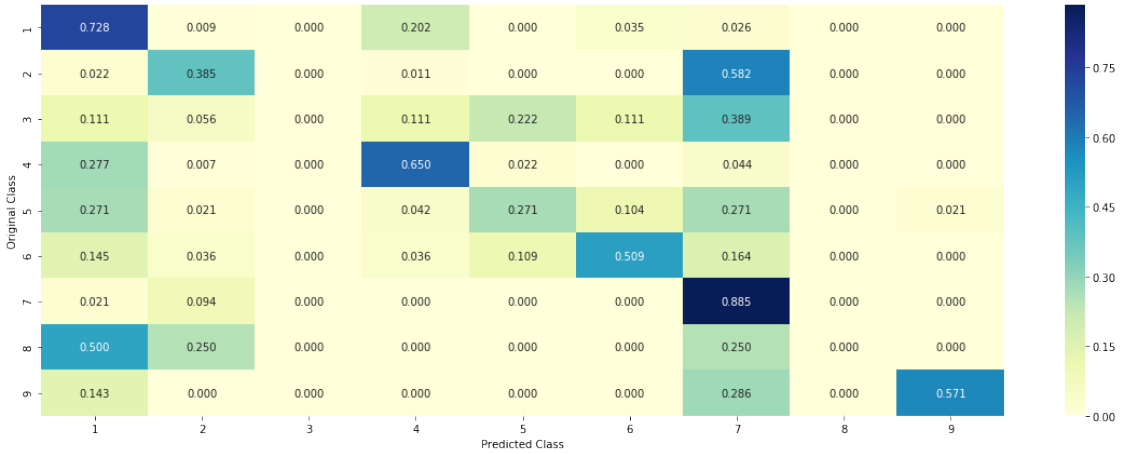
4.7.3 Maximum Voting classifier

[88]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.
 ↪VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf',␣
 ↪sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.
 ↪predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.
 ↪predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.
 ↪predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.
 ↪predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.
 ↪predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.8355531601579244
Log loss (CV) on the VotingClassifier : 1.234445815002799
Log loss (test) on the VotingClassifier : 1.1946995393970032
Number of missclassified point : 0.3669172932330827
-------------------- 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

Print Model Outputs with CountVectorizer i.e. BOW

[91]:
```
# Please compare all your models using Prettytable library
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
```

```
x = PrettyTable()
x.field_names = ["Model", "Test Loss", "No. of Misclassified Points in %"]
x.add_row(["Random Model", "2.5110096208457717", "NA"])
x.add_row(["Naive Bayes", "1.319385809714747", 0.41729323308270677*100])
x.add_row(["KNN", "1.1173900073859244", 0.37218045112781956*100])
x.add_row(["Logistic Regression", "1.1726560952891412", 0.
↪37593984962406013*100])
x.add_row(["Logistic Regression Without Class balancing", "1.185465209733062",␣
↪0.36278195488721804*100])
x.add_row(["Linear Support Vector Machines", "1.2149539168945986",  0.
↪39849624060150374*100])
x.add_row(["Random Forest Classifier (With One hot Encoding)", "1.
↪2100733367289231", 0.4041353383458647*100])
x.add_row(["Random Forest Classifier (With Response Coding)", "1.
↪3593960561354537", 0.47368421052631576*100])
x.add_row(["Stack the models", "1.1151389024773126", 0.3533834586466165*100])
x.add_row(["Maximum Voting classifier", "1.1963358723370534", 0.
↪3533834586466165*100])

print(x)
```

```
+-------------------------------------------------+--------------------+-------
--------------------------+
|                      Model                      |     Test Loss      | No. of
Misclassified Points in % |
+-------------------------------------------------+--------------------+-------
--------------------------+
|                  Random Model                   | 2.5110096208457717 |
NA                |
|                  Naive Bayes                    | 1.319385809714747  |
41.72932330827068         |
|                     KNN                         | 1.1173900073859244 |
37.21804511278196         |
|              Logistic Regression                | 1.1726560952891412 |
37.59398496240601         |
|   Logistic Regression Without Class balancing   | 1.185465209733062  |
36.278195488721806        |
|          Linear Support Vector Machines         | 1.2149539168945986 |
39.849624060150376        |
| Random Forest Classifier (With One hot Encoding) | 1.2100733367289231 |
40.41353383458647         |
| Random Forest Classifier (With Response Coding)  | 1.3593960561354537 |
47.368421052631575        |
|                Stack the models                 | 1.1151389024773126 |
```

```
35.338345864661655        |
|            Maximum Voting classifier             | 1.1963358723370534 |
35.338345864661655        |
+--------------------------------------------------+--------------------+-------
--------------------------+
```

Print Model Outputs with TfidfVectorizer

[92]:
```
x = PrettyTable()
x.field_names = ["Model", "Test Loss", "No. of Misclassified Points in %"]
x.add_row(["Random Model", "2.4891260853746404", "NA"])
x.add_row(["Naive Bayes", "1.3039713635684613", 0.4266917293233083*100])
x.add_row(["KNN", "1.0817519098560806", 0.3890977443609023*100])
x.add_row(["Logistic Regression", "1.162517109586788", 0.3609022556390977*100])
x.add_row(["Logistic Regression Without Class balancing", "1.1850832332691803",␣
 ↪0.36466165413533835*100])
x.add_row(["Linear Support Vector Machines", "1.2169634866263805",  0.
 ↪39097744360902253*100])
x.add_row(["Random Forest Classifier (With One hot Encoding)", "1.
 ↪1919793594773513", 0.40037593984962405*100])
x.add_row(["Random Forest Classifier (With Response Coding)", "1.
 ↪3026489025851031", 0.4774436090225564*100])
x.add_row(["Stack the models", "1.0515191193988829", 0.3458646616541353*100])
x.add_row(["Maximum Voting classifier", "1.0845590332286914", 0.
 ↪347368421052631 6*100])

print(x)
```

```
+--------------------------------------------------+--------------------+-------
--------------------------+
|                      Model                       |     Test Loss      | No. of
Misclassified Points in % |
+--------------------------------------------------+--------------------+-------
--------------------------+
|                  Random Model                    | 2.4891260853746404 |
NA                |
|                  Naive Bayes                     | 1.3039713635684613 |
42.66917293233083         |
|                      KNN                         | 1.0817519098560806 |
38.90977443609023         |
|              Logistic Regression                 | 1.162517109586788  |
36.09022556390977         |
|   Logistic Regression Without Class balancing    | 1.1850832332691803 |
36.46616541353384         |
|          Linear Support Vector Machines          | 1.2169634866263805 |
39.097744360902254        |
| Random Forest Classifier (With One hot Encoding) | 1.1919793594773513 |
40.037593984962406        |
```

```
| Random Forest Classifier (With Response Coding)  | 1.3026489025851031 |
47.744360902255636        |
|                  Stack the models                | 1.0515191193988829 |
34.58646616541353         |
|            Maximum Voting classifier             | 1.0845590332286914 |
34.73684210526316         |
+--------------------------------------------------+--------------------+-------
--------------------------+
```

Print Model Outputs with TfidfVectorizer With Top 1000 Features

[93]:
```
x = PrettyTable()
x.field_names = ["Model", "Test Loss", "No. of Misclassified Points in %"]
x.add_row(["Random Model", "2.4891260853746404", "NA"])
x.add_row(["Naive Bayes", "1.2066017622733376", 0.3984962406015037*100])
x.add_row(["KNN", "1.081876583678065", 0.39285714285714285*100])
x.add_row(["Logistic Regression", "1.0731158684634137", 0.
 ↪39097744360902253*100])
x.add_row(["Logistic Regression Without Class balancing", "1.0944256930516638",␣
 ↪0.38345864661654133*100])
x.add_row(["Linear Support Vector Machines", "1.0546829663555854",  0.
 ↪3684210526315789*100])
x.add_row(["Random Forest Classifier (With One hot Encoding)", "1.
 ↪2137016653694468", 0.424812030075188*100])
x.add_row(["Random Forest Classifier (With Response Coding)", "1.
 ↪4692336808767528", 0.4981203007518797*100])
x.add_row(["Stack the models", "1.1571333133793873", 0.37894736842105264*100])
x.add_row(["Maximum Voting classifier", "1.1946995393970032", 0.
 ↪3669172932330827*100])

print(x)
```

```
+--------------------------------------------------+--------------------+-------
--------------------------+
|                      Model                       |     Test Loss      | No. of
Misclassified Points in % |
+--------------------------------------------------+--------------------+-------
--------------------------+
|                  Random Model                    | 2.4891260853746404 |
NA                |
|                  Naive Bayes                     | 1.2066017622733376 |
39.849624060150376        |
|                     KNN                          | 1.081876583678065  |
39.285714285714285        |
|              Logistic Regression                 | 1.0731158684634137 |
39.097744360902254        |
|   Logistic Regression Without Class balancing    | 1.0944256930516638 |
38.34586466165413         |
```

```
|          Linear Support Vector Machines          | 1.0546829663555854 |
36.84210526315789         |
| Random Forest Classifier (With One hot Encoding) | 1.2137016653694468 |
42.4812030075188         |
| Random Forest Classifier (With Response Coding)  | 1.4692336808767528 |
49.81203007518797         |
|                 Stack the models                 | 1.1571333133793873 |
37.89473684210527         |
|            Maximum Voting classifier             | 1.1946995393970032 |
36.69172932330827         |
+--------------------------------------------------+--------------------+-------
--------------------------+
```

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

[97]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding = gene_vectorizer.
 ↪fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

print(train_gene_feature_onehotCoding.shape)
print(test_gene_feature_onehotCoding.shape)
print(cv_gene_feature_onehotCoding.shape)
```

```
(2124, 228)
(665, 228)
(532, 228)
```

[98]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding = variation_vectorizer.
 ↪fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.
 ↪transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.
 ↪transform(cv_df['Variation'])

print(train_variation_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
(2124, 2068)
(665, 2068)
(532, 2068)
```

[108]:
```
# building a CountVectorizer with all the words that occured minimum 10 times␣
 ↪in train data
text_vectorizer = CountVectorizer(min_df=10, ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.
 ↪fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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

# 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)

print(train_text_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
Total number of unique words in train data : 237463
(2124, 237463)
(665, 2068)
(532, 2068)
```

[109]:
```
# merge all features
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']))
```

[110]:
```
print(train_x_onehotCoding.shape, train_y.shape)
print(test_x_onehotCoding.shape, test_y.shape)
print(cv_x_onehotCoding.shape, cv_y.shape)
```

```
(2124, 239759) (2124,)
(665, 239759) (665,)
(532, 239759) (532,)
```

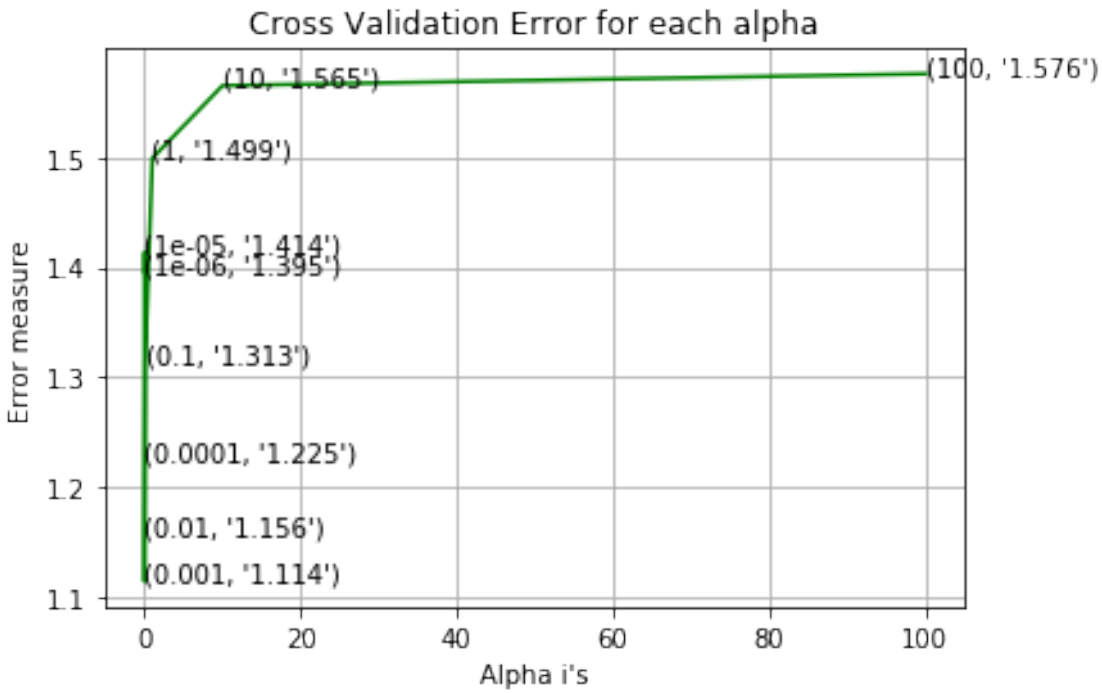
- Hyperparameter Tuning

[111]:
```
# 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 : 1.3954379234410594
for alpha = 1e-05
Log Loss : 1.4137279174772743
for alpha = 0.0001
Log Loss : 1.225237654288811
for alpha = 0.001
Log Loss : 1.1141721798290058
for alpha = 0.01
Log Loss : 1.155683818057866
for alpha = 0.1
Log Loss : 1.3130333664283742
for alpha = 1
Log Loss : 1.4991068801834122
for alpha = 10
Log Loss : 1.565204687262526
for alpha = 100
Log Loss : 1.575916192328653
```



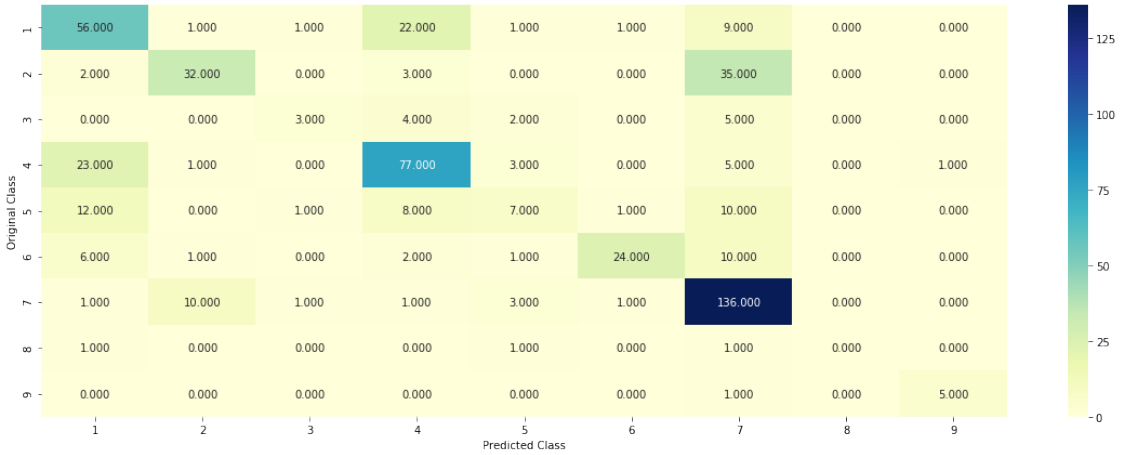

```
For values of best alpha =  0.001 The train log loss is: 0.6647529134085501
For values of best alpha =  0.001 The cross validation log loss is:
1.1141721798290058
For values of best alpha =  0.001 The test log loss is: 1.1888666290504575
```

- Train the model using best hyperparameter

[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/
#------------------------------
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 : 1.1141721798290058
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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

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

[113]:
```
x = PrettyTable()
x.field_names = ["Model", "Test Loss", "No. of Misclassified Points in %"]
x.add_row(["Logistic Regression", 1.1141721798290058, 0.3609022556390977*100])

print(x)
```

```
+---------------------+--------------------+----------------------------------+
|        Model        |     Test Loss      | No. of Misclassified Points in % |
+---------------------+--------------------+----------------------------------+
| Logistic Regression | 1.1141721798290058 |        36.09022556390977         |
+---------------------+--------------------+----------------------------------+
```

- Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0
- LOGISTIC REGRESSION(1–4 GRAMS) TOP 2000 TF-IDF FEATURES From Text

[120]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(ngram_range=(1,4))
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'])

print(train_gene_feature_onehotCoding.shape)
print(test_gene_feature_onehotCoding.shape)
print(cv_gene_feature_onehotCoding.shape)
```

```
(2124, 228)
(665, 228)
(532, 228)
```

[121]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(ngram_range=(1,4))
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'])

print(train_variation_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
(2124, 2070)
(665, 2070)
(532, 2070)
```

[122]:
```
# building a TfidfVectorizer with all the words that occured minimum 3 times in␣
 ↪train data
text_vectorizer = TfidfVectorizer(min_df=3, ngram_range=(1,4),␣
 ↪max_features=2000)
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))

# 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)

print(train_text_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
Total number of unique words in train data : 2000
(2124, 2000)
(665, 2070)
(532, 2070)
```

[123]:
```
# merge all features
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']))
```

[124]:
```
print(train_x_onehotCoding.shape, train_y.shape)
print(test_x_onehotCoding.shape, test_y.shape)
print(cv_x_onehotCoding.shape, cv_y.shape)
```

```
(2124, 4298) (2124,)
(665, 4298) (665,)
(532, 4298) (532,)
```

- Hyper Parameter Tuning

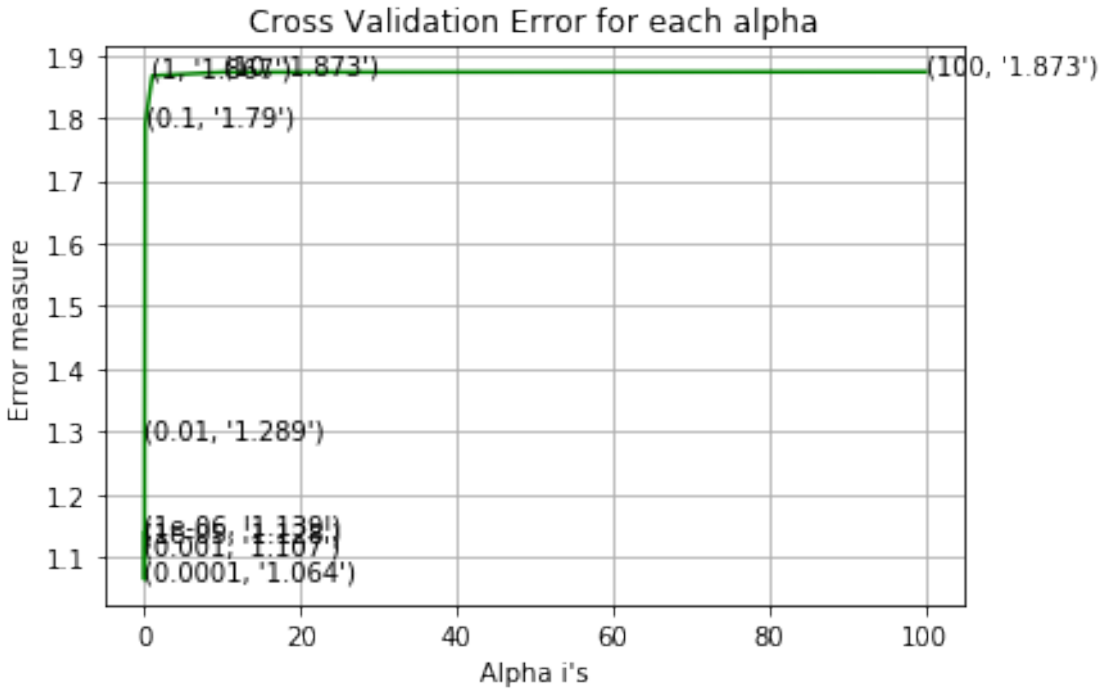
[125]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
```

```
# ------------------------------
# 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 : 1.1392519002582724
for alpha = 1e-05
Log Loss : 1.127625901151244
for alpha = 0.0001
Log Loss : 1.0644472105290257
for alpha = 0.001
Log Loss : 1.107061821334901
for alpha = 0.01
Log Loss : 1.2888635206425618
for alpha = 0.1
Log Loss : 1.7897330972982275
for alpha = 1
```

```
Log Loss : 1.8672168974896006
for alpha = 10
Log Loss : 1.8727858756047937
for alpha = 100
Log Loss : 1.8733966028453002
```

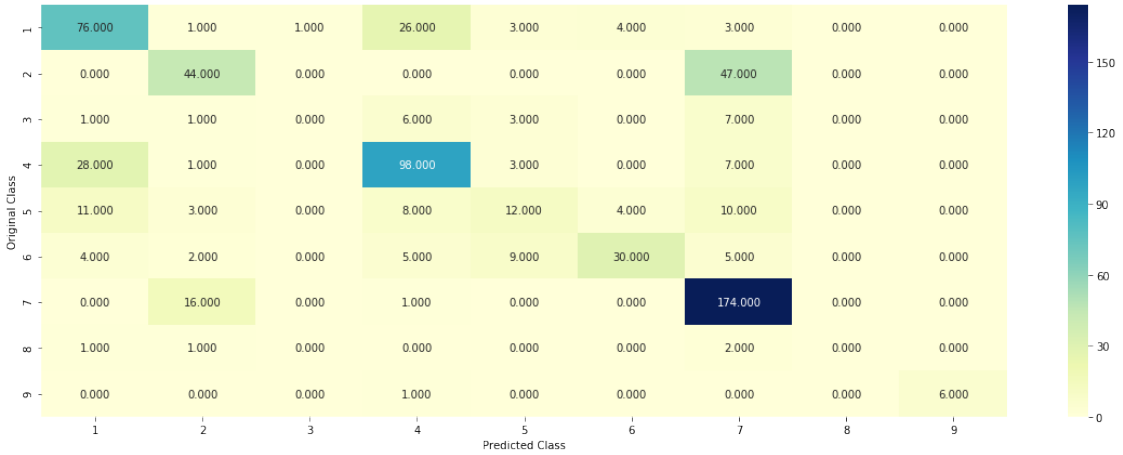



```
For values of best alpha =  0.0001 The train log loss is: 0.4369420110214827
For values of best alpha =  0.0001 The cross validation log loss is:
1.0644472105290257
For values of best alpha =  0.0001 The test log loss is: 0.9887923967638099
```

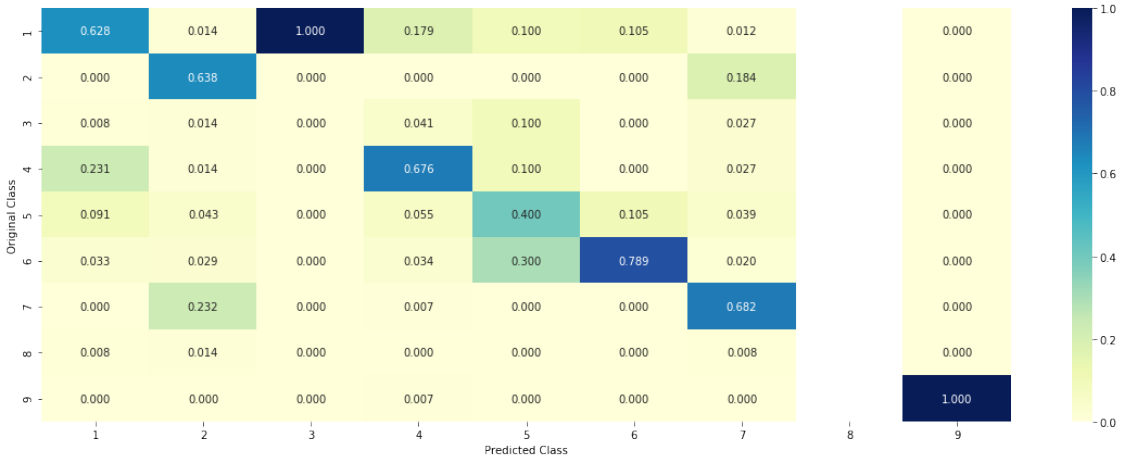
[128]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/
 ↪generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15,␣
 ↪fit_intercept=True, max_iter=None, 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,␣
 ↪test_x_onehotCoding, test_y, clf)
```

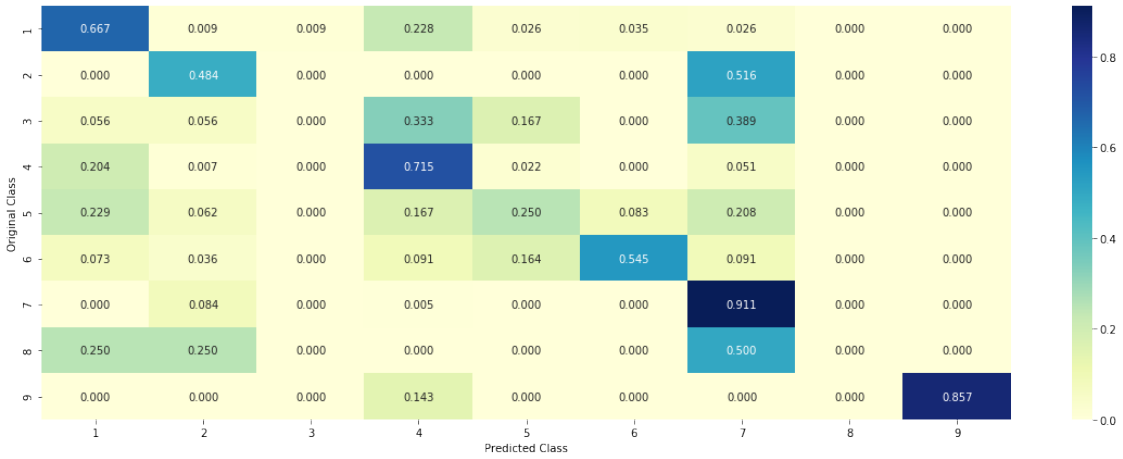
```
Log loss : 0.9887923967638099
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```




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



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



[129]:
```
x = PrettyTable()
x.field_names = ["Model", "Test Loss", "No. of Misclassified Points in %"]
x.add_row(["Logistic Regression", 0.9887923967638099, 0.3383458646616541*100])

print(x)
```

```
+---------------------+---------------------+----------------------------------+
|        Model        |     Test Loss       | No. of Misclassified Points in % |
+---------------------+---------------------+----------------------------------+
| Logistic Regression | 0.9887923967638099 |        33.83458646616541         |
+---------------------+---------------------+----------------------------------+
```

[ ]:
```

```