

Personalized cancer diagnosis

Here I am using TfidfVectorizer and I am Taking all the features over here...

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=qxXRKVompl8>

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 contains 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 have a common column called ID
- Data file's information:
 - training_variants (ID , Gene, Variations, Class)
 - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

ID, Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying

STAR syndrome. Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning 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 probabilities => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
```

```

from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression

```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```

In [2]: data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()

```

```

Number of data points : 3321
Number of features : 4
Features : ['ID' 'Gene' 'Variation' 'Class']

```

Out[2]:

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

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

Fields are

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

3.1.2. Reading Text Data

```
In [3]: # note the separator in this file
data_text = pd.read_csv("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']
```

Out[3]:

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

3.1.3. Preprocessing of text

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

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

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

        data_text[column][index] = string

In [5]: #text processing stage.
start_time = time.clock()
```



```

for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "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 : 167.73038498468722 seconds

```

```

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

```

Out[6]:

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

```

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

```

Out[7]:

	ID	Gene	Variation	Class	TEXT
--	----	------	-----------	-------	------

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

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

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

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

# split the data into test and train by maintaining same distribution of
# output variable '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 variable 'y_train' [stratify=y_train]
```

```
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

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

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

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

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

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

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of  $y_i$  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()

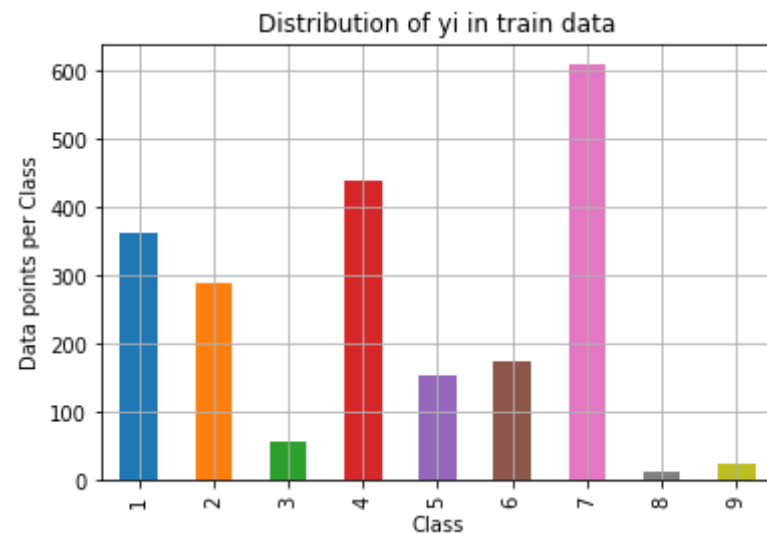
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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

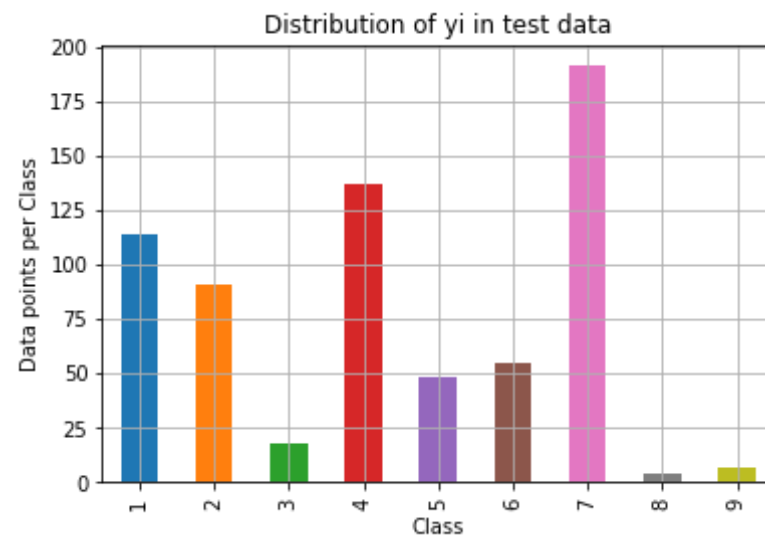
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:

```

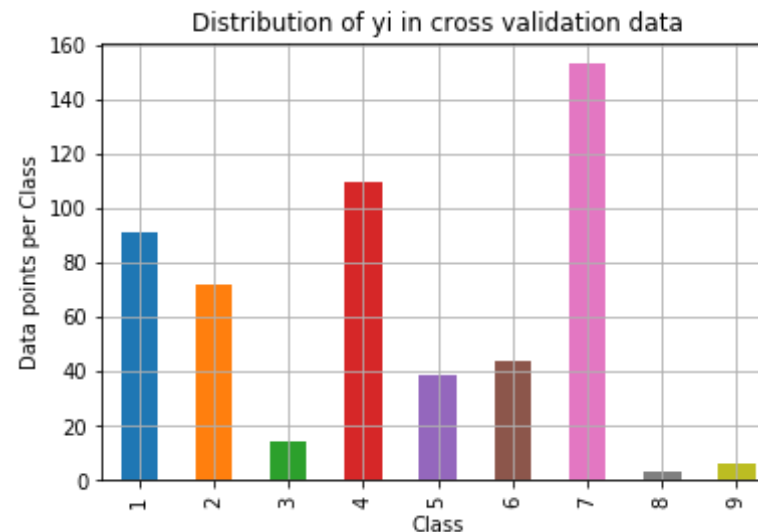
```
print('Number of data points in class', i+1, ': ', cv_class_distribution.values[i], ' (', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```



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 probabilities randomly such that they sum to 1.

```
In [13]: # This function plots the confusion matrices given y_i, y_i_hat.  
def plot_confusion_matrix(test_y, predict_y):  
    C = confusion_matrix(test_y, predict_y)
```

```

# C = 9,9 matrix, each cell (i,j) represents number of points of class i 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 corresponds to columns and axis=1 corresponds to rows in two dimensional array
# C.sum(axis=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 corresponds to columns and axis=1 corresponds to rows in two dimensional array
# C.sum(axis=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 (Column Sum=1)", "-"*20)
plt.figure(figsize=(20,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

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

```

```

In [14]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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))

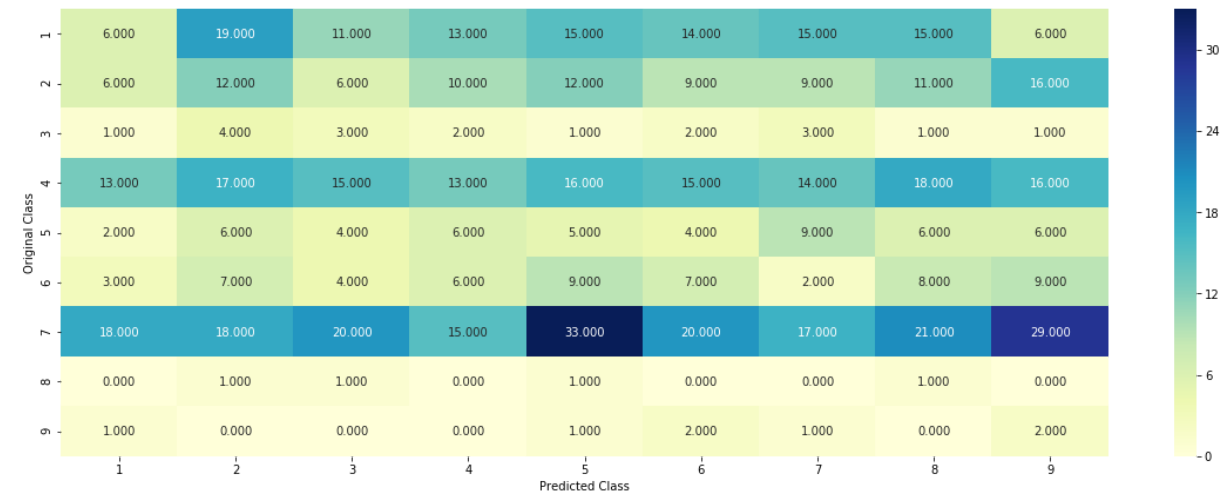
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

Log loss on Cross Validation Data using Random Model 2.3850758085115262

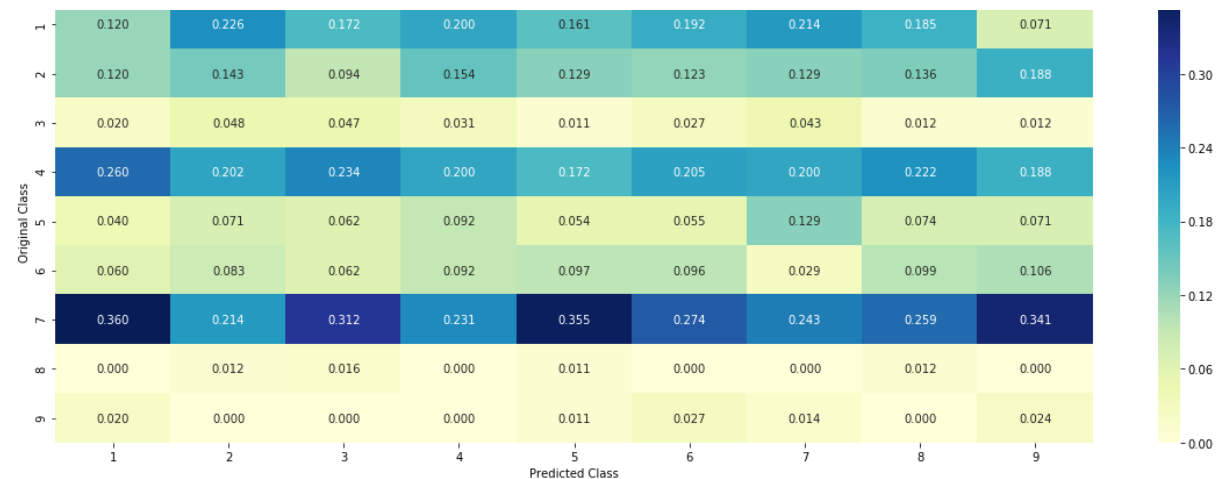
Log loss on Test Data using Random Model 2.5014478055751113

----- Confusion matrix -----

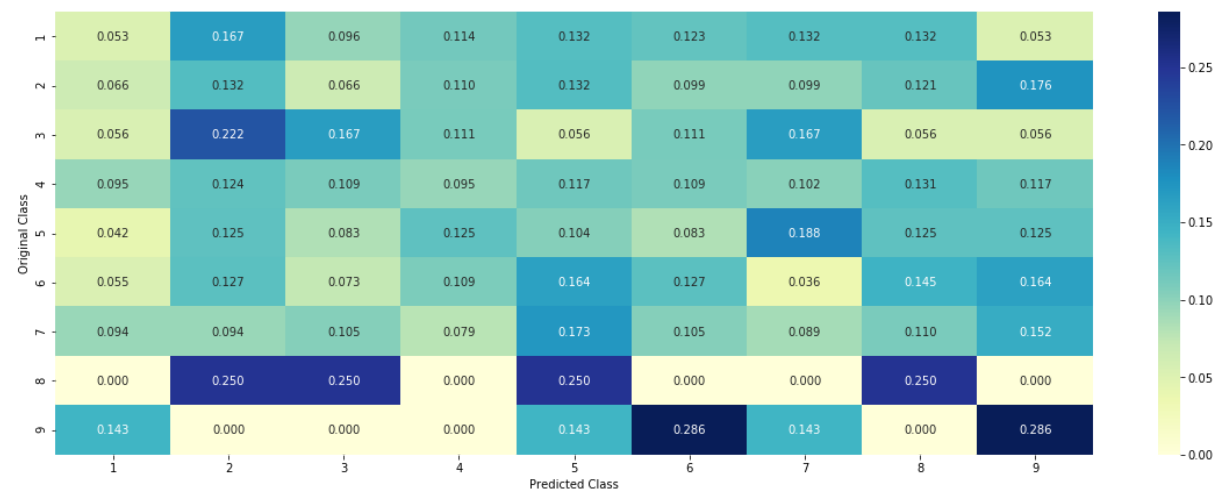


----- Precision matrix (Column Sum=1) -----

--



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



3.3 Univariate Analysis

```
In [15]: # code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
```

```

# algorithm
# -----
# Consider all unique values and the number of occurrences of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occurred 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 variation 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 occurred in whole data
for i, denominator in value_count.items():
    # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
    # vec is 9 dimensional 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 occurred 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 a
s value
        gv_dict[i]=vec
        return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #      {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
0.068181818181818177, 0.068181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
60606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.09333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666
6],
    #      ...
    #      }

```

```

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

- $(\text{numerator} + 10 \cdot \alpha) / (\text{denominator} + 90 \cdot \alpha)$

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

```

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

```

Number of Unique Genes : 233

BRCA1 170

TP53 106

EGFR 95

PTEN 83

BRCA2 82

KIT 62

BRAF 53

ALK 47

ERBB2 45

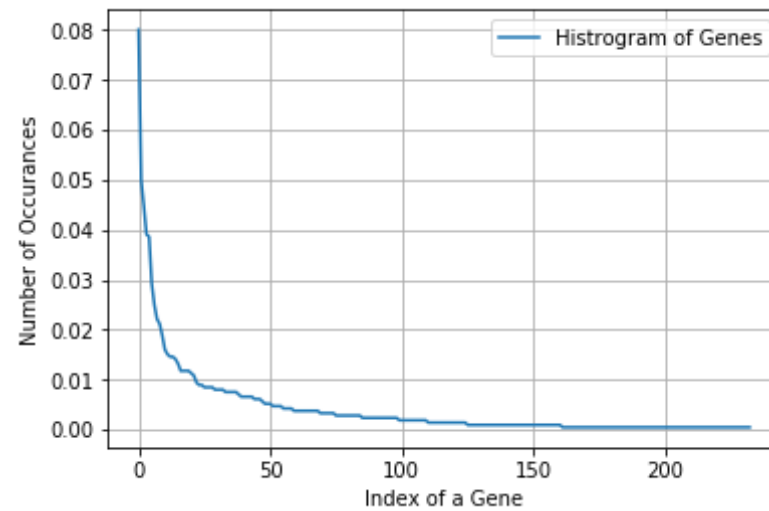
PDGFRA 40

Name: Gene, dtype: int64

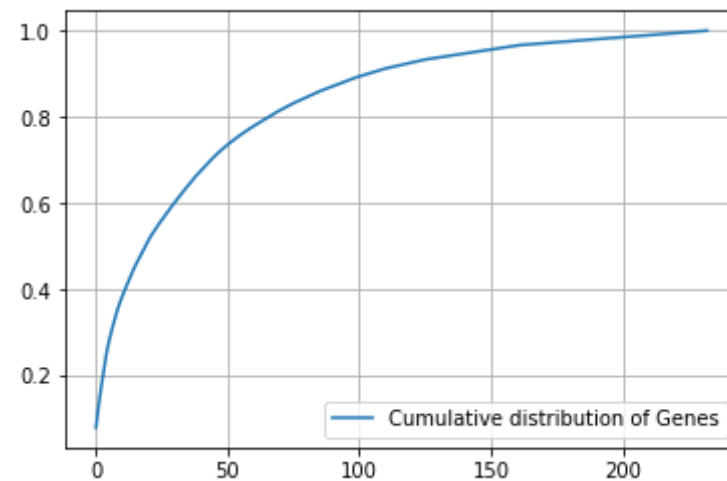
```
In [17]: print("Ans: There are", unique_genes.shape[0] , "different categories of  
genes in the train data, and they are distributed as follows",)
```

Ans: There are 233 different categories of genes in the train data, and they are distributed as follows

```
In [18]: s = sum(unique_genes.values);  
h = unique_genes.values/s;  
plt.plot(h, label="Histogram of Genes")  
plt.xlabel('Index of a Gene')  
plt.ylabel('Number of Occurances')  
plt.legend()  
plt.grid()  
plt.show()
```

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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

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

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

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

```
In [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'])
```

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

```
Out[23]: 340      ROS1
          2215     PTEN
          2797    BRCA2
          2125    CCND1
          2844    BRCA2
          Name: Gene, dtype: object
```

```
In [24]: 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, 232)
```

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

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

```
In [25]: alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
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_))
```

```

ses_, 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 vali
dation 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 l
oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

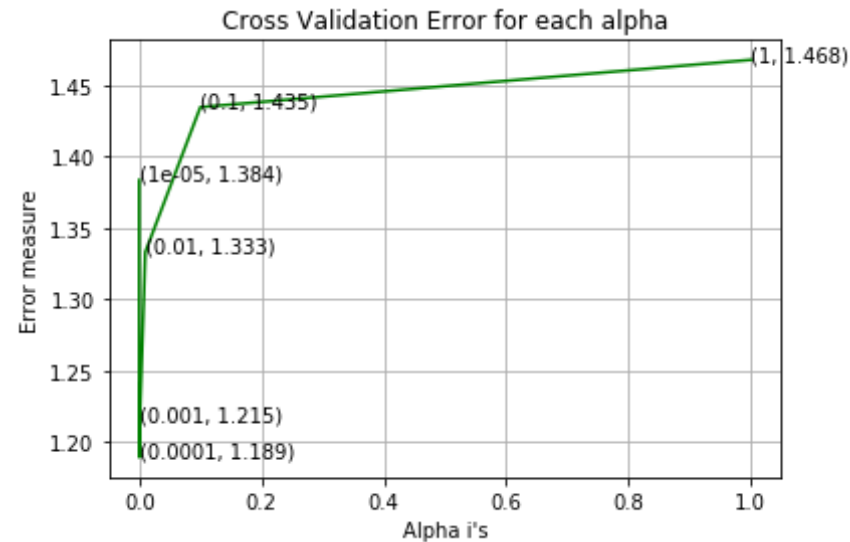
```

For values of alpha = 1e-05 The log loss is: 1.3835840603199143
For values of alpha = 0.0001 The log loss is: 1.1890541730465527
For values of alpha = 0.001 The log loss is: 1.215100849911499
For values of alpha = 0.01 The log loss is: 1.333041178640383

```

For values of alpha = 0.1 The log loss is: 1.4347509686310185

For values of alpha = 1 The log loss is: 1.4679768523947858



For values of best alpha = 0.0001 The train log loss is: 1.0361212724912523

For values of best alpha = 0.0001 The cross validation log loss is: 1.1890541730465527

For values of best alpha = 0.0001 The test log loss is: 1.2218021580012324

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

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

```
In [26]: 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']  
)))]
```

```
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 23 genes in train dataset?

Ans

1. In test data 641 out of 665 : 96.39097744360903

2. In cross validation data 518 out of 532 : 97.36842105263158

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

```
In [27]: unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occurred most
print(unique_variations.head(10))
```

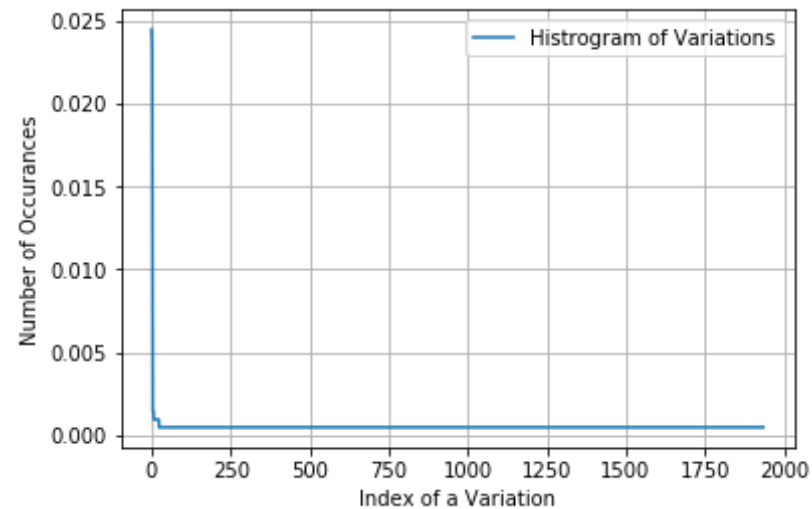
```
Number of Unique Variations : 1935
Truncating_Mutations      52
Amplification              51
Deletion                  49
Fusions                   18
T58I                      3
E17K                      3
Overexpression            3
Q61K                      2
Promoter_Hypermethylation 2
```

```
TMPRSS2-ETV1_Fusion      2
Name: Variation, dtype: int64
```

```
In [28]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the train data, and they are distributed as follows",)
```

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

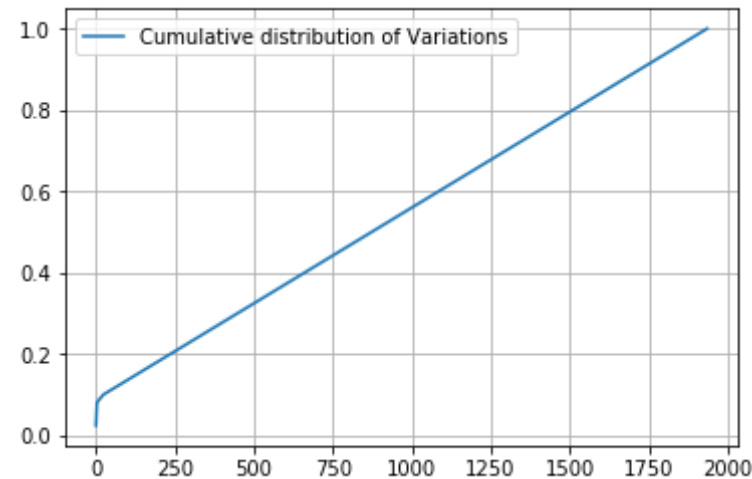
```
In [29]: s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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

```
plt.grid()
plt.legend()
plt.show()
```

```
[0.02448211 0.04849341 0.07156309 ... 0.99905838 0.99952919 1. ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

```
In [31]: # 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, "V
ariation", cv_df))
```

In [32]: `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)

In [33]: `# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])`

In [34]: `print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)`

train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 1966)

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

Let's build a model just like the earlier!

In [35]: `alpha = [10 ** x for x in range(-5, 1)]
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()

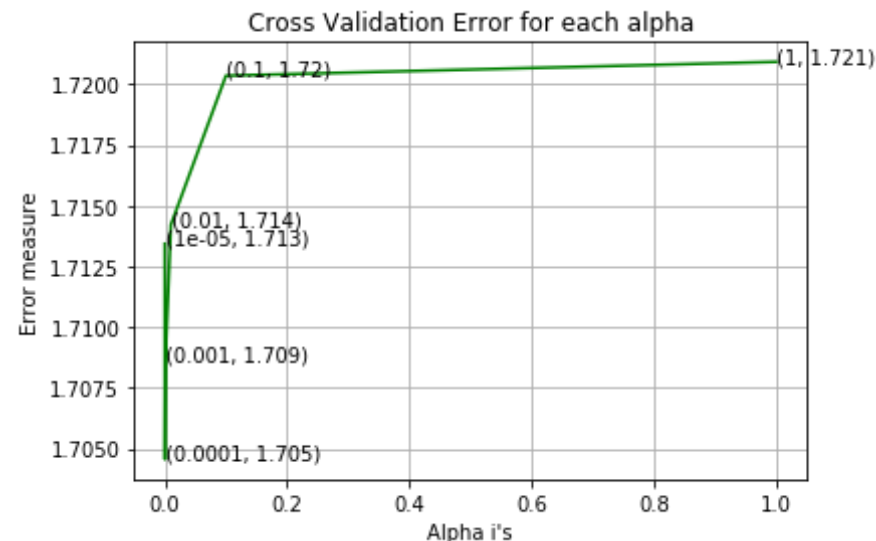
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

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: 1.713434599654619
 For values of alpha = 0.0001 The log loss is: 1.70455221322481
 For values of alpha = 0.001 The log loss is: 1.7086316651890452
 For values of alpha = 0.01 The log loss is: 1.7142172849479687
 For values of alpha = 0.1 The log loss is: 1.7203595277898927
 For values of alpha = 1 The log loss is: 1.7209306526893509



For values of best alpha = 0.0001 The train log loss is: 0.7532741774207281
 For values of best alpha = 0.0001 The cross validation log loss is: 1.70455221322481
 For values of best alpha = 0.0001 The test log loss is: 1.69678582120871

Q11. Is the Variation feature stable across all the data sets (Test, Train,

Cross validation)?

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

```
In [36]: 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'])))]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))]
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 1935 genes in test and cross validation data sets?

Ans

1. In test data 79 out of 665 : 11.879699248120302
2. In cross validation data 51 out of 532 : 9.586466165413533

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 predicting y_i ?
5. Is the text feature stable across train, test and CV datasets?

```
In [37]: def extract_dictionary_paddle(cls_text):
dictionary = defaultdict(int)
for index, row in cls_text.iterrows():
    for word in row['TEXT'].split():
        dictionary[word] +=1
return dictionary
```

```
In [38]: import math
#https://stackoverflow.com/a/1602964
def get_text_responseCoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10)/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and
# returns (1*number of 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 occurred
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 : 53130

```
In [41]: dict_list = []
# dict_list =[] contains 9 dictionaries 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 build on whole training text data
total_dict = extract_dictionary_paddle(train_df)

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

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

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

```
In [44]: # don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
```

```

ding, axis=0)

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

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

```

In [45]: [#https://stackoverflow.com/a/2258273/4084039](https://stackoverflow.com/a/2258273/4084039)

```

sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
    x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))

```

In [47]:

```

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]))

```

```

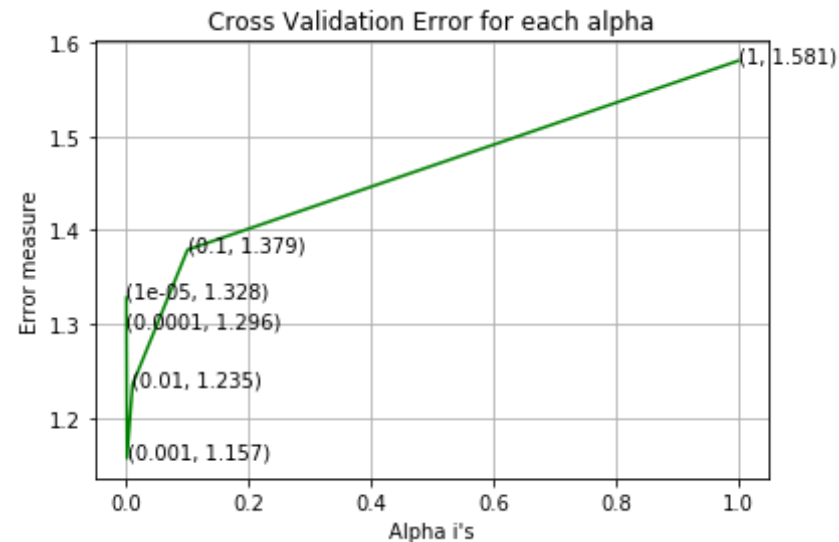
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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 vali
dation 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 l
oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

For values of alpha = 1e-05 The log loss is: 1.3278872369174244
For values of alpha = 0.0001 The log loss is: 1.2958957960991966
For values of alpha = 0.001 The log loss is: 1.1565873526299628
For values of alpha = 0.01 The log loss is: 1.2352400401776273
For values of alpha = 0.1 The log loss is: 1.3791577430189437
For values of alpha = 1 The log loss is: 1.580707605359497

```

For values of best alpha = 0.001 The train log loss is: 0.6457603750092045

For values of best alpha = 0.001 The cross validation log loss is: 1.1565873526299628

For values of best alpha = 0.001 The test log loss is: 1.1655756385011267

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

Ans. Yes, it seems like!

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

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

```
len2 = len(set(train_text_features) & set(df_text_features))
return len1, len2
```

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

95.905 % of word of test data appeared in train data
97.961 % of word of Cross Validation appeared in train data

4. Machine Learning Models

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

#Misc. functionns for ML models

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

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

```
In [51]: def report_log_loss(train_x, train_y, test_x, test_y, clf):
    clf.fit(train_x, train_y)
```

```

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x)
return log_loss(test_y, sig_clf_probs, eps=1e-15)

```

```

In [53]: 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)

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 p
oint [{}]".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

```
In [54]: 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))
```

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

```
One hot encoding features :
(number of data points * number of features) in train data = (2124, 55328)
(number of data points * number of features) in test data = (665, 55328)
(number of data points * number of features) in cross validation data = (532, 55328)
```

```
In [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

```
In [57]: 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 probabilitites 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_a
rray[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")
```

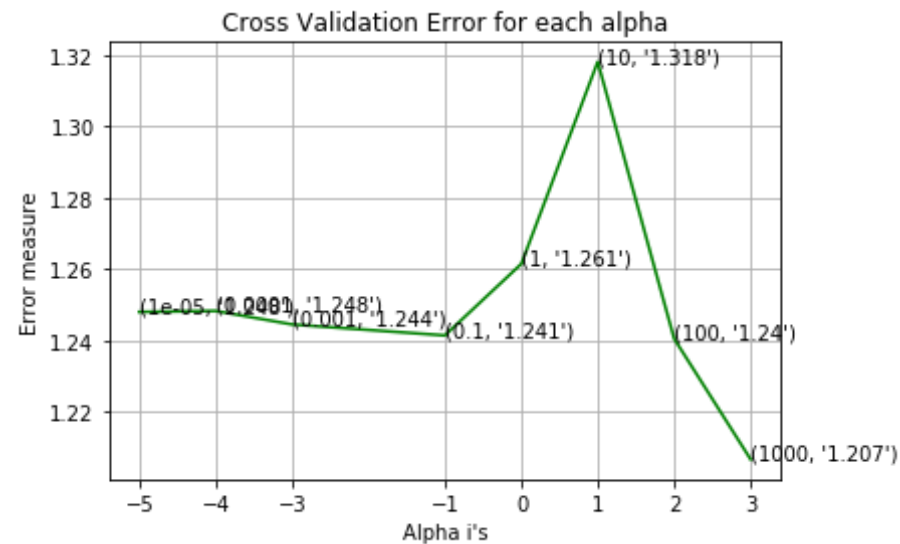
```

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 vali
      dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
      =1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
      oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

for alpha = 1e-05
Log Loss : 1.2479875256163828
for alpha = 0.0001
Log Loss : 1.2482659152111826
for alpha = 0.001
Log Loss : 1.2444814188967428
for alpha = 0.1
Log Loss : 1.2413973993604515
for alpha = 1
Log Loss : 1.261464069228379
for alpha = 10
Log Loss : 1.3181270094324165
for alpha = 100
Log Loss : 1.240474809788655
for alpha = 1000
Log Loss : 1.2066535418295192

```



For values of best alpha = 1000 The train log loss is: 0.9152796498061373

For values of best alpha = 1000 The cross validation log loss is: 1.2066535418295192

For values of best alpha = 1000 The test log loss is: 1.240747096113884

4.1.1.2. Testing the model with best hyper parameters

```
In [58]: 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 probabilities 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]))
```

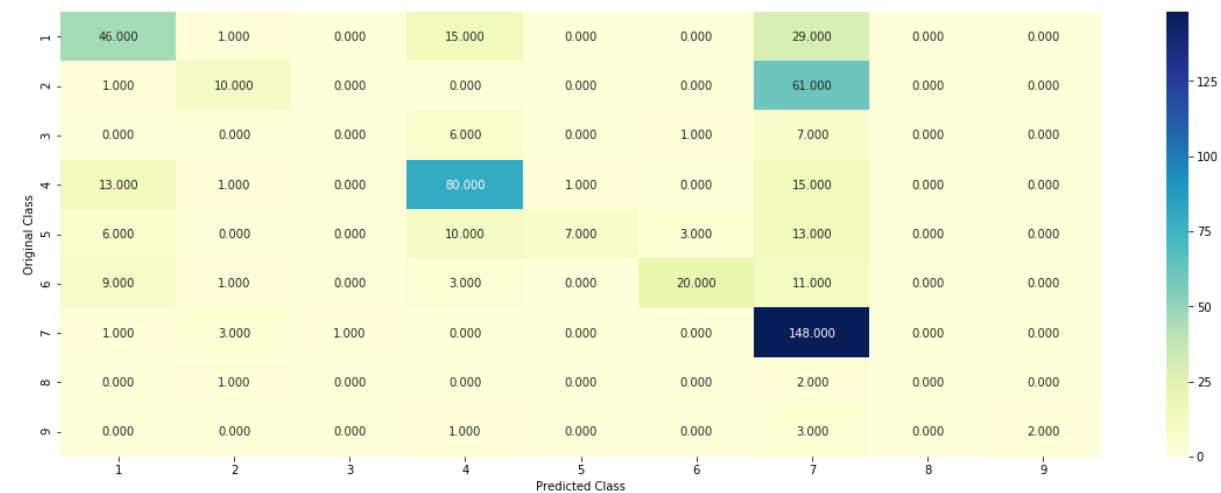


```
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))
```

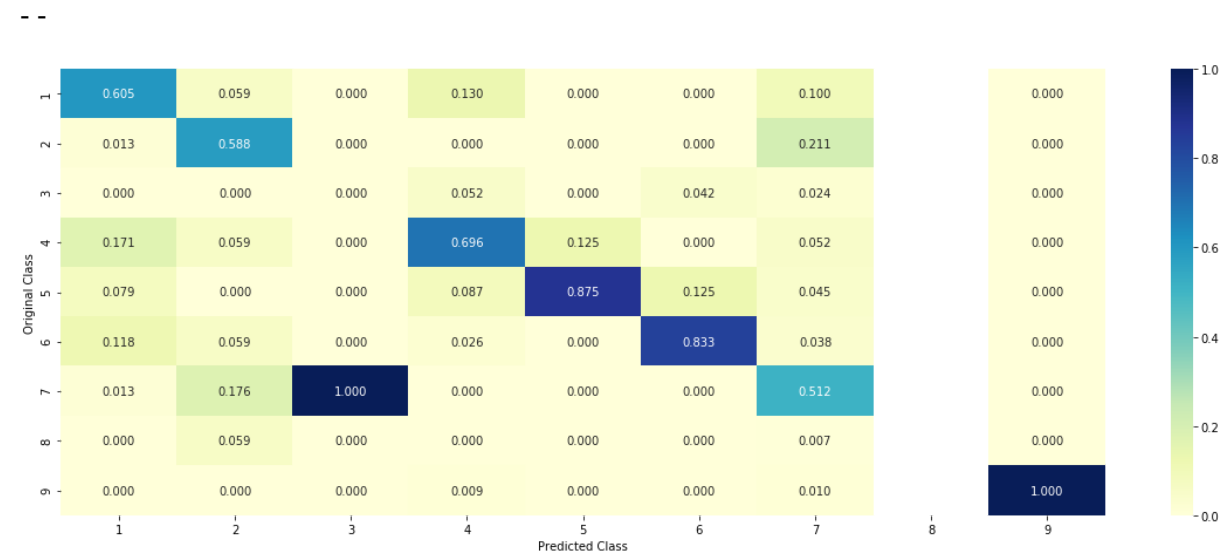
Log Loss : 1.2066535418295192

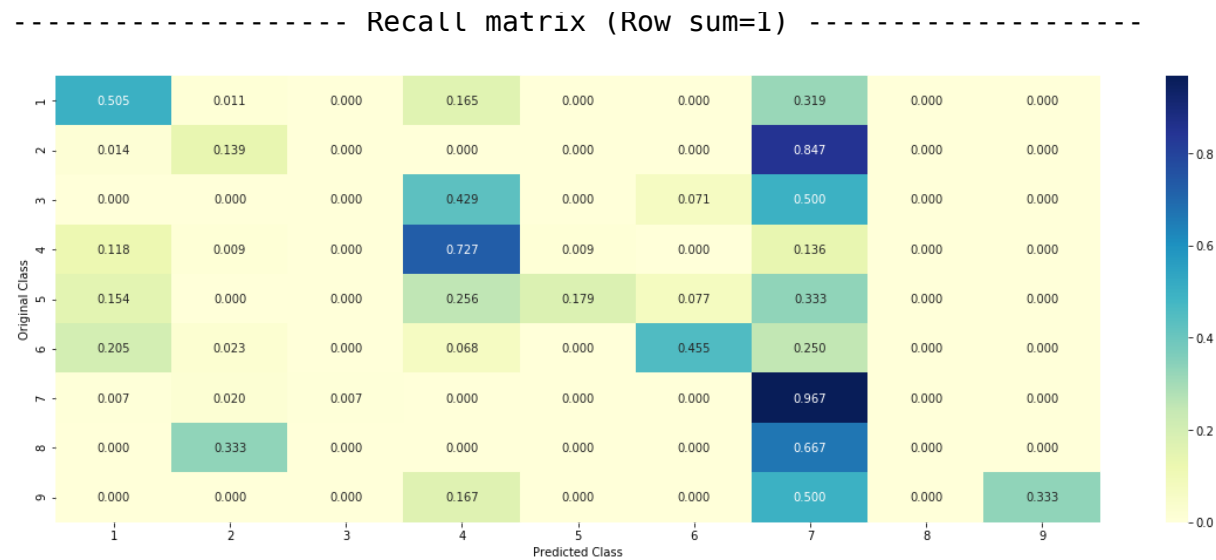
Number of missclassified point : 0.4116541353383459

----- Confusion matrix -----



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





4.1.1.3. Feature Importance, Correctly classified point

```
In [59]: test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
,test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[5.660e-02 6.760e-02 5.700e-03 8.180e-0
2 2.630e-02 1.660e-02 7.421e-01
2.900e-03 4.000e-04]]
Actual Class : 7
```

17 Text feature [cells] present in test data point [True]
18 Text feature [activated] present in test data point [True]
20 Text feature [kinase] present in test data point [True]
21 Text feature [cell] present in test data point [True]
22 Text feature [downstream] present in test data point [True]
23 Text feature [activation] present in test data point [True]
24 Text feature [phosphorylation] present in test data point [True]
25 Text feature [presence] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
27 Text feature [shown] present in test data point [True]
28 Text feature [expressing] present in test data point [True]
29 Text feature [factor] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [signaling] present in test data point [True]
32 Text feature [recently] present in test data point [True]
33 Text feature [inhibitor] present in test data point [True]
34 Text feature [however] present in test data point [True]
35 Text feature [suggest] present in test data point [True]
36 Text feature [compared] present in test data point [True]
37 Text feature [growth] present in test data point [True]
38 Text feature [addition] present in test data point [True]
39 Text feature [10] present in test data point [True]
40 Text feature [previously] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [independent] present in test data point [True]
44 Text feature [1a] present in test data point [True]
45 Text feature [increased] present in test data point [True]
46 Text feature [tyrosine] present in test data point [True]
47 Text feature [mechanism] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [figure] present in test data point [True]
50 Text feature [found] present in test data point [True]
51 Text feature [showed] present in test data point [True]
52 Text feature [mutations] present in test data point [True]
53 Text feature [higher] present in test data point [True]
54 Text feature [described] present in test data point [True]
56 Text feature [enhanced] present in test data point [True]

57 Text feature [constitutive] present in test data point [True]

57 Text feature [constitutive] present in test data point [True]
59 Text feature [observed] present in test data point [True]
60 Text feature [constitutively] present in test data point [True]
61 Text feature [activating] present in test data point [True]
62 Text feature [mutant] present in test data point [True]
63 Text feature [using] present in test data point [True]
65 Text feature [potential] present in test data point [True]
66 Text feature [including] present in test data point [True]
67 Text feature [3b] present in test data point [True]
68 Text feature [sensitive] present in test data point [True]
69 Text feature [treatment] present in test data point [True]
71 Text feature [followed] present in test data point [True]
72 Text feature [antibodies] present in test data point [True]
73 Text feature [may] present in test data point [True]
74 Text feature [various] present in test data point [True]
75 Text feature [furthermore] present in test data point [True]
76 Text feature [obtained] present in test data point [True]
77 Text feature [confirmed] present in test data point [True]
78 Text feature [reported] present in test data point [True]
79 Text feature [mutation] present in test data point [True]
80 Text feature [inhibition] present in test data point [True]
81 Text feature [approved] present in test data point [True]
82 Text feature [two] present in test data point [True]
83 Text feature [identified] present in test data point [True]
85 Text feature [expression] present in test data point [True]
86 Text feature [without] present in test data point [True]
87 Text feature [3a] present in test data point [True]
88 Text feature [respectively] present in test data point [True]
89 Text feature [inhibitors] present in test data point [True]
90 Text feature [proliferation] present in test data point [True]
91 Text feature [absence] present in test data point [True]
92 Text feature [pathways] present in test data point [True]
94 Text feature [fig] present in test data point [True]
95 Text feature [increase] present in test data point [True]
97 Text feature [suggesting] present in test data point [True]
98 Text feature [expressed] present in test data point [True]
99 Text feature [total] present in test data point [True]
Out of the top 100 features 75 are present in query point

4.1.1.4. Feature Importance, Incorrectly classified point

```
In [60]: test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
,test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

Predicted Class : 7

Predicted Class Probabilities: [[4.410e-02 1.531e-01 5.700e-03 6.480e-0
2 3.150e-02 2.130e-02 6.762e-01
2.900e-03 4.000e-04]]

Actual Class : 2

17 Text feature [cells] present in test data point [True]
18 Text feature [activated] present in test data point [True]
20 Text feature [kinase] present in test data point [True]
21 Text feature [cell] present in test data point [True]
22 Text feature [downstream] present in test data point [True]
23 Text feature [activation] present in test data point [True]
24 Text feature [phosphorylation] present in test data point [True]
25 Text feature [presence] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
27 Text feature [shown] present in test data point [True]
28 Text feature [expressing] present in test data point [True]
29 Text feature [factor] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [signaling] present in test data point [True]
33 Text feature [inhibitor] present in test data point [True]
34 Text feature [however] present in test data point [True]
35 Text feature [suggest] present in test data point [True]
36 Text feature [compared] present in test data point [True]

37 Text feature [growth] present in test data point [True]
38 Text feature [addition] present in test data point [True]
39 Text feature [10] present in test data point [True]
40 Text feature [previously] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [similar] present in test data point [True]
44 Text feature [1a] present in test data point [True]
45 Text feature [increased] present in test data point [True]
47 Text feature [mechanism] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [figure] present in test data point [True]
50 Text feature [found] present in test data point [True]
51 Text feature [showed] present in test data point [True]
52 Text feature [mutations] present in test data point [True]
53 Text feature [higher] present in test data point [True]
54 Text feature [described] present in test data point [True]
56 Text feature [enhanced] present in test data point [True]
58 Text feature [interestingly] present in test data point [True]
59 Text feature [observed] present in test data point [True]
60 Text feature [constitutively] present in test data point [True]
61 Text feature [activating] present in test data point [True]
62 Text feature [mutant] present in test data point [True]
63 Text feature [using] present in test data point [True]
64 Text feature [demonstrated] present in test data point [True]
65 Text feature [potential] present in test data point [True]
66 Text feature [including] present in test data point [True]
67 Text feature [3b] present in test data point [True]
68 Text feature [sensitive] present in test data point [True]
69 Text feature [treatment] present in test data point [True]
70 Text feature [consistent] present in test data point [True]
71 Text feature [followed] present in test data point [True]
72 Text feature [antibodies] present in test data point [True]
73 Text feature [may] present in test data point [True]
75 Text feature [furthermore] present in test data point [True]
77 Text feature [confirmed] present in test data point [True]
78 Text feature [reported] present in test data point [True]
79 Text feature [mutation] present in test data point [True]
80 Text feature [inhibition] present in test data point [True]
82 Text feature [two] present in test data point [True]

```
83 Text feature [identified] present in test data point [True]
85 Text feature [expression] present in test data point [True]
86 Text feature [without] present in test data point [True]
87 Text feature [3a] present in test data point [True]
89 Text feature [inhibitors] present in test data point [True]
92 Text feature [pathways] present in test data point [True]
93 Text feature [antibody] present in test data point [True]
94 Text feature [fig] present in test data point [True]
95 Text feature [increase] present in test data point [True]
97 Text feature [suggesting] present in test data point [True]
98 Text feature [expressed] present in test data point [True]
99 Text feature [total] present in test data point [True]
Out of the top 100 features 69 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```
In [61]: 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 probabilities 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 vali
      dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
      =1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
      oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

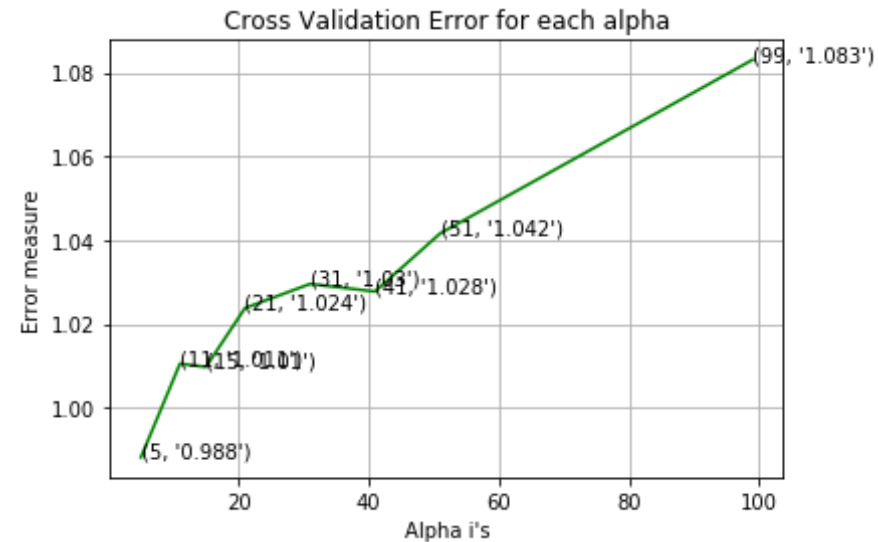
```

```

for alpha = 5
Log Loss : 0.9881838246433937
for alpha = 11
Log Loss : 1.0105218019021749
for alpha = 15
Log Loss : 1.0097559384922905
for alpha = 21
Log Loss : 1.023794697931827
for alpha = 31
Log Loss : 1.0295520045087077
for alpha = 41
Log Loss : 1.0277470407316165
for alpha = 51
Log Loss : 1.0416120699103093

```


for alpha = 99
Log Loss : 1.083037614717538

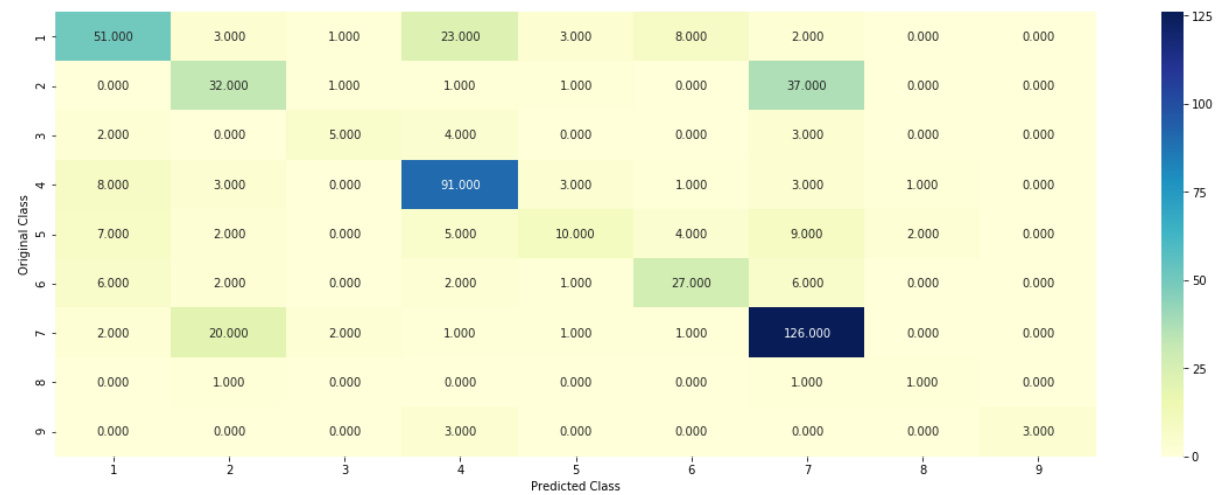


For values of best alpha = 5 The train log loss is: 0.50011509043501
For values of best alpha = 5 The cross validation log loss is: 0.9881838246433937
For values of best alpha = 5 The test log loss is: 1.0525659637326632

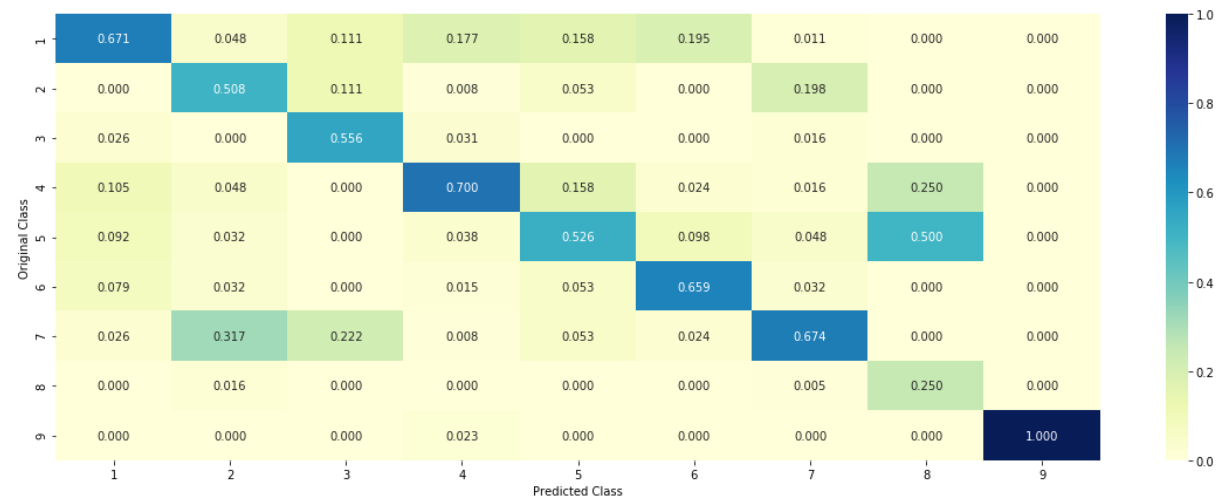
4.2.2. Testing the model with best hyper paramters

```
In [62]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])  
         predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x  
         _responseCoding, cv_y, clf)
```

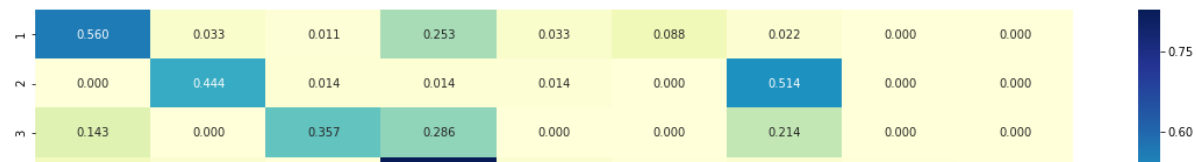
Log loss : 0.9881838246433937
Number of mis-classified points : 0.34962406015037595
----- Confusion matrix -----

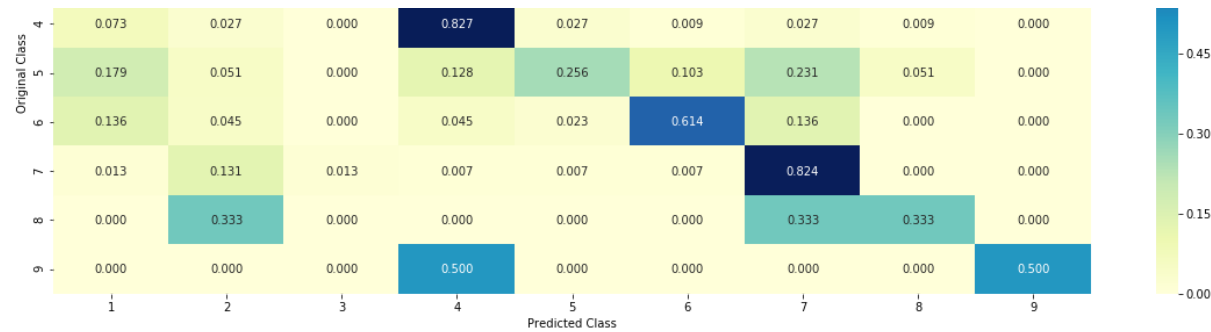


----- Precision matrix (Column Sum=1) -----



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





4.2.3. Sample Query point -1

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

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

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

4.2.4. Sample Query Point-2

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

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].resh
ape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours 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 : 2

the k value for knn is 5 and the nearest neighbours of the test points

belongs to classes [2 7 7 2 2]

Fequency of nearest points : Counter({2: 3, 7: 2})

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

```
In [65]: 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 probabilities 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], p
enalty='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)

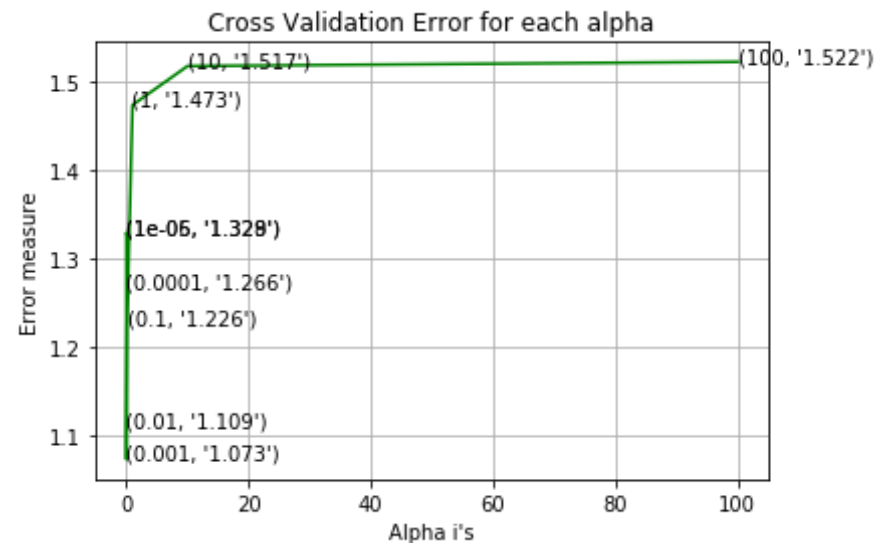
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-06
Log Loss : 1.3282329294525494
for alpha = 1e-05
Log Loss : 1.328821736653203
for alpha = 0.0001
Log Loss : 1.2658348069408547
for alpha = 0.001
Log Loss : 1.0730693523609576
for alpha = 0.01
Log Loss : 1.1093372438888018
for alpha = 0.1
Log Loss : 1.2256770926510925
for alpha = 1
Log Loss : 1.4731374260149372
for alpha = 10
Log Loss : 1.5171311499116427
for alpha = 100
Log Loss : 1.5219095209728866

```



For values of best alpha = 0.001 The train log loss is: 0.5631375273337421

For values of best alpha = 0.001 The cross validation log loss is: 1.0730693523609576

For values of best alpha = 0.001 The cross validation log loss is: 1.0730693523609576

For values of best alpha = 0.001 The test log loss is: 1.0821610079638722

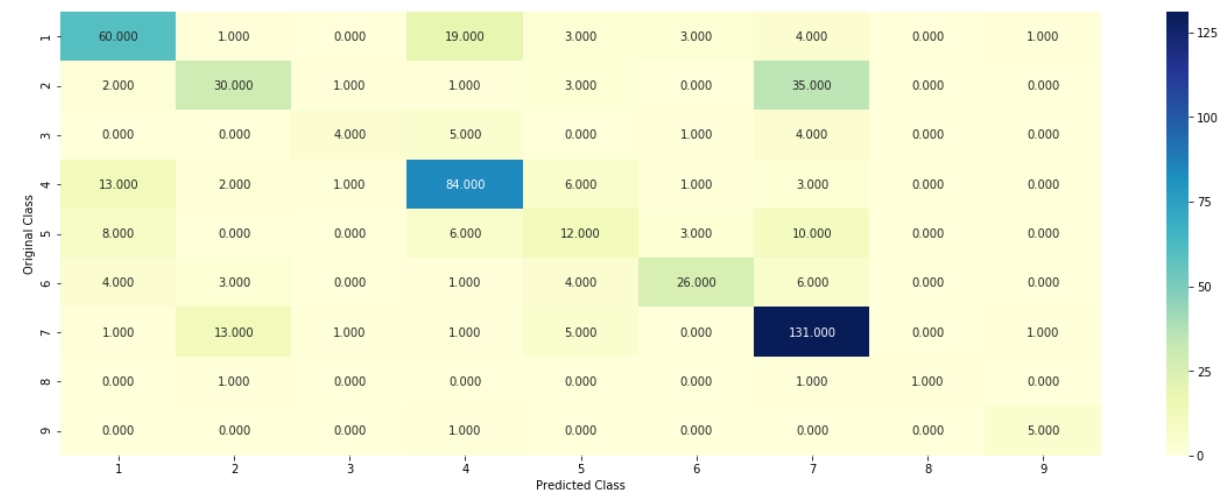
4.3.1.2. Testing the model with best hyper paramters

```
In [66]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
        :   enalty='l2', loss='log', random_state=42)
        :   predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
        :   nehotCoding, cv_y, clf)
```

Log loss : 1.0730693523609576

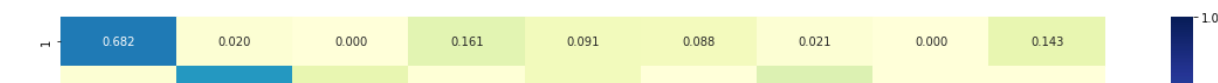
Number of mis-classified points : 0.33646616541353386

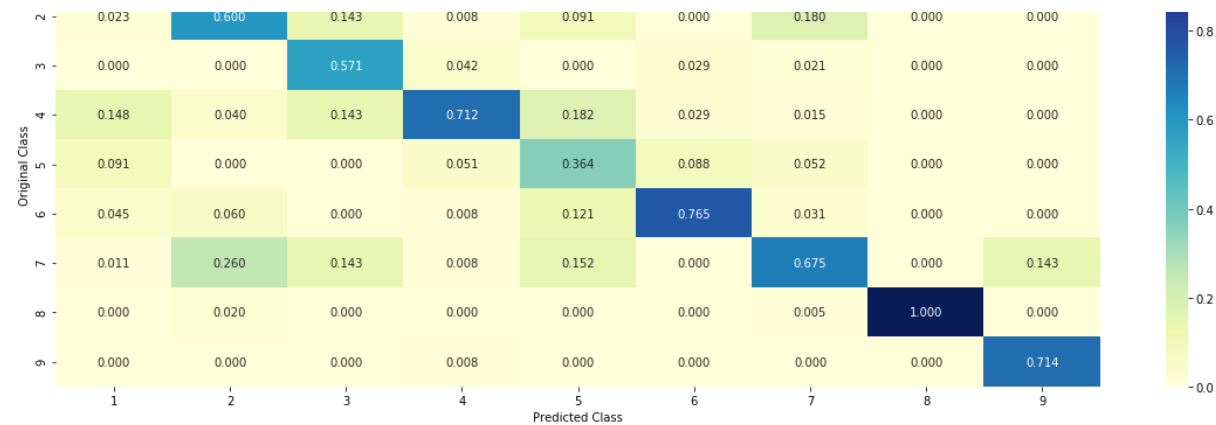
----- Confusion matrix -----



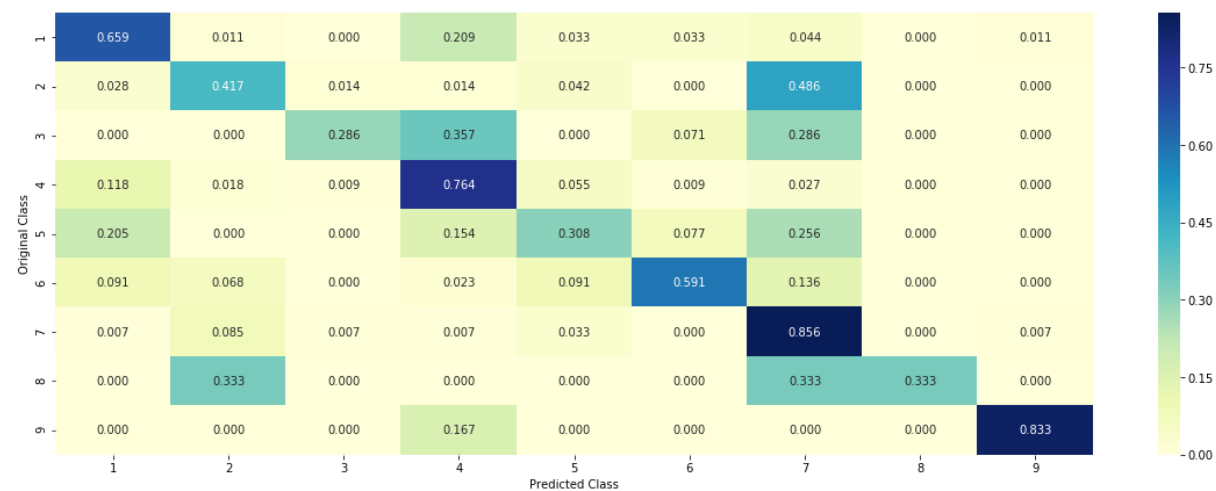
----- Precision matrix (Column Sum=1) -----

--





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



4.3.1.3. Feature Importance

```
In [67]: def get_imp_feature_names(text, indices, removed_ind = []):
          word_present = 0
          tabulte_list = []
          incresingorder_ind = 0
          for i in indices:
              if i < train_gene_feature_onehotCoding.shape[1]:
```



```

        tabulte_list.append([increasingorder_ind, "Gene", "Yes"])
    elif i < 18:
        tabulte_list.append([increasingorder_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([increasingorder_ind, train_text_features
[i], yes_no])
            increasingorder_ind += 1
        print(word_present, "most important features are present in our que
ry point")
        print("-"*50)
        print("The features that are most important of the ", predicted_cls[
0], " class:")
        print(tabulate(tabulte_list, headers=["Index", 'Feature name', 'Pre
sent or Not']))

```

4.3.1.3.1. Correctly Classified point

```

In [68]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test
_point_index], no_feature)

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0288 0.0655 0.0164 0.0644 0.0252 0.0
094 0.7749 0.0087 0.0068]]
Actual Class : 7
-----
22 Text feature [constitutive] present in test data point [True]
29 Text feature [activated] present in test data point [True]
34 Text feature [stat] present in test data point [True]
37 Text feature [oncogene] present in test data point [True]
61 Text feature [constitutively] present in test data point [True]
68 Text feature [downstream] present in test data point [True]
73 Text feature [receptors] present in test data point [True]
83 Text feature [ba] present in test data point [True]
92 Text feature [f3] present in test data point [True]
128 Text feature [stat5] present in test data point [True]
133 Text feature [activation] present in test data point [True]
150 Text feature [ligand] present in test data point [True]
155 Text feature [technology] present in test data point [True]
207 Text feature [ccl3] present in test data point [True]
238 Text feature [improvement] present in test data point [True]
246 Text feature [janus] present in test data point [True]
251 Text feature [transformation] present in test data point [True]
252 Text feature [eco] present in test data point [True]
253 Text feature [scf] present in test data point [True]
331 Text feature [infect] present in test data point [True]
348 Text feature [subcutaneous] present in test data point [True]
351 Text feature [expressing] present in test data point [True]
380 Text feature [jak] present in test data point [True]
399 Text feature [doses] present in test data point [True]
403 Text feature [socs2] present in test data point [True]
434 Text feature [casp3] present in test data point [True]
447 Text feature [phosphorylation] present in test data point [True]
Out of the top 500 features 27 are present in query point

```

4.3.1.3.2. Incorrectly Classified point

```
In [71]: test_point_index = 101
```

```

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

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0295 0.0915 0.0101 0.0841 0.0153 0.0
153 0.7328 0.0178 0.0036]]
Actual Class : 2

```

```

-----
37 Text feature [oncogene] present in test data point [True]
68 Text feature [downstream] present in test data point [True]
73 Text feature [receptors] present in test data point [True]
89 Text feature [behaviors] present in test data point [True]
133 Text feature [activation] present in test data point [True]
155 Text feature [technology] present in test data point [True]
223 Text feature [tumors17] present in test data point [True]
351 Text feature [expressing] present in test data point [True]
367 Text feature [overlapped] present in test data point [True]
413 Text feature [hazards] present in test data point [True]
439 Text feature [thiazolyl] present in test data point [True]
440 Text feature [glyl2val] present in test data point [True]
494 Text feature [oncogenes] present in test data point [True]
Out of the top 500 features 13 are present in query point

```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```

In [72]: 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 vali
dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))

```

```

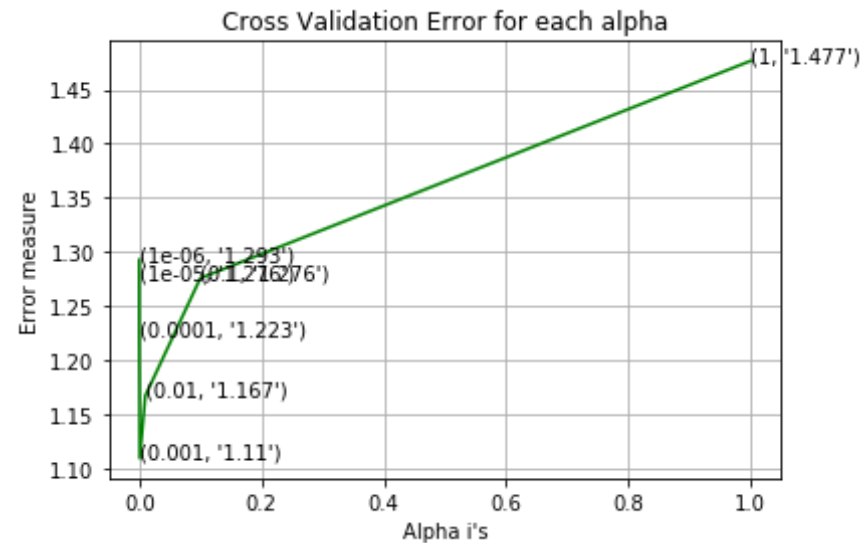
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```

for alpha = 1e-06
Log Loss : 1.2928040278150748
for alpha = 1e-05
Log Loss : 1.2760330515803373
for alpha = 0.0001
Log Loss : 1.2228972909402183
for alpha = 0.001
Log Loss : 1.1095237316841549
for alpha = 0.01
Log Loss : 1.167269910497706
for alpha = 0.1
Log Loss : 1.2758577096288812
for alpha = 1
Log Loss : 1.4765100362965882

```



For values of best alpha = 0.001 The train log loss is: 0.5571296122667088

For values of best alpha = 0.001 The cross validation log loss is: 1.1095237316841549

For values of best alpha = 0.001 The test log loss is: 1.0966923741621784

4.3.2.2. Testing model with best hyper parameters

```
In [73]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

Log loss : 1.1095237316841549

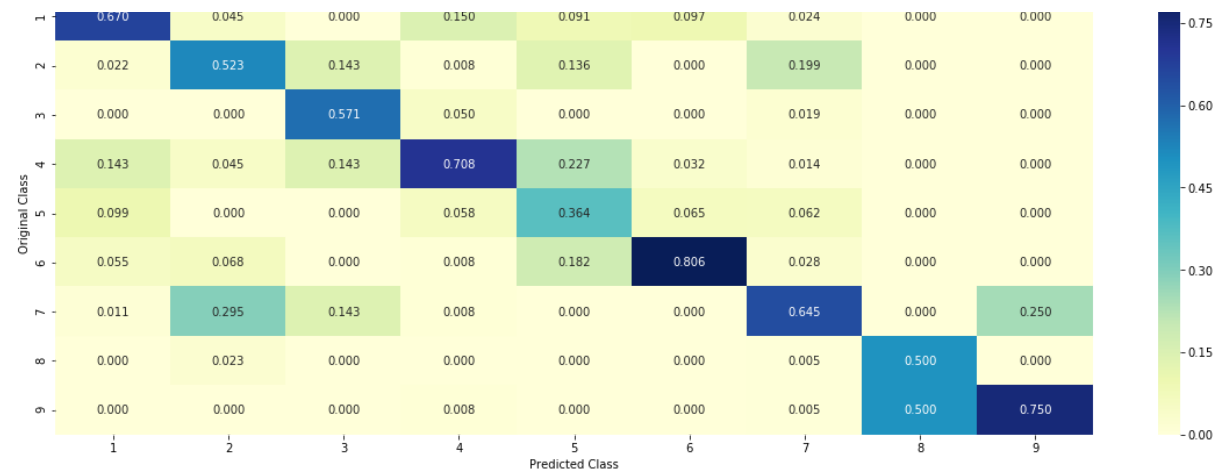
Number of mis-classified points : 0.34962406015037595

----- Confusion matrix -----

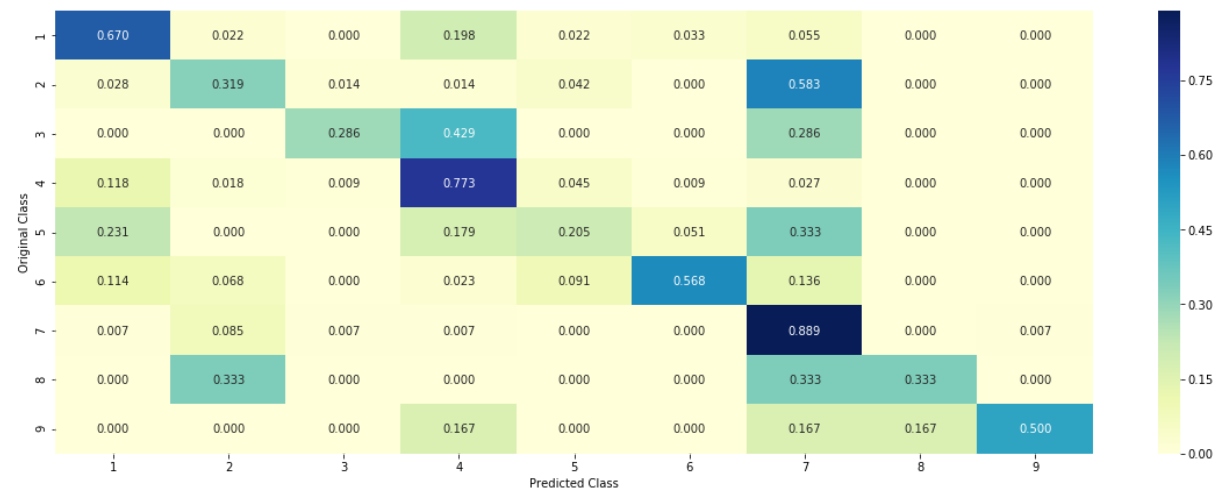


----- Precision matrix (Column Sum=1) -----

--



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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [74]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
```

```

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

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0313 0.0663 0.0049 0.076 0.0174 0.0
075 0.7894 0.0062 0.001]]

Actual Class : 7

```

-----
45 Text feature [constitutive] present in test data point [True]
85 Text feature [activated] present in test data point [True]
92 Text feature [oncogene] present in test data point [True]
123 Text feature [constitutively] present in test data point [True]
137 Text feature [ba] present in test data point [True]
139 Text feature [stat] present in test data point [True]
141 Text feature [downstream] present in test data point [True]
142 Text feature [f3] present in test data point [True]
180 Text feature [receptors] present in test data point [True]
192 Text feature [expressing] present in test data point [True]
199 Text feature [stat5] present in test data point [True]
206 Text feature [ligand] present in test data point [True]
213 Text feature [activation] present in test data point [True]
227 Text feature [transformation] present in test data point [True]
261 Text feature [technology] present in test data point [True]
279 Text feature [phosphorylation] present in test data point [True]
350 Text feature [ccl3] present in test data point [True]
430 Text feature [improvement] present in test data point [True]
437 Text feature [subcutaneous] present in test data point [True]
444 Text feature [casp3] present in test data point [True]
463 Text feature [doses] present in test data point [True]
469 Text feature [driving] present in test data point [True]

```


490 Text feature [janus] present in test data point [True]
Out of the top 500 features 23 are present in query point

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [76]: test_point_index = 101
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
,test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.270e-02 6.360e-02 1.100e-03 9.050e-0
2 5.900e-03 7.700e-03 7.812e-01
2.720e-02 1.000e-04]]
```

```
Actual Class : 2
```

```
-----
92 Text feature [oncogene] present in test data point [True]
141 Text feature [downstream] present in test data point [True]
180 Text feature [receptors] present in test data point [True]
192 Text feature [expressing] present in test data point [True]
213 Text feature [activation] present in test data point [True]
255 Text feature [behaviors] present in test data point [True]
261 Text feature [technology] present in test data point [True]
313 Text feature [tumors17] present in test data point [True]
365 Text feature [glyl2val] present in test data point [True]
384 Text feature [overlapped] present in test data point [True]
424 Text feature [proto] present in test data point [True]
Out of the top 500 features 11 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper parameter tuning

```
In [77]: 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 vali
      dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
      =1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
      oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

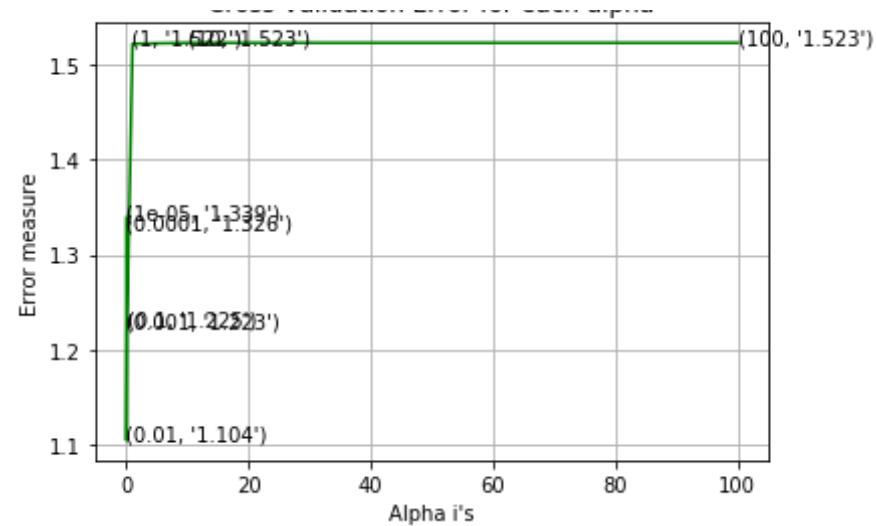
```

```

for C = 1e-05
Log Loss : 1.3386925514577264
for C = 0.0001
Log Loss : 1.3264513253751111
for C = 0.001
Log Loss : 1.22258298538971
for C = 0.01
Log Loss : 1.104427265253595
for C = 0.1
Log Loss : 1.2249462350246827
for C = 1
Log Loss : 1.5217980180981943
for C = 10
Log Loss : 1.5228372231678236
for C = 100
Log Loss : 1.522837513137167

```

Cross Validation Error for each alpha



For values of best alpha = 0.01 The train log loss is: 0.6876365076532395
 For values of best alpha = 0.01 The cross validation log loss is: 1.104427265253595
 For values of best alpha = 0.01 The test log loss is: 1.0994975551953015

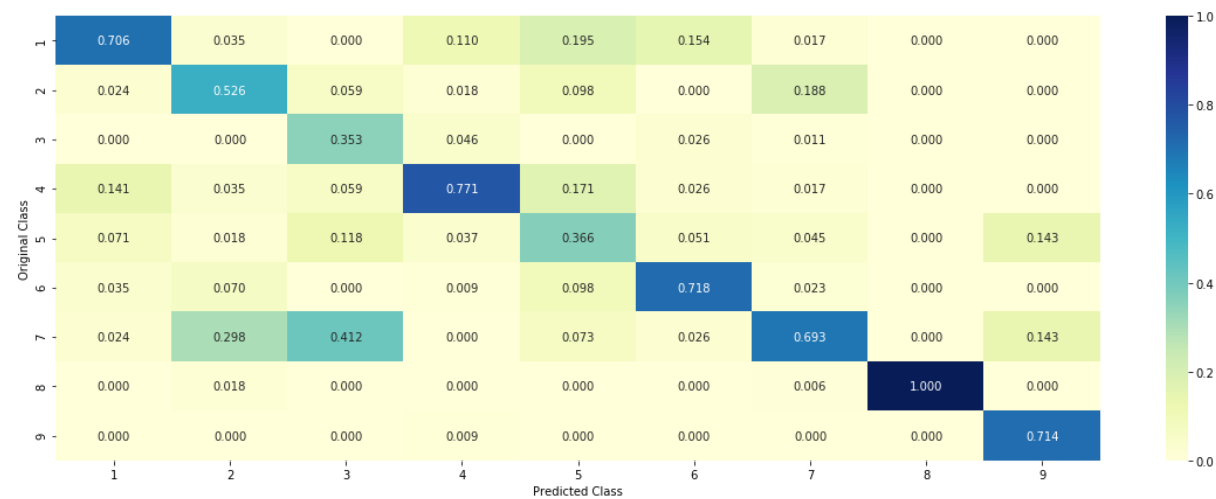
4.4.2. Testing model with best hyper parameters

```
In [78]: # 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 : 1.104427265253595
 Number of mis-classified points : 0.34022556390977443
 ----- Confusion matrix -----



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
In [79]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index])
```

```
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 7

Predicted Class Probabilities: [[0.054 0.0716 0.0157 0.0726 0.0392 0.0146 0.7191 0.0082 0.0051]]

Actual Class : 7

```
-----
22 Text feature [constitutive] present in test data point [True]
29 Text feature [ba] present in test data point [True]
30 Text feature [f3] present in test data point [True]
31 Text feature [activated] present in test data point [True]
32 Text feature [constitutively] present in test data point [True]
33 Text feature [stat] present in test data point [True]
46 Text feature [activation] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
50 Text feature [stat5] present in test data point [True]
59 Text feature [transformation] present in test data point [True]
62 Text feature [ligand] present in test data point [True]
63 Text feature [downstream] present in test data point [True]
78 Text feature [phosphorylation] present in test data point [True]
79 Text feature [receptors] present in test data point [True]
112 Text feature [oncogene] present in test data point [True]
115 Text feature [technology] present in test data point [True]
123 Text feature [janus] present in test data point [True]
127 Text feature [stably] present in test data point [True]
129 Text feature [casp3] present in test data point [True]
158 Text feature [infect] present in test data point [True]
160 Text feature [eco] present in test data point [True]
165 Text feature [activating] present in test data point [True]
177 Text feature [doses] present in test data point [True]
186 Text feature [jak] present in test data point [True]
231 Text feature [ccl3] present in test data point [True]
243 Text feature [il] present in test data point [True]
315 Text feature [transduced] present in test data point [True]
317 Text feature [subcutaneous] present in test data point [True]
349 Text feature [electroporated] present in test data point [True]
359 Text feature [subcutaneously] present in test data point [True]
364 Text feature [jak1] present in test data point [True]
373 Text feature [absence] present in test data point [True]
```

```

385 Text feature [institutional] present in test data point [True]
403 Text feature [akt] present in test data point [True]
422 Text feature [jak2] present in test data point [True]
427 Text feature [scf] present in test data point [True]
440 Text feature [elicited] present in test data point [True]
441 Text feature [signaling] present in test data point [True]
450 Text feature [enhanced] present in test data point [True]
453 Text feature [socs2] present in test data point [True]
466 Text feature [receptor] present in test data point [True]
468 Text feature [promote] present in test data point [True]
487 Text feature [oncogenic] present in test data point [True]
Out of the top 500 features 43 are present in query point

```

4.3.3.2. For Incorrectly classified point

```

In [80]: test_point_index = 101
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0834 0.1345 0.017  0.1962 0.036  0.0
36  0.4693 0.0213 0.0064]]
Actual Class : 2
-----
46 Text feature [activation] present in test data point [True]
47 Text feature [expressing] present in test data point [True]
60 Text feature [behaviors] present in test data point [True]
63 Text feature [downstream] present in test data point [True]
79 Text feature [receptors] present in test data point [True]

```



```

112 Text feature [oncogene] present in test data point [True]
115 Text feature [technology] present in test data point [True]
165 Text feature [activating] present in test data point [True]
271 Text feature [overlapped] present in test data point [True]
325 Text feature [419] present in test data point [True]
441 Text feature [signaling] present in test data point [True]
450 Text feature [enhanced] present in test data point [True]
487 Text feature [oncogenic] present in test data point [True]
Out of the top 500 features 13 are present in query point

```

4.5 Random Forest Classifier

4.5.1. Hyper parameter tuning (With One hot Encoding)

```

In [81]: 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]).ra
vel()
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)), (featur

```

```

es[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)], cri
terion='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.cl
asses_, 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_, ep
s=1e-15))

for n_estimators = 100 and max depth = 5
Log Loss : 1.2452757506840286
for n_estimators = 100 and max depth = 10
Log Loss : 1.1705590936557948
for n_estimators = 200 and max depth = 5
Log Loss : 1.2275638799535957
for n_estimators = 200 and max depth = 10
Log Loss : 1.158123502496537
for n_estimators = 500 and max depth = 5
Log Loss : 1.2088634681971422
for n_estimators = 500 and max depth = 10

```

```

Log Loss : 1.1436300704868536
for n_estimators = 1000 and max depth = 5
Log Loss : 1.205109350341921
for n_estimators = 1000 and max depth = 10
Log Loss : 1.1477033993573926
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2022096388496406
for n_estimators = 2000 and max depth = 10
Log Loss : 1.146544882662607
For values of best estimator = 500 The train log loss is: 0.6683272380
844386
For values of best estimator = 500 The cross validation log loss is:
1.1436300704868536
For values of best estimator = 500 The test log loss is: 1.17019715614
68505

```

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

```

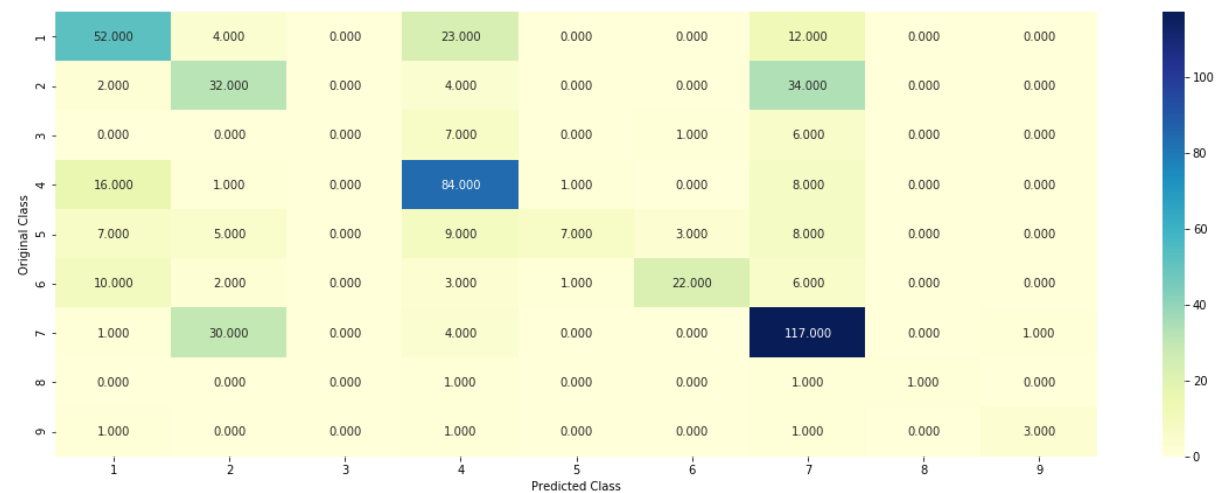
In [82]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)

```

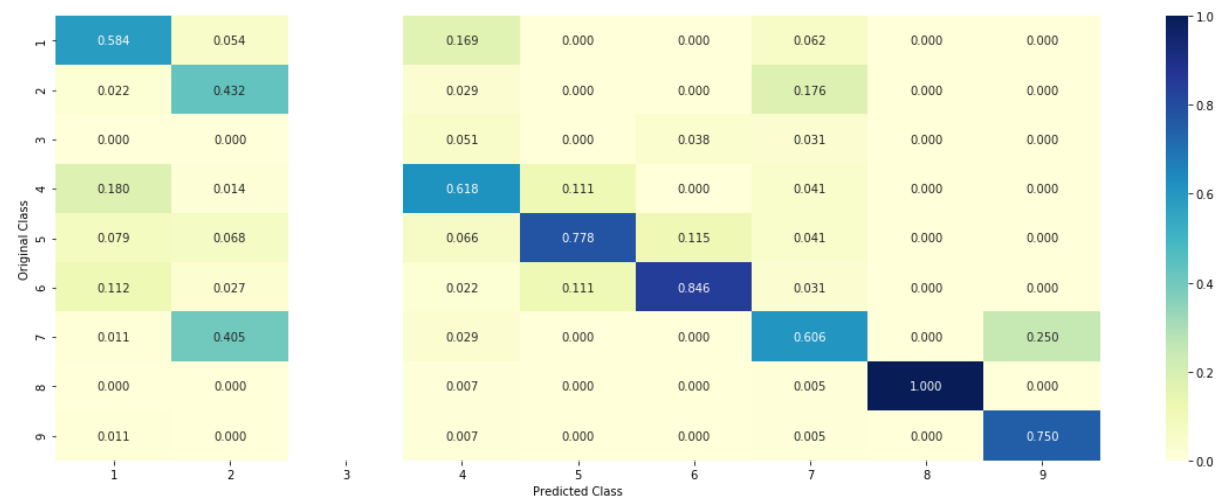
```

Log loss : 1.1436300704868536
Number of mis-classified points : 0.40225563909774437
----- Confusion matrix -----

```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
In [83]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
```

```

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

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0518 0.1391 0.022 0.0603 0.0458 0.037 0.6323 0.0061 0.0056]]

Actual Class : 7

```

-----
0 Text feature [activated] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [kinase] present in test data point [True]
7 Text feature [signaling] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
9 Text feature [constitutive] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [cells] present in test data point [True]
16 Text feature [receptor] present in test data point [True]
17 Text feature [ba] present in test data point [True]
18 Text feature [cell] present in test data point [True]
19 Text feature [inhibition] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
23 Text feature [transforming] present in test data point [True]
24 Text feature [constitutively] present in test data point [True]
26 Text feature [akt] present in test data point [True]
27 Text feature [function] present in test data point [True]
29 Text feature [efficacy] present in test data point [True]
30 Text feature [treated] present in test data point [True]
31 Text feature [kinases] present in test data point [True]
32 Text feature [trials] present in test data point [True]
33 Text feature [proliferation] present in test data point [True]
34 Text feature [clinical] present in test data point [True]
35 Text feature [treatment] present in test data point [True]
37 Text feature [harboring] present in test data point [True]

```

```

39 Text feature [drug] present in test data point [True]
40 Text feature [f3] present in test data point [True]
41 Text feature [effective] present in test data point [True]
42 Text feature [growth] present in test data point [True]
47 Text feature [protein] present in test data point [True]
49 Text feature [survival] present in test data point [True]
51 Text feature [functional] present in test data point [True]
52 Text feature [oncogene] present in test data point [True]
55 Text feature [advanced] present in test data point [True]
57 Text feature [concentrations] present in test data point [True]
63 Text feature [response] present in test data point [True]
67 Text feature [respond] present in test data point [True]
75 Text feature [extracellular] present in test data point [True]
76 Text feature [pathway] present in test data point [True]
78 Text feature [inhibited] present in test data point [True]
79 Text feature [expressing] present in test data point [True]
83 Text feature [amplification] present in test data point [True]
87 Text feature [lines] present in test data point [True]
88 Text feature [factor] present in test data point [True]
91 Text feature [therapeutic] present in test data point [True]
96 Text feature [active] present in test data point [True]
99 Text feature [length] present in test data point [True]
Out of the top 100 features 51 are present in query point

```

4.5.3.2. Inorrectly Classified point

```

In [84]: test_point_index = 101
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_po

```

```
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].  
iloc[test_point_index], no_feature)
```

Predicted Class : 2

Predicted Class Probabilities: [[0.1478 0.2737 0.0263 0.1458 0.0675 0.0
614 0.2577 0.0085 0.0113]]

Actual Class : 2

1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [kinase] present in test data point [True]
7 Text feature [signaling] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [cells] present in test data point [True]
18 Text feature [cell] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [loss] present in test data point [True]
27 Text feature [function] present in test data point [True]
33 Text feature [proliferation] present in test data point [True]
34 Text feature [clinical] present in test data point [True]
36 Text feature [patients] present in test data point [True]
37 Text feature [harboring] present in test data point [True]
39 Text feature [drug] present in test data point [True]
42 Text feature [growth] present in test data point [True]
47 Text feature [protein] present in test data point [True]
49 Text feature [survival] present in test data point [True]
51 Text feature [functional] present in test data point [True]
52 Text feature [oncogene] present in test data point [True]
56 Text feature [egfr] present in test data point [True]
63 Text feature [response] present in test data point [True]
70 Text feature [defective] present in test data point [True]
75 Text feature [extracellular] present in test data point [True]
76 Text feature [pathway] present in test data point [True]
78 Text feature [inhibited] present in test data point [True]
79 Text feature [expressing] present in test data point [True]
80 Text feature [sensitivity] present in test data point [True]
86 Text feature [variants] present in test data point [True]
87 Text feature [lines] present in test data point [True]
90 Text feature [functions] present in test data point [True]
...

Out of the top 100 features 32 are present in query point

4.5.3. Hyper paramter tuning (With Response Coding)

```
In [85]: 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]).ra
vel()
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)), (featur
es[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)], cri
terion='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 tra
in 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 cro
ss validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t 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 : 2.1369065965947756
for n_estimators = 10 and max depth = 3
Log Loss : 1.6944707639366468
for n_estimators = 10 and max depth = 5
Log Loss : 1.3638466676871959
for n_estimators = 10 and max depth = 10
Log Loss : 1.7693196808828073
for n_estimators = 50 and max depth = 2
Log Loss : 1.6790050178630294
for n_estimators = 50 and max depth = 3
Log Loss : 1.50065866044049
for n_estimators = 50 and max depth = 5
Log Loss : 1.3846055628013572
for n_estimators = 50 and max depth = 10
Log Loss : 1.7033158907647268
for n_estimators = 100 and max depth = 2
Log Loss : 1.5654302003723346
for n_estimators = 100 and max depth = 3
Log Loss : 1.5104541294765395
for n_estimators = 100 and max depth = 5
Log Loss : 1.3230175944786415
for n_estimators = 100 and max depth = 10

```

```

Log Loss : 1.6976466262759657
for n_estimators = 200 and max depth = 2
Log Loss : 1.605880502577581
for n_estimators = 200 and max depth = 3
Log Loss : 1.4928058162220916
for n_estimators = 200 and max depth = 5
Log Loss : 1.3818802200529443
for n_estimators = 200 and max depth = 10
Log Loss : 1.6868265414229946
for n_estimators = 500 and max depth = 2
Log Loss : 1.7038155637626107
for n_estimators = 500 and max depth = 3
Log Loss : 1.5665949315769603
for n_estimators = 500 and max depth = 5
Log Loss : 1.3758027754719382
for n_estimators = 500 and max depth = 10
Log Loss : 1.6703663058060363
for n_estimators = 1000 and max depth = 2
Log Loss : 1.6761983445950632
for n_estimators = 1000 and max depth = 3
Log Loss : 1.5833179232685373
for n_estimators = 1000 and max depth = 5
Log Loss : 1.3890518169021773
for n_estimators = 1000 and max depth = 10
Log Loss : 1.6744210103967314
For values of best alpha = 100 The train log loss is: 0.05328484948122
7387
For values of best alpha = 100 The cross validation log loss is: 1.323
0175944786413
For values of best alpha = 100 The test log loss is: 1.365041852871573
3

```

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

```

In [86]: clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='au
to', random_state=42)

```

```
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

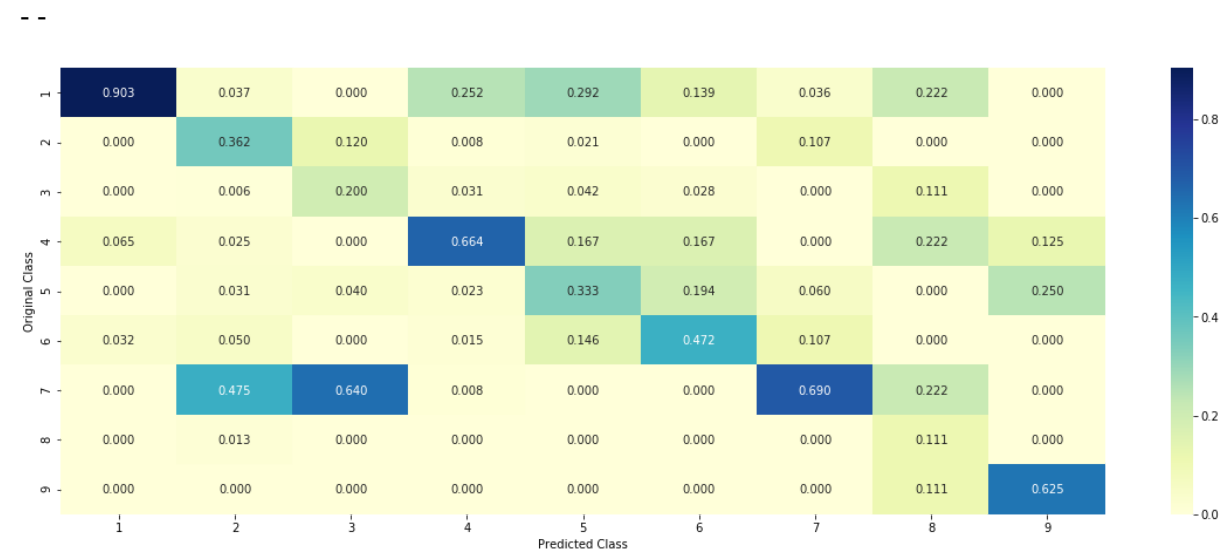
Log loss : 1.3230175944786415

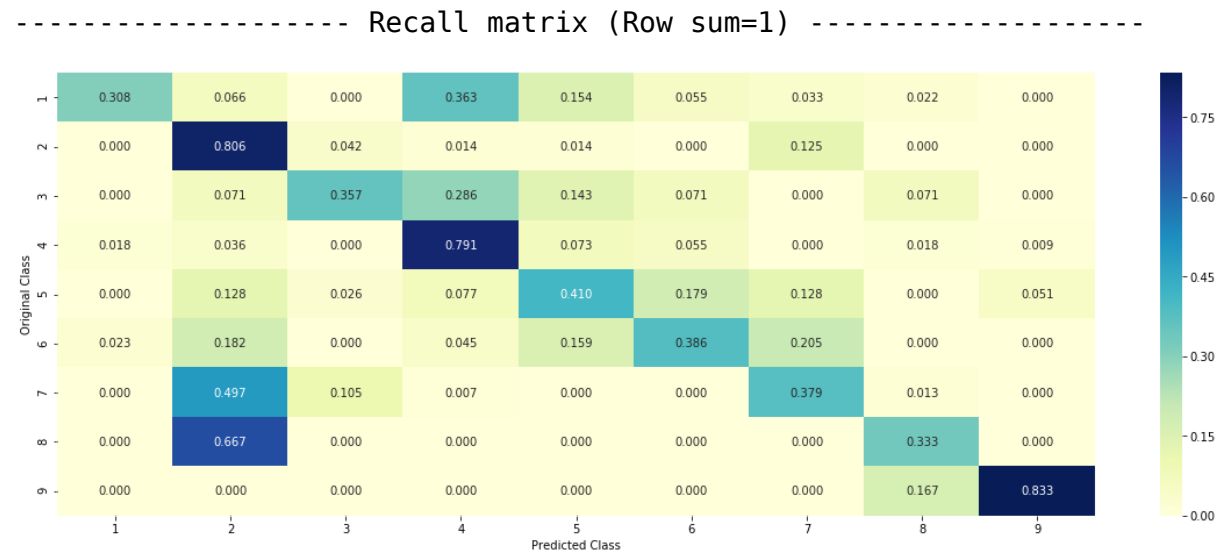
Number of mis-classified points : 0.4830827067669173

----- Confusion matrix -----



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
In [87]: 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 : 2
Predicted Class Probabilities: [[0.0749 0.1635 0.1471 0.0582 0.045  0.1
094 0.1196 0.1401 0.1422]]
Actual Class : 7

```

```

-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene is important feature

```

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

4.5.5.2. Incorrectly Classified point

```
In [88]: test_point_index = 101
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 : 2
Predicted Class Probabilities: [[0.0161 0.3754 0.0959 0.0264 0.0439 0.0
495 0.3546 0.0177 0.0205]]
Actual Class : 2
```

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

```
Gene is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```
In [89]: 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 Classifier : 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: 1.08
Support vector machines : Log Loss: 1.52
Naive Bayes : Log Loss: 1.24

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.487
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.097
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.207
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.525

```

4.7.2 testing the model with the best hyper parameters

```
In [90]: lr = LogisticRegression(C=0.1)
scf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
scf.fit(train_x_onehotCoding, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((scf.predict(test_x_onehotCoding) - test_y)))
plot_confusion_matrix(test_y=test_y, predict_y=scf.predict(test_x_onehotCoding))
```

Log loss (train) on the stacking classifier : 0.6245556767112924

Log loss (CV) on the stacking classifier : 1.097260894582807

Log loss (test) on the stacking classifier : 1.141165067793339

Number of missclassified point : 0.37894736842105264

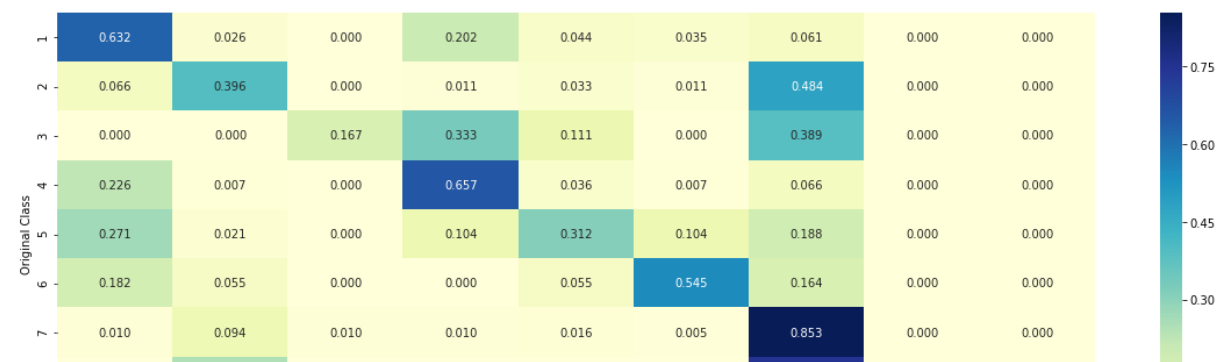
----- Confusion matrix -----

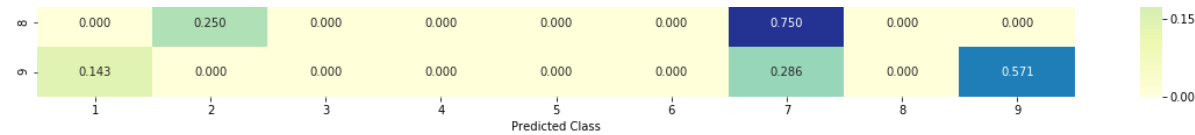


----- Precision matrix (Column Sum=1) -----



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

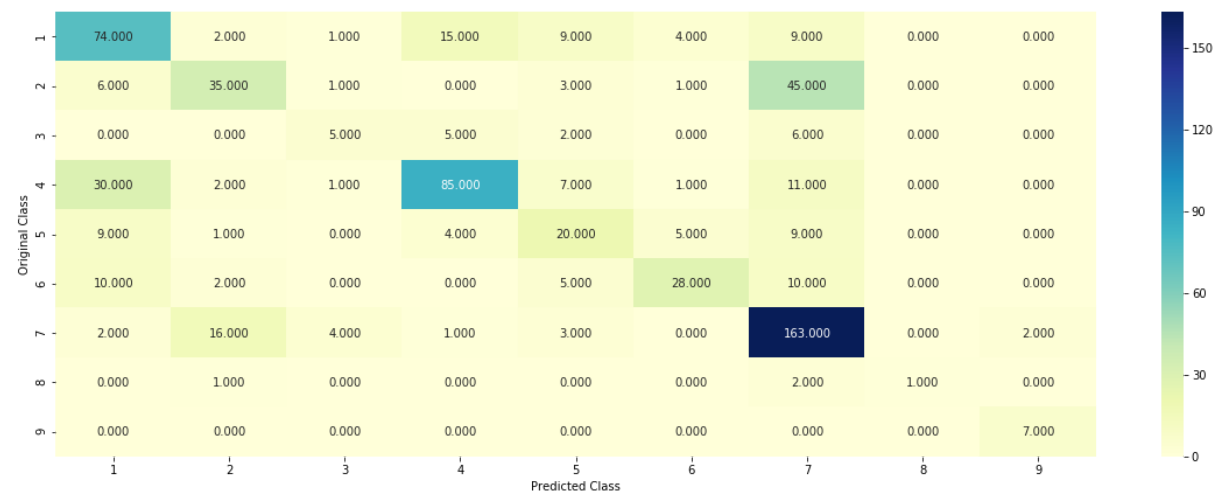




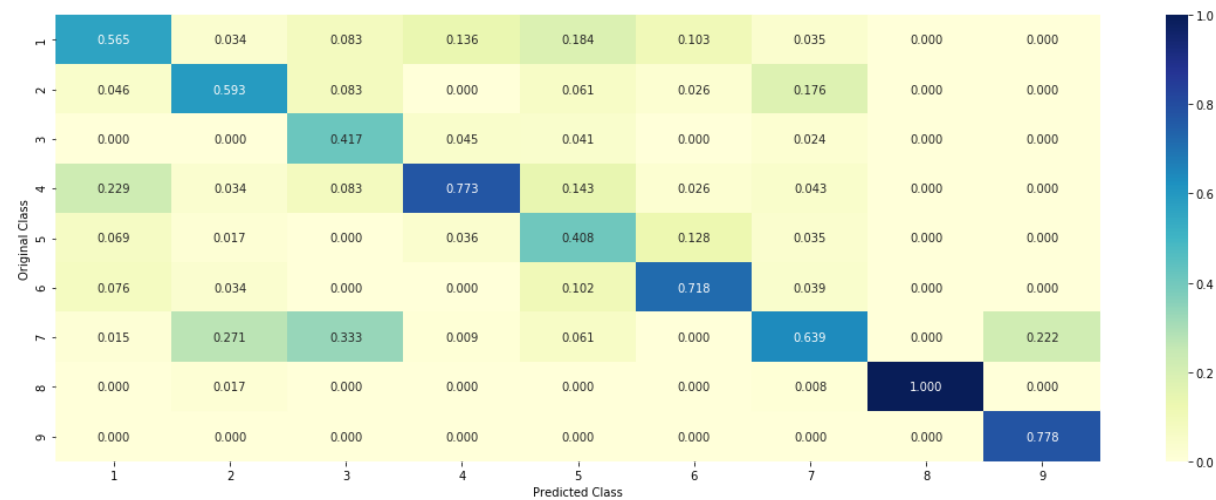
4.7.3 Maximum Voting classifier

```
In [91]: #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))
```

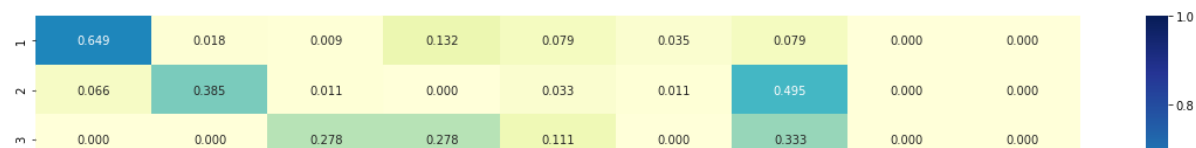
```
Log loss (train) on the VotingClassifier : 0.8460644743993057
Log loss (CV) on the VotingClassifier : 1.128257826906367
Log loss (test) on the VotingClassifier : 1.1669814813112478
Number of missclassified point : 0.37142857142857144
----- Confusion matrix -----
```

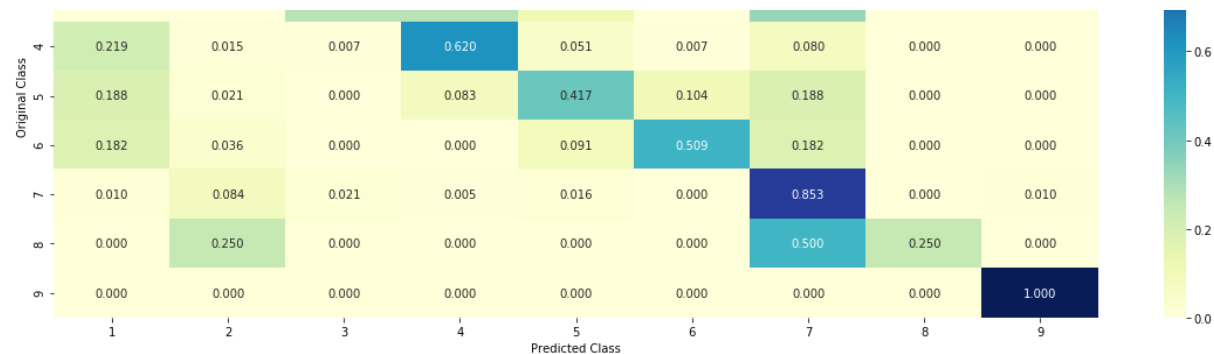


----- Precision matrix (Column Sum=1) -----



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





```
In [92]: from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No", "Model", "Train logloss", "Cv logloss", "Test logloss", "Misclassified error"]

x.add_row(["1", "Naive Bayes", "0.915", "1.206", "1.240", "0.41"])
x.add_row(["2", "KNN", "0.500", "0.988", "1.052", "0.34"])
x.add_row(["3", "Logistic regression with C.B", "0.56", "1.07", "1.08", "0.336"])
x.add_row(["4", "Logistic regression without C.B", "0.557", "1.109", "1.109", "0.349"])
x.add_row(["5", "Linear svm(with one hot encoding)", "0.687", "1.104", "1.099", "0.340"])
x.add_row(["6", "Random Forest(with one hot encoding)", "0.66", "1.14", "1.17", "0.402"])
x.add_row(["7", "Random Forest(with response coding)", "0.05", "1.32", "1.36", "0.483"])
x.add_row(["8", "Stacking classifier", "0.62", "1.09", "1.14", "0.37"])
x.add_row(["9", "Maximum voting classifier", "0.84", "1.12", "1.16", "0.37"])

print(x)
```

```
+-----+-----+-----+-----+-----+
| S.No | Model | Train logloss | Cv logloss | Test logloss | Misclassified error |
+-----+-----+-----+-----+-----+
| 1 | Naive Bayes | 0.915 | 1.206 | 1.240 | 0.41 |
| 2 | KNN | 0.500 | 0.988 | 1.052 | 0.34 |
| 3 | Logistic regression with C.B | 0.56 | 1.07 | 1.08 | 0.336 |
| 4 | Logistic regression without C.B | 0.557 | 1.109 | 1.109 | 0.349 |
| 5 | Linear svm(with one hot encoding) | 0.687 | 1.104 | 1.099 | 0.340 |
| 6 | Random Forest(with one hot encoding) | 0.66 | 1.14 | 1.17 | 0.402 |
| 7 | Random Forest(with response coding) | 0.05 | 1.32 | 1.36 | 0.483 |
| 8 | Stacking classifier | 0.62 | 1.09 | 1.14 | 0.37 |
| 9 | Maximum voting classifier | 0.84 | 1.12 | 1.16 | 0.37 |
```

S.NO	Model	Train logloss	CV logloss
Test logloss	Misclassified error		
1	Naive Bayes	0.915	1.206
2	KNN	0.500	0.988
3	Logistic regression with C.B	0.56	1.07
4	Logistic regression without C.B	0.557	1.109
5	Linear svm(with one hot encoding)	0.687	1.104
6	Random Forest(with one hot encoding)	0.66	1.14
7	Random Forest(with response coding)	0.05	1.32
8	Stacking classifier	0.62	1.09
9	Maximum voting classifier	0.84	1.12

Conclusion

Here I replaced everything with Tfidf vectorizer and i took all the points and when we observe this KNN has decent logloss values and reasonable Misclassification error compared to all other.. It seems slightly unstable but better than all other ...