

Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Problem statement : ¶

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>
(<https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>)
2. <https://www.youtube.com/watch?v=UwbuW7oK8rk> (<https://www.youtube.com/watch?v=UwbuW7oK8rk>)
3. <https://www.youtube.com/watch?v=qxXRKVompl8> (<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>
(<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>
(<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 [2]: 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 [3]: 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[3]:

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

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

Fields are

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

3.1.2. Reading Text Data

```
In [4]: # note the separator in this file
data_text = pd.read_csv("training_text", sep="\|", engine="python", names=
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[4]:

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

3.1.3. Preprocessing of text

```
In [5]: # loading stop words from nltk library
stop_words = set(stopwords.words('english'))

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

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

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

```
In [6]: #text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time)
```

```
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 : 225.48872599999999 seconds
```

```
In [7]: #merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[7]:

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

```
In [8]: result[result.isnull().any(axis=1)]
```

Out[8]:

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

```
In [9]: result.loc[result['TEXT'].isnull(), 'TEXT'] = result['Gene'] + ' '+result['Variation'] + ' '+result['Class']
```



```
In [10]: result[result['ID']==1109]
```

```
Out[10]:
```

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

3.1.4. Test, Train and Cross Validation Split

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

```
In [11]: y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution
X_train, test_df, y_train, y_test = train_test_split(result, y_true, s
# split the train data into train and cross validation by maintaining
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, st
```

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

```
In [12]: print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

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

```
In [13]: # it returns a dict, keys as class labels and values as the number of
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort
# -(train_class_distribution.values): the minus sign will give us in d
```

```

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', train_class_distr

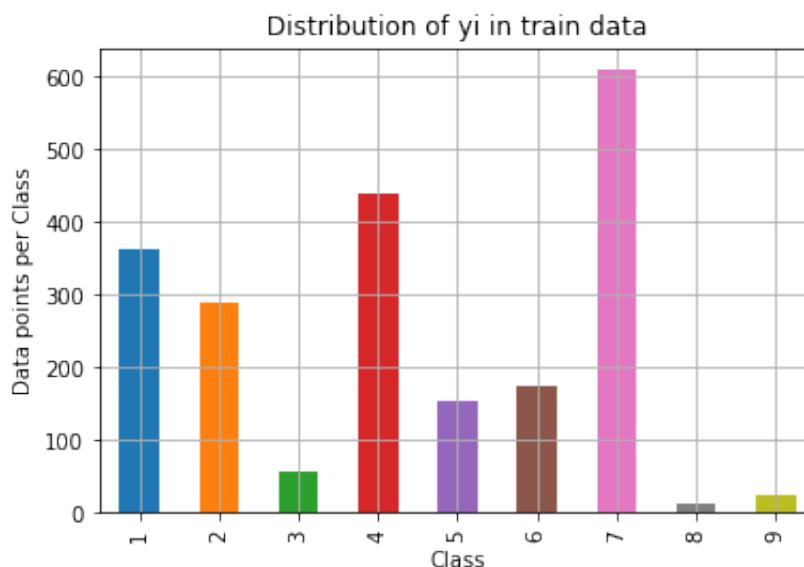
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in descending 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_distrib

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()

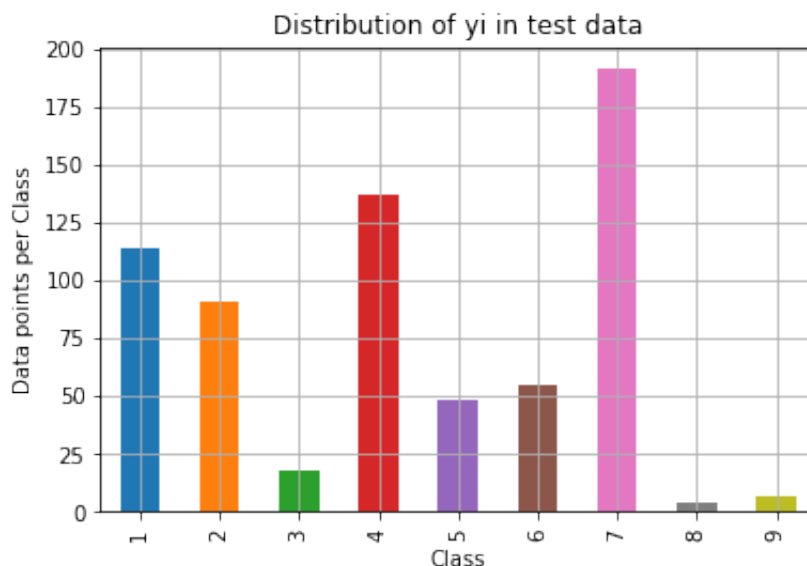
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in descending order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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

```

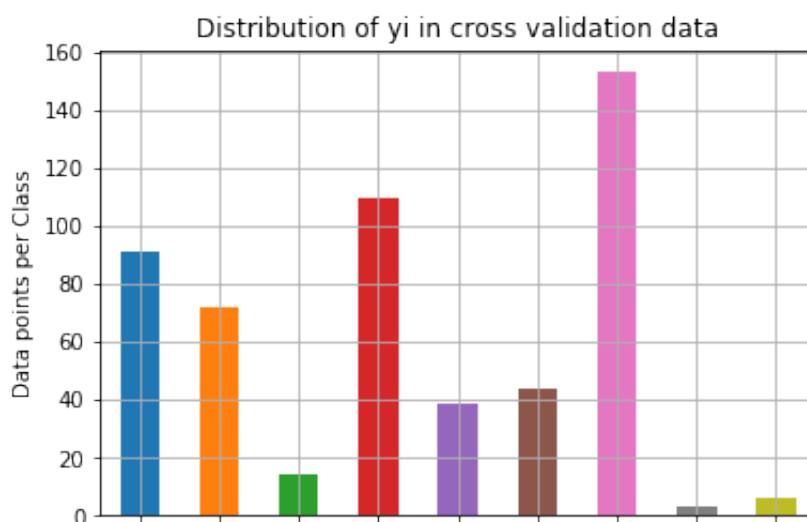


Number of data points in class 7 : 609 (28.672 %)

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 %)



1	2	3	4	5	6	7	8	9
				Class				

```

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

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

    # C = [[1, 2],
    #      [3, 4]]
    # C.sum(axis = 0) axis=0 corresponds to columns and axis=1 corresponds
    # to rows
    # 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))

```

```

plt.figure(figsize=(20,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=1)
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=1)
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=1)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

```

```

In [15]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers
# 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))

# 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))

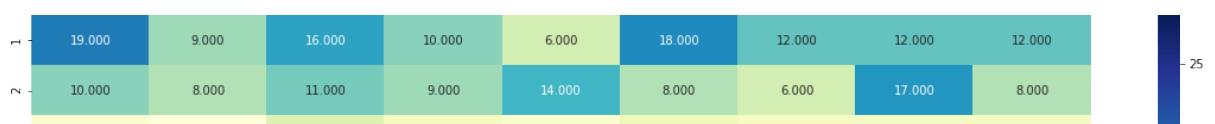
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

Log loss on Cross Validation Data using Random Model 2.4632994965176294

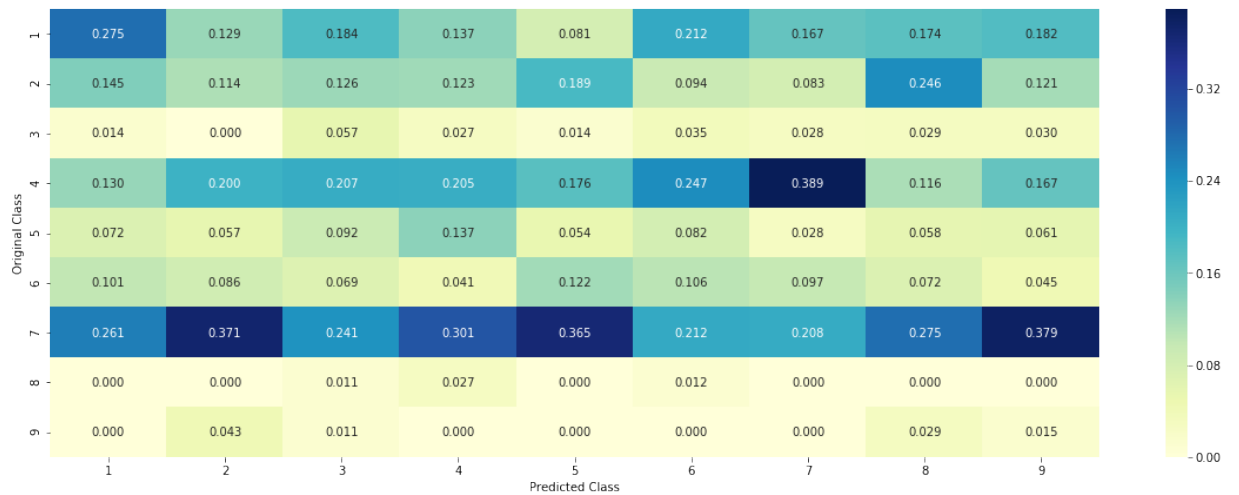
Log loss on Test Data using Random Model 2.5194527470490704

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

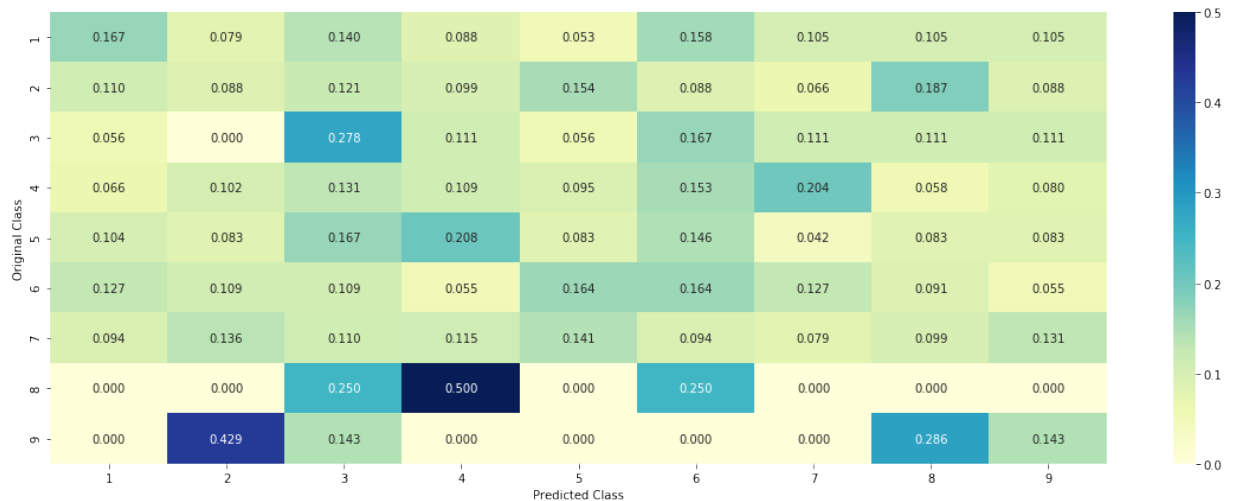




----- Precision matrix (Column Sum=1) -----



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



3.3 Univariate Analysis

```
In [16]: # code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# alqorithm
```

```

# -----
# Consider all unique values and the number of occurrences of given feature
# build a vector (1*9) , the first element = (number of times it occurs)
# gv_dict is like a look up table, for every gene it stores a (1*9) representation
# 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
    gv_dict = dict()

    # denominator will contain the number of times that particular feature occurs
    for i, denominator in value_count.items():
        # vec will contain (p(yi=1/Gi) probability of gene/variation)
        # vec is 9 dimensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']==i)]['Variation'])
            #          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['Class']!=k)]

# cls_cnt.shape[0](numerator) will contain the number of t
vec.append((cls_cnt.shape[0] + alpha*10)/(denominator + 90*alpha))

# we are adding the gene/variation to the dict as key and vec
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, 0.06818181818181818],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366],
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284728472847284],
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333333333333333],
    # ...
    # }
    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
    gv_fea = []
    # for every feature values in the given data frame we will check if
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    # gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea

```

when we 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 [17]: 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 : 232

BRCA1 168

TP53 106

EGFR 96

BRCA2 80

PTEN 74

KIT 63

BRAF 62

PDGFRA 45

ERBB2 40

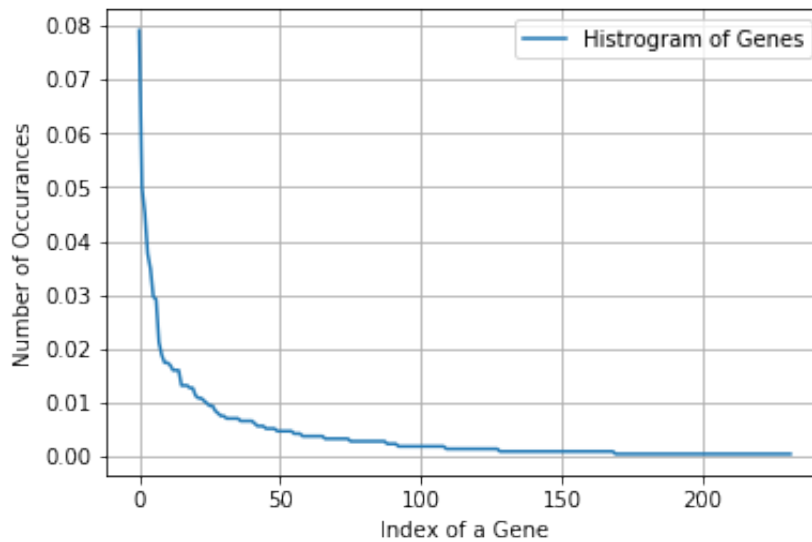
CDKN2A 37

Name: Gene, dtype: int64

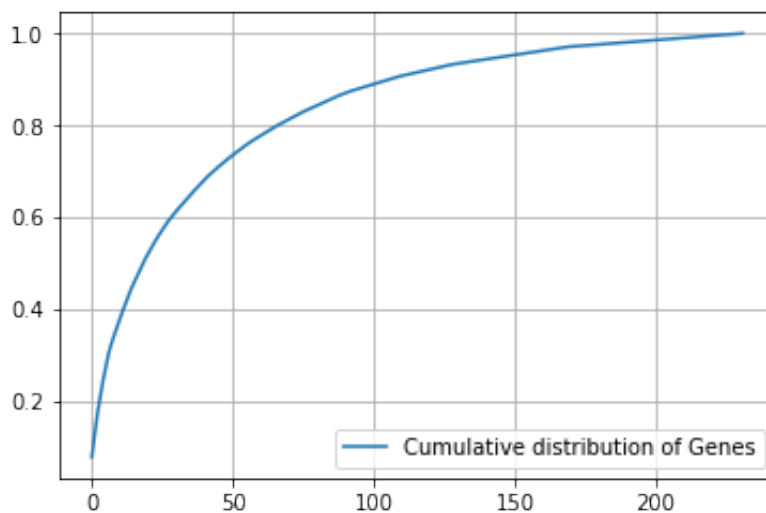
```
In [18]: print("Ans: There are", unique_genes.shape[0], "different categories of")
```

Ans: There are 232 different categories of genes in the train data,
and they are distributed as follows

```
In [19]: 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 [20]: c = np.cumsum(h)  
plt.plot(c, label='Cumulative distribution of Genes')  
plt.grid()  
plt.legend()  
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>
[\(https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/\)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

1. One hot Encoding
2. Response coding

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

```
In [21]: #response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene"))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene"))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene"))
```

```
In [22]: print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature: (2124, 9)")

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

```
In [23]: # one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer() #Using bi-grams

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

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

```
Out[24]: 2293      STAT3
2618      BRCA1
2741      BRAF
2823      BRCA2
2918      NFE2L2
Name: Gene, dtype: object
```

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

```
Out[25]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1a',
          'arid1b',
          'arid2',
          'arid5b',
          'asxl2',
          'atm',
          'atr',
          'atrx',
          'aurka',
          ...]
```

```
In [26]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 231)")
```

train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 231)

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

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

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

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

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

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

```

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=0)
    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_))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_))

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=0)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

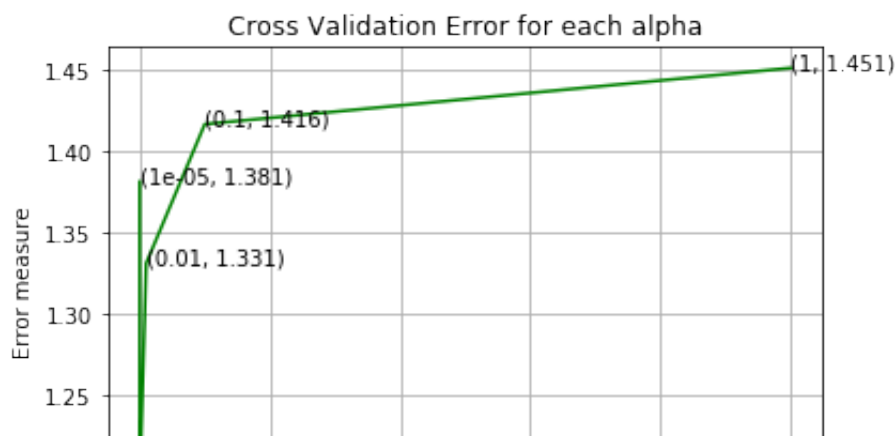
predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_))

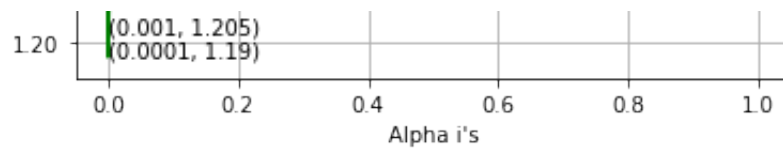
```

```

For values of alpha = 1e-05 The log loss is: 1.380687529785292
For values of alpha = 0.0001 The log loss is: 1.1904722298571981
For values of alpha = 0.001 The log loss is: 1.2054797578448564
For values of alpha = 0.01 The log loss is: 1.3305395439864576
For values of alpha = 0.1 The log loss is: 1.416035221242707
For values of alpha = 1 The log loss is: 1.4505534469196093

```





For values of best alpha = 0.0001 The train log loss is: 1.0402535763384477

For values of best alpha = 0.0001 The cross validation log loss is: 1.1904722298571981

For values of best alpha = 0.0001 The test log loss is: 1.2234370651329984

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

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

```
In [28]: print("Q6. How many data points in Test and CV datasets are covered by
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],
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0])
```

Q6. How many data points in Test and CV datasets are covered by the 232 genes in train dataset?

Ans

1. In test data 643 out of 665 : 96.69172932330827

2. In cross validation data 512 out of 532 : 96.2406015037594

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

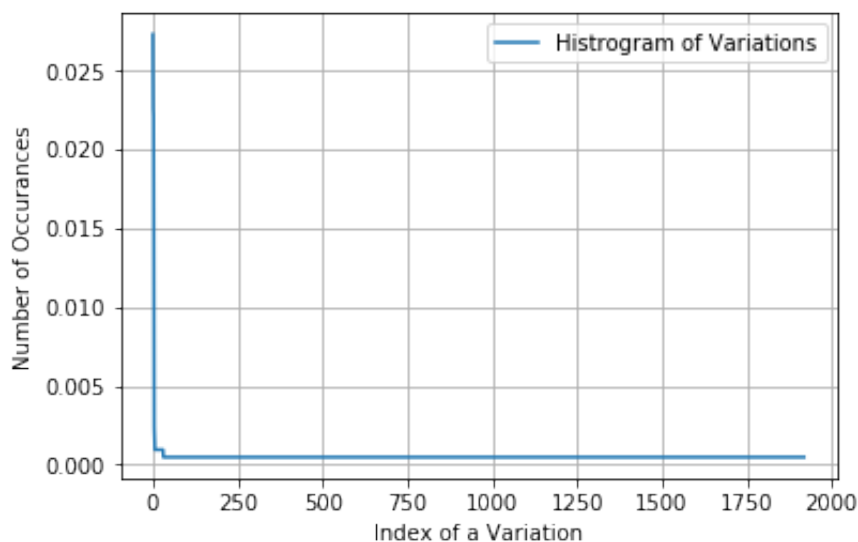
```
In [29]: unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occurred most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1919
Truncating_Mutations      58
Amplification              48
Deletion                   47
Fusions                    25
Overexpression             5
Q61H                       3
T58I                       2
Q61K                       2
EWSR1-ETV1_Fusion         2
R841K                      2
Name: Variation, dtype: int64
```

```
In [30]: print("Ans: There are", unique_variations.shape[0] , "different categories")
```

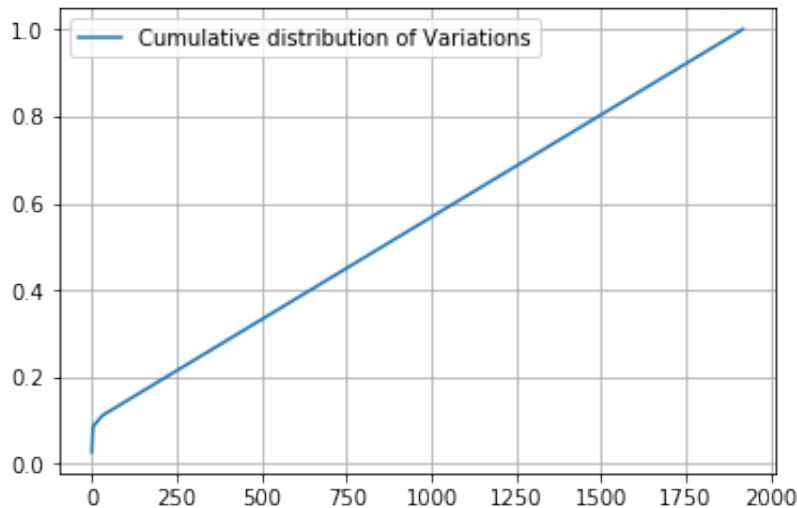
Ans: There are 1919 different categories of variations in the train data, and they are distributed as follows

```
In [31]: 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 [32]: c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()

[0.02730697 0.04990584 0.0720339 ... 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/>
[\(https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/\)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

1. One hot Encoding
2. Response coding

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

```
In [33]: # alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "
```



```
In [34]: print("train_variation_feature_responseCoding is a converted feature using  
train_variation_feature_responseCoding is a converted feature using  
the response coding method. The shape of Variation feature: (2124, 9  
)
```

```
In [35]: # one-hot encoding of variation feature.  
variation_vectorizer = TfidfVectorizer() #Using bi-grams  
  
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_variation_feature_responseCoding)  
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_variation_feature_responseCoding)  
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_variation_feature_responseCoding)
```

```
In [36]: print("train_variation_feature_onehotEncoded is converted feature using  
train_variation_feature_onehotEncoded is converted feature using the  
one-hot encoding method. The shape of Variation feature: (2124, 195  
2)
```

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

Let's build a model just like the earlier!

```
In [37]: alpha = [10 ** x for x in range(-5, 1)]  
  
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html  
# -----  
# default parameters  
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,  
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, class_weight=None, warm_start=False, average=False, n_iter=None)  
  
# some of methods  
# fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.  
# predict(X) Predict class labels for samples in X.  
  
#-----  
# video link:  
#-----  
  
cv_log_error_array=[]  
for i in alpha:  
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=None)  
    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_))
```

```

cv_log_error_array.append(log_loss(y_cv, predict_y, labels=labels))
print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv

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

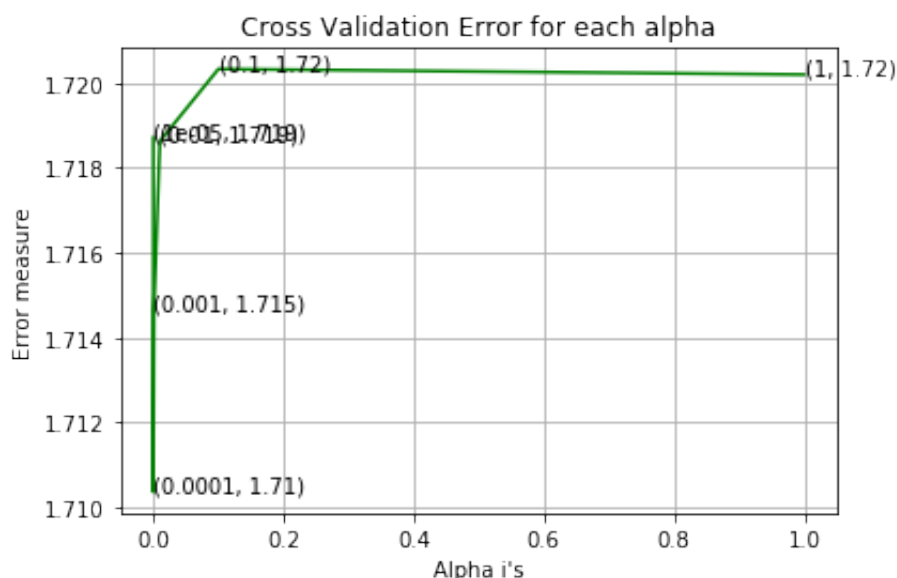
predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log

```

```

For values of alpha = 1e-05 The log loss is: 1.7187121836118737
For values of alpha = 0.0001 The log loss is: 1.710355540683771
For values of alpha = 0.001 The log loss is: 1.7146544947359295
For values of alpha = 0.01 The log loss is: 1.718624341597116
For values of alpha = 0.1 The log loss is: 1.7203261406626855
For values of alpha = 1 The log loss is: 1.7201945659105753

```



```

For values of best alpha = 0.0001 The train log loss is: 0.74346292
45510527
For values of best alpha = 0.0001 The cross validation log loss is:

```

1.710355540683771

For values of best alpha = 0.0001 The test log loss is: 1.701296564989786

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

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

```
In [38]: print("Q12. How many data points are covered by total ", unique_variat.
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Var
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],
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape
```

Q12. How many data points are covered by total 1919 genes in test and cross validation data sets?

Ans

1. In test data 61 out of 665 : 9.172932330827068

2. In cross validation data 52 out of 532 : 9.774436090225564

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

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

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

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

```
In [41]: # building a CountVectorizer with all the words that occurred minimum 3
text_vectorizer = TfidfVectorizer() #Using bi-grams

train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_
# 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
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 num
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count

print("Total number of unique words in train data :", len(train_text_f

Total number of unique words in train data : 126314
```

```
In [42]: 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 [43]: #response coding of text features
train_text_feature_responseCoding = get_text_responsecoding(train_df)
test_text_feature_responseCoding = get_text_responsecoding(test_df)
cv_text_feature_responseCoding = get_text_responsecoding(cv_df)
```

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

```
In [45]: # don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding)

# 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)

# 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)
```

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

```
Counter({0.0016241460067415996: 724, 0.0017742518788684777: 632, 0.0023811499886211775: 586, 0.01215265333032062: 447, 0.0045472234145204045: 435, 0.004706963801623958: 424, 0.0019076721867479663: 418, 0.0026774193727881527: 390, 0.001740668570665917: 358, 0.0020538730599307603: 347, 0.0013810780305760924: 342, 0.006876297234983572: 340, 0.011828800030378658: 318, 0.003206454220793202: 313, 0.008561776461906387: 312, 0.002894144597605948: 309, 0.018973089296599084: 303, 0.009779050929874977: 281, 0.006311576755229421: 279, 0.0031691269469084735: 279, 0.002161241405003483: 277, 0.009117232070461192: 267, 0.007350436853733815: 256, 0.002640276642578047: 256, 0.004049170909958916: 254, 0.004116494470329017: 252, 0.005904511667296492: 246, 0.00437678355403366: 243, 0.002196812737605799: 242, 0.0037182238975493336: 239, 0.007919908211507277: 233, 0.034927256804940694: 232, 0.005337112948446239: 231, 0.004476099624305302: 231, 0.007125740501085315: 228, 0.007711860693453143: 222, 0.00230228389758648: 221, 0.0054924375632370405: 215, 0.002453894457131122: 202, 0.07613621466410424: 201, 0.004622586903943541: 200, 0.003576582160589414: 198, 0.0025544468183422986: 195, 0.011241308734368655: 194, 0.002375131801556675: 193, 0.004739707896820621: 191, 0.0021545562712934783: 190, 0.0021173333333333333: 188, 0.0021313333333333333: 188, 0.0021566132750666667: 188})
```

http://localhost:8888/notebooks/Documents/cancer%20diagnosis/ms...ing-cancer-treatment/PersonalizedCancerDiagnosis%20tf-idf.ipynb Page 30 of 80

```

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_arr
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',
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

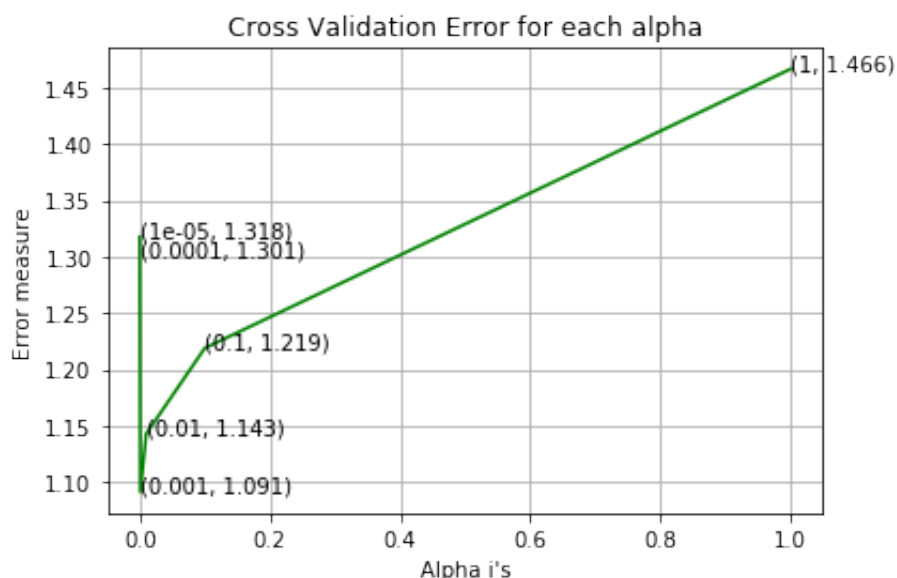
predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log

```

```

For values of alpha = 1e-05 The log loss is: 1.317654082361611
For values of alpha = 0.0001 The log loss is: 1.301297311457381
For values of alpha = 0.001 The log loss is: 1.0913506133869442
For values of alpha = 0.01 The log loss is: 1.1429743932121128
For values of alpha = 0.1 The log loss is: 1.2189599178008097
For values of alpha = 1 The log loss is: 1.4664021945584043

```



```

For values of best alpha = 0.001 The train log loss is: 0.652859786
1756431
For values of best alpha = 0.001 The cross validation log loss is:
1.0913506133869442
For values of best alpha = 0.001 The test log loss is: 1.1922718782

```

473933

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

Ans. Yes, it seems like!

```
In [49]: def get_intersec_text(df):
          df_text_vec = TfidfVectorizer() #Using bi-grams

          df_text_fea = df_text_vec.fit_transform(df['TEXT'])
          df_text_features = df_text_vec.get_feature_names()

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

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

77.901 % of word of test data appeared in train data

81.661 % of word of Cross Validation appeared in train data

4. Machine Learning Models

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

          #Misc. fonctionns for ML models

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

          # for calculating log_loss we willl provide the array of probabilities
          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)))
          plot_confusion_matrix(test_y, pred_y)
```



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

```
In [64]: # this function will be used just for naive bayes
        # for the given indices, we will print the name of the features
        # and we will check whether the feature present in the test point text
        def get_impfeature_names(indices, text, gene, var, no_features):
            gene_count_vec = TfidfVectorizer()
            var_count_vec = TfidfVectorizer()
            text_count_vec = TfidfVectorizer()

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

            print("Out of the top ",no_features," features ", word_present, " are present")
```

Stacking the three types of features

```

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

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding, test_gene_var_onehotCoding)
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding, cv_gene_var_onehotCoding)

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

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

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

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding, train_gene_var_onehotCoding)
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding, test_gene_var_onehotCoding)
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding, cv_gene_var_onehotCoding)

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 = ",
print("(number of data points * number of features) in test data = ",
print("(number of data points * number of features) in cross validation data = ")

```

```

One hot encoding features :
(number of data points * number of features) in train data = (2124, 128497)
(number of data points * number of features) in test data = (665, 128497)
(number of data points * number of features) in cross validation data = (532, 128497)

```

```
In [56]: print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
print("(number of data points * number of features) in test data = ",
print("(number of data points * number of features) in cross validation data = ")
```

```
Response encoding features :
(number of data points * number of features) in train data = (2124, 27)
(number of data points * number of features) in test data = (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=5)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100, 1000]
```

```

alpha = [0.00001, 0.00001, 0.0001, 0.001, 0.01, 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_))
    # to avoid rounding error while multiplying probabilities we use log
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is: ", log_loss(train_y, predict_y, labels=clf.classes_))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is: ", log_loss(cv_y, predict_y, labels=clf.classes_))
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(test_y, predict_y, labels=clf.classes_))

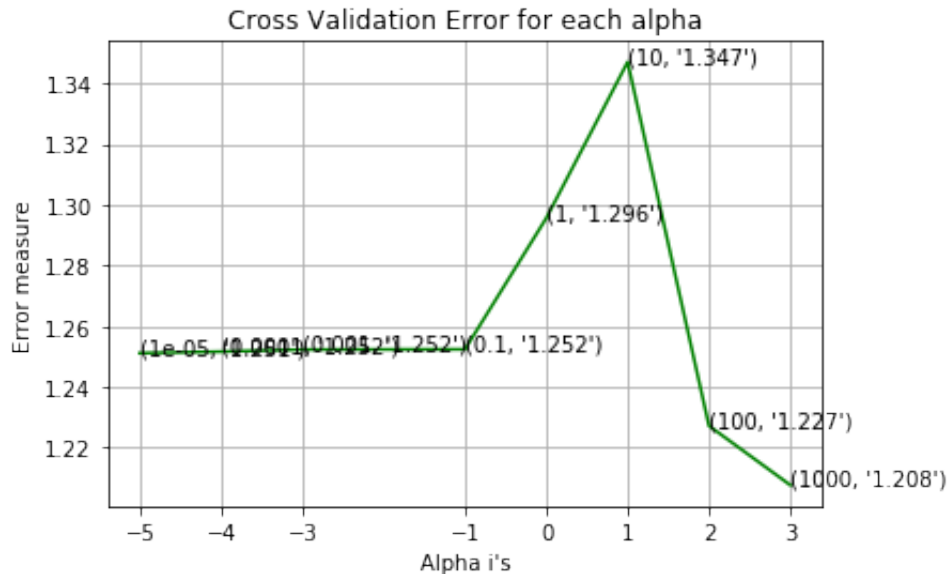
```

```

for alpha = 1e-05
Log Loss : 1.2511280991459093
for alpha = 0.0001
Log Loss : 1.251637247281397
for alpha = 0.001
Log Loss : 1.2523092800419529
for alpha = 0.1
Log Loss : 1.2523984842282103
for alpha = 1
Log Loss : 1.2956117206731583
for alpha = 10
Log Loss : 1.347007457591381
for alpha = 100
Log Loss : 1.2272110083914034

```

for alpha = 1000
Log Loss : 1.2076840489134355



For values of best alpha = 1000 The train log loss is: 0.8302960503094823

For values of best alpha = 1000 The cross validation log loss is: 1.2076840489134355

For values of best alpha = 1000 The test log loss is: 1.244895873946656

4.1.1.2. Testing the model with best hyper paramters

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

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to training set X and class labels y.
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vectors X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=5)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Return probability estimates for the test vectors X.
```

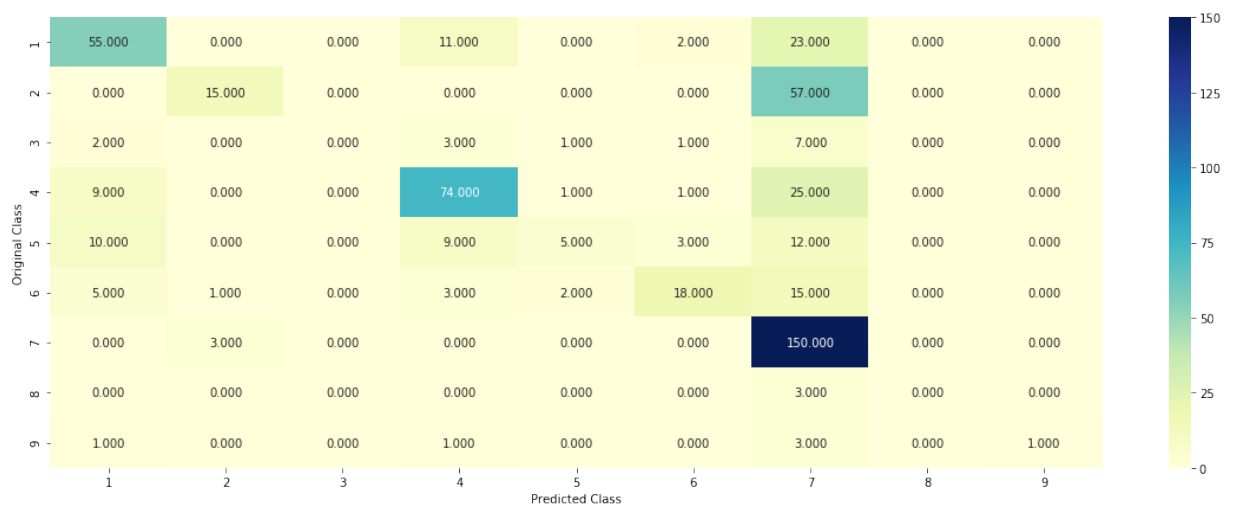
```
# predict_proba(x) POSTERIOR PROBABILITIES OF CLASSIFICATION
# -----

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

Log Loss : 1.2076840489134355

Number of missclassified point : 0.40225563909774437

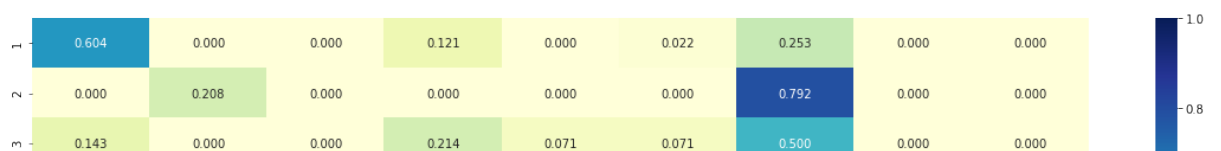
----- Confusion matrix -----

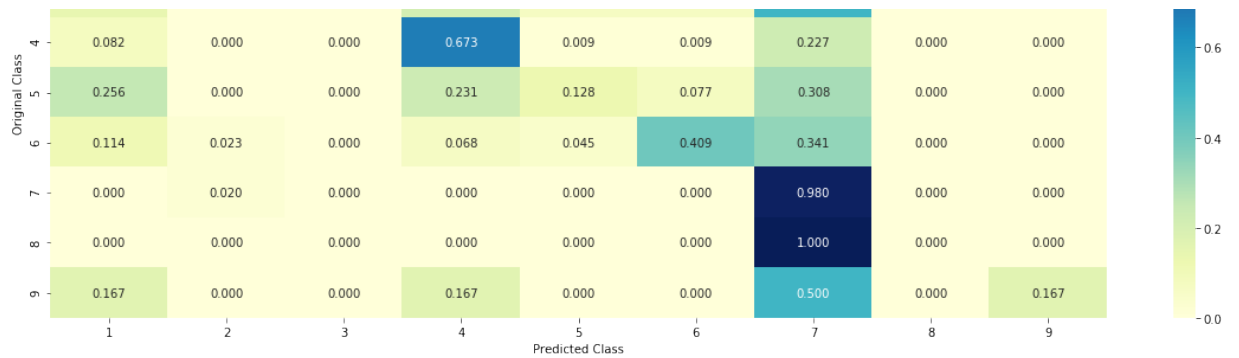


----- Precision matrix (Column Sum=1) -----



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





4.1.1.3. Feature Importance, Correctly classified point

```
In [69]: test_point_index = 20
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(
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])
```

Predicted Class : 4

Predicted Class Probabilities: [[0.142 0.04 0.0213 0.5814 0.0328
0.0272 0.1487 0.0035 0.0031]]

Actual Class : 4

```
-----
11 Text feature [proteins] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [activity] present in test data point [True]
14 Text feature [experiments] present in test data point [True]
15 Text feature [indicated] present in test data point [True]
16 Text feature [described] present in test data point [True]
17 Text feature [acid] present in test data point [True]
18 Text feature [whereas] present in test data point [True]
19 Text feature [shown] present in test data point [True]
20 Text feature [whether] present in test data point [True]
21 Text feature [determined] present in test data point [True]
25 Text feature [pten] present in test data point [True]
26 Text feature [results] present in test data point [True]
27 Text feature [two] present in test data point [True]
28 Text feature [function] present in test data point [True]
29 Text feature [retained] present in test data point [True]
30 Text feature [amino] present in test data point [True]
31 Text feature [important] present in test data point [True]
32 Text feature [loss] present in test data point [True]
33 Text feature [mammalian] present in test data point [True]
34 Text feature [vitro] present in test data point [True]
35 Text feature [type] present in test data point [True]
36 Text feature [bind] present in test data point [True]
37 Text feature [ability] present in test data point [True]
```

```

38 Text feature [also] present in test data point [True]
39 Text feature [missense] present in test data point [True]
40 Text feature [containing] present in test data point [True]
41 Text feature [reduced] present in test data point [True]
42 Text feature [functions] present in test data point [True]
43 Text feature [tagged] present in test data point [True]
44 Text feature [purified] present in test data point [True]
45 Text feature [indicate] present in test data point [True]
46 Text feature [wild] present in test data point [True]
47 Text feature [expressed] present in test data point [True]
48 Text feature [either] present in test data point [True]
49 Text feature [levels] present in test data point [True]
50 Text feature [transfected] present in test data point [True]
51 Text feature [tris] present in test data point [True]
52 Text feature [buffer] present in test data point [True]
53 Text feature [nacl] present in test data point [True]
54 Text feature [sds] present in test data point [True]
56 Text feature [incubated] present in test data point [True]
57 Text feature [mutations] present in test data point [True]
59 Text feature [partially] present in test data point [True]
60 Text feature [although] present in test data point [True]
61 Text feature [mm] present in test data point [True]
62 Text feature [determine] present in test data point [True]
63 Text feature [lower] present in test data point [True]
65 Text feature [expression] present in test data point [True]
68 Text feature [30] present in test data point [True]
69 Text feature [see] present in test data point [True]
70 Text feature [effects] present in test data point [True]
71 Text feature [standard] present in test data point [True]
72 Text feature [may] present in test data point [True]
74 Text feature [related] present in test data point [True]
75 Text feature [affected] present in test data point [True]
76 Text feature [vivo] present in test data point [True]
77 Text feature [abrogate] present in test data point [True]
78 Text feature [contribute] present in test data point [True]
82 Text feature [result] present in test data point [True]
84 Text feature [using] present in test data point [True]
85 Text feature [resulting] present in test data point [True]
86 Text feature [performed] present in test data point [True]
89 Text feature [transfection] present in test data point [True]
90 Text feature [thus] present in test data point [True]
91 Text feature [addition] present in test data point [True]
92 Text feature [previously] present in test data point [True]
93 Text feature [discussion] present in test data point [True]
94 Text feature [possible] present in test data point [True]
96 Text feature [suggest] present in test data point [True]
97 Text feature [fact] present in test data point [True]
99 Text feature [analyzed] present in test data point [True]
Out of the top 100 features 72 are present in query point

```

4.1.1.4. Feature Importance, Incorrectly classified point


```
In [70]: 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
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])
```

Predicted Class : 7

Predicted Class Probabilities: [[0.0792 0.2417 0.026 0.1081 0.049
0.0399 0.4489 0.004 0.0033]]

Actual Class : 2

```
-----
17 Text feature [activated] present in test data point [True]
19 Text feature [activation] present in test data point [True]
20 Text feature [kinase] present in test data point [True]
23 Text feature [presence] present in test data point [True]
24 Text feature [factor] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
27 Text feature [growth] present in test data point [True]
28 Text feature [also] present in test data point [True]
31 Text feature [shown] present in test data point [True]
32 Text feature [however] present in test data point [True]
35 Text feature [signaling] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [recently] present in test data point [True]
38 Text feature [found] present in test data point [True]
40 Text feature [compared] present in test data point [True]
41 Text feature [independent] present in test data point [True]
42 Text feature [addition] present in test data point [True]
43 Text feature [increased] present in test data point [True]
44 Text feature [similar] present in test data point [True]
45 Text feature [previously] present in test data point [True]
48 Text feature [constitutively] present in test data point [True]
51 Text feature [consistent] present in test data point [True]
52 Text feature [figure] present in test data point [True]
55 Text feature [tyrosine] present in test data point [True]
57 Text feature [mutations] present in test data point [True]
59 Text feature [well] present in test data point [True]
63 Text feature [mutant] present in test data point [True]
65 Text feature [showed] present in test data point [True]
68 Text feature [may] present in test data point [True]
69 Text feature [increase] present in test data point [True]
70 Text feature [observed] present in test data point [True]
74 Text feature [furthermore] present in test data point [True]
76 Text feature [absence] present in test data point [True]
79 Text feature [using] present in test data point [True]
82 Text feature [recent] present in test data point [True]
83 Text feature [reported] present in test data point [True]
85 Text feature [including] present in test data point [True]
89 Text feature [respectively] present in test data point [True]
90 Text feature [fig] present in test data point [True]
```

```

91 Text feature [confirmed] present in test data point [True]
95 Text feature [either] present in test data point [True]
96 Text feature [mutation] present in test data point [True]
97 Text feature [role] present in test data point [True]
Out of the top 100 features 43 are present in query point

```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```

In [71]: # find more about KNeighborsClassifier() here http://scikit-learn.org/
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='au
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target v
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-----

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

alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf
    "

```

```

# to avoid rounding error while multiplying probabilities we use 10
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)

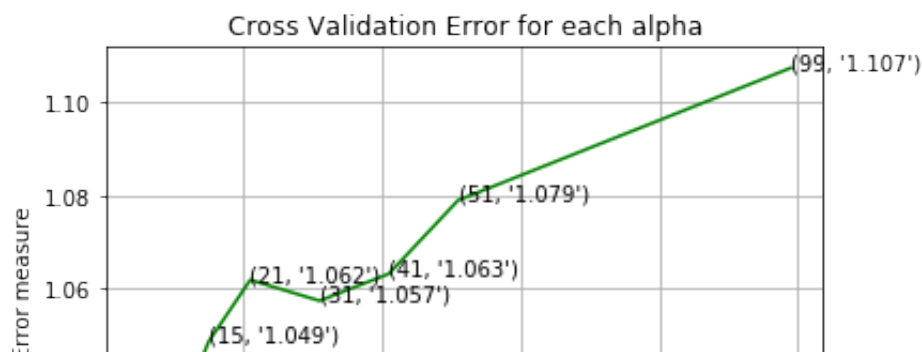
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log")
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val")
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log")

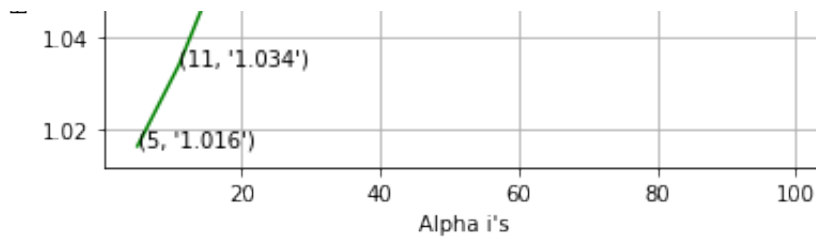
```

```

for alpha = 5
Log Loss : 1.0164770326205763
for alpha = 11
Log Loss : 1.0339537630266795
for alpha = 15
Log Loss : 1.0486375692718692
for alpha = 21
Log Loss : 1.0618684039948516
for alpha = 31
Log Loss : 1.0573819926781782
for alpha = 41
Log Loss : 1.0631807465680965
for alpha = 51
Log Loss : 1.0789721082207637
for alpha = 99
Log Loss : 1.107412809128586

```





For values of best alpha = 5 The train log loss is: 0.4775100726774
 55
 For values of best alpha = 5 The cross validation log loss is: 1.01
 64770326205763
 For values of best alpha = 5 The test log loss is: 1.11440255992177
 66

4.2.2. Testing the model with best hyper paramters

```
In [72]: # find more about KNeighborsClassifier() here http://scikit-learn.org/
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='au
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target v
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-----

clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_
```

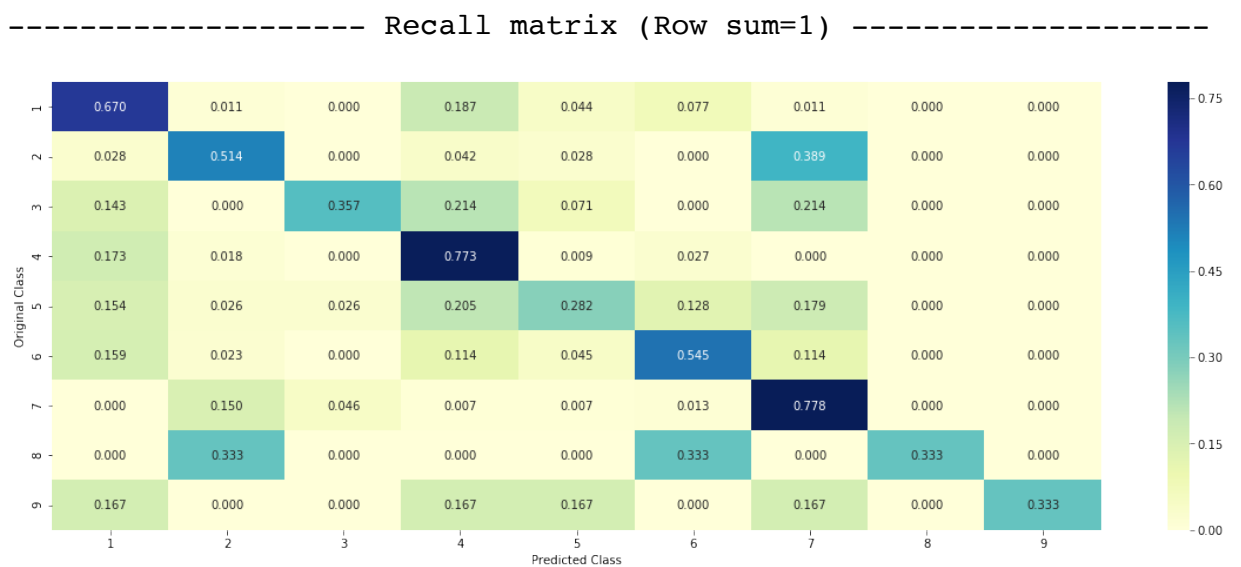
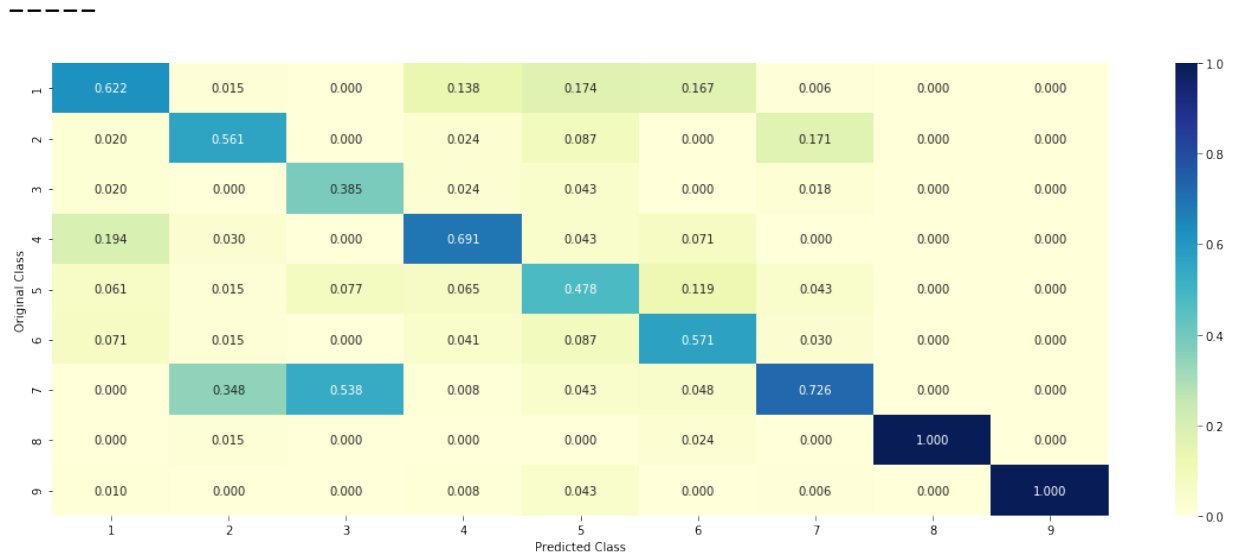
Log loss : 1.0164770326205763

Number of mis-classified points : 0.35150375939849626

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



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



4.2.3. Sample Query point -1

```
In [73]: 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].res
print("The ",alpha[best_alpha]," nearest neighbours of the test points
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))

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

4.2.4. Sample Query Point-2

```
In [74]: 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]
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].res
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
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 poin
ts belongs to classes [2 4 6 6 1]
Fequency of nearest points : Counter({6: 2, 2: 1, 4: 1, 1: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [75]:

```

# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-----

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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2')
    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
    # to avoid rounding error while multiplying probabilitites we use lo
    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 = SGDCClassifier(class_weight='balanced', 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)

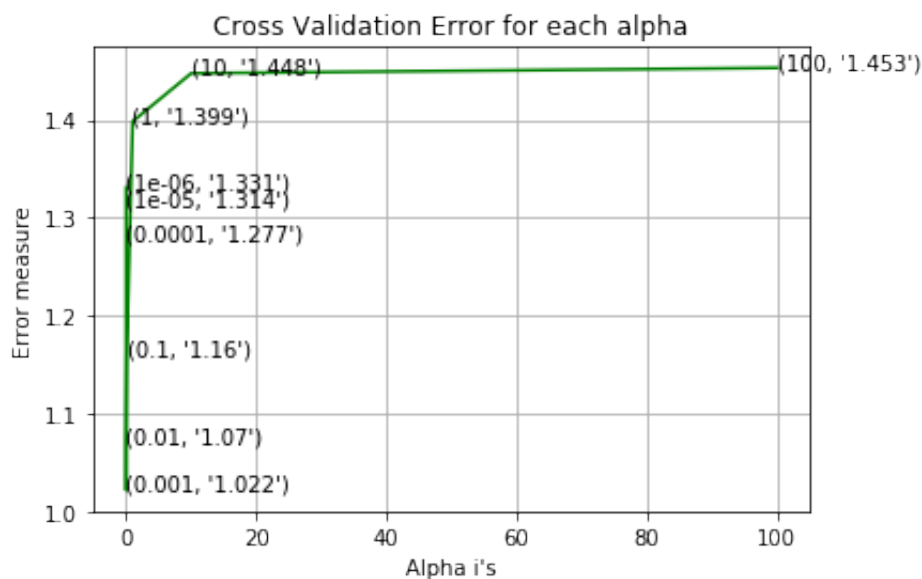
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log

```

```

for alpha = 1e-06
Log Loss : 1.3309241533336307
for alpha = 1e-05
Log Loss : 1.3135272357203416
for alpha = 0.0001
Log Loss : 1.2771176678822391
for alpha = 0.001
Log Loss : 1.021906075362836
for alpha = 0.01
Log Loss : 1.0701521103610392
for alpha = 0.1
Log Loss : 1.1599316290253257
for alpha = 1
Log Loss : 1.3985285500609437
for alpha = 10
Log Loss : 1.447701901263474
for alpha = 100
Log Loss : 1.4532523764823189

```



```

For values of best alpha = 0.001 The train log loss is: 0.583369008
0594884
For values of best alpha = 0.001 The cross validation log loss is:
1.021906075362836

```


For values of best alpha = 0.001 The test log loss is: 1.1218878639982124

4.3.1.2. Testing the model with best hyper paramters

```
In [76]: # read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_c
```

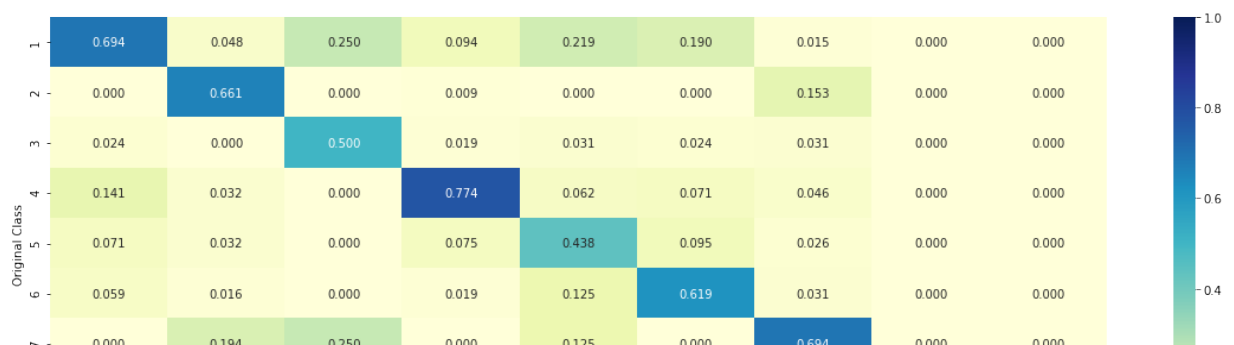
Log loss : 1.021906075362836

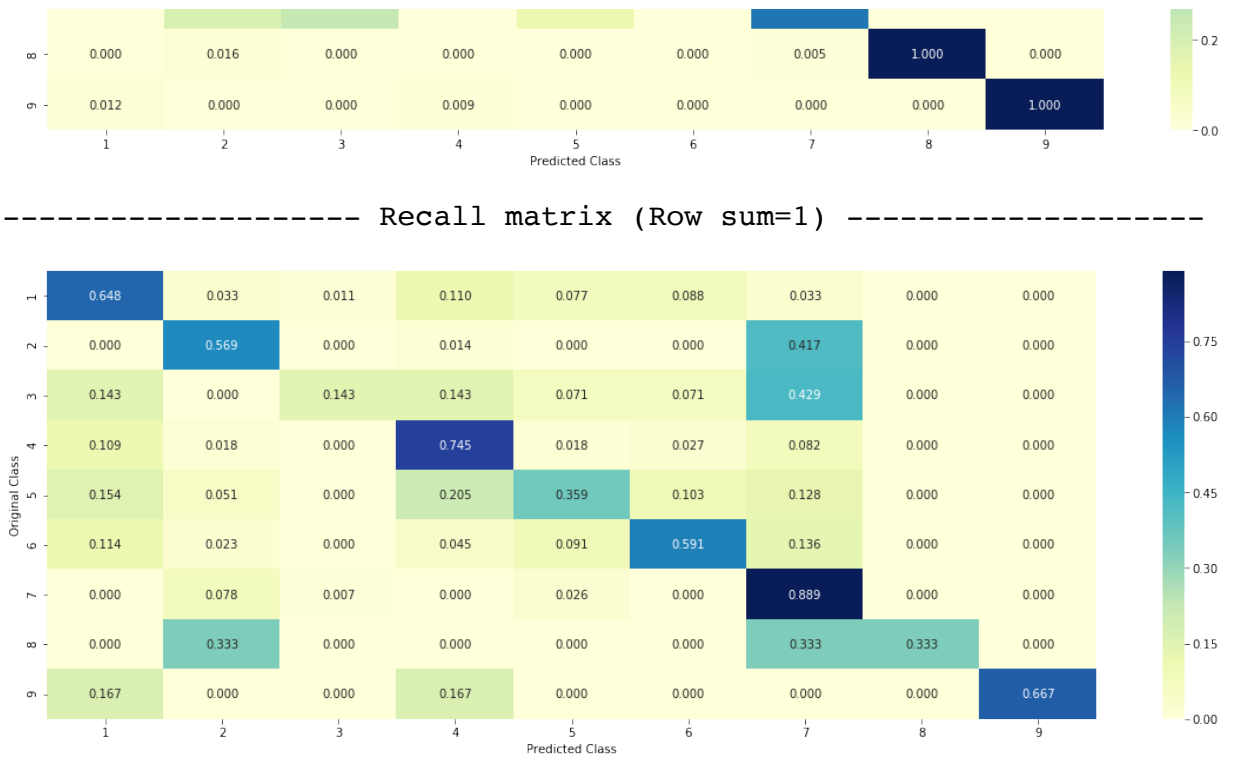
Number of mis-classified points : 0.31390977443609025

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



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





4.3.1.3. Feature Importance

```
In [77]: def get_imp_feature_names(text, indices, removed_ind = []):
word_present = 0
tabulte_list = []
increasingorder_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], word_present])
        increasingorder_ind += 1
print(word_present, "most important features are present in our query")
print("-"*50)
print("The features that are most important of the ", predicted_cls)
print(tabulate(tabulte_list, headers=["Index", "Feature name", "Proportion"])
```

4.3.1.3.1. Correctly Classified point

```
In [80]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha],
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 20
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
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])
```

Predicted Class : 4

Predicted Class Probabilities: [[0.0623 0.0513 0.0134 0.7445 0.0262
0.0143 0.0769 0.0056 0.0055]]

Actual Class : 4

```
-----
173 Text feature [traf2] present in test data point [True]
176 Text feature [cycloheximide] present in test data point [True]
182 Text feature [come] present in test data point [True]
233 Text feature [strengthen] present in test data point [True]
237 Text feature [precipitate] present in test data point [True]
239 Text feature [suppressor] present in test data point [True]
286 Text feature [pten] present in test data point [True]
360 Text feature [microscopy] present in test data point [True]
458 Text feature [supersignal] present in test data point [True]
463 Text feature [degrade] present in test data point [True]
497 Text feature [apparently] present in test data point [True]
Out of the top 500 features 11 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
In [81]: test_point_index = 23
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
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])
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0159 0.0178 0.0187 0.9125 0.0113
0.0046 0.0114 0.0046 0.0032]]
Actual Class : 3
-----
159 Text feature [germline] present in test data point [True]
229 Text feature [hamartoma] present in test data point [True]
239 Text feature [suppressor] present in test data point [True]
260 Text feature [cowden] present in test data point [True]
286 Text feature [pten] present in test data point [True]
334 Text feature [gl29r] present in test data point [True]
345 Text feature [gl29e] present in test data point [True]
360 Text feature [microscopy] present in test data point [True]
Out of the top 500 features 8 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```
In [82]: # read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.o
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
"
```

```

#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])           Get parameters for this estimator.
# predict(X)                    Predict the target of new samples.
# predict_proba(X)              Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=None)
    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_))
    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=None)
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(train_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is: ", log_loss(cv_y, predict_y))
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(test_y, predict_y))

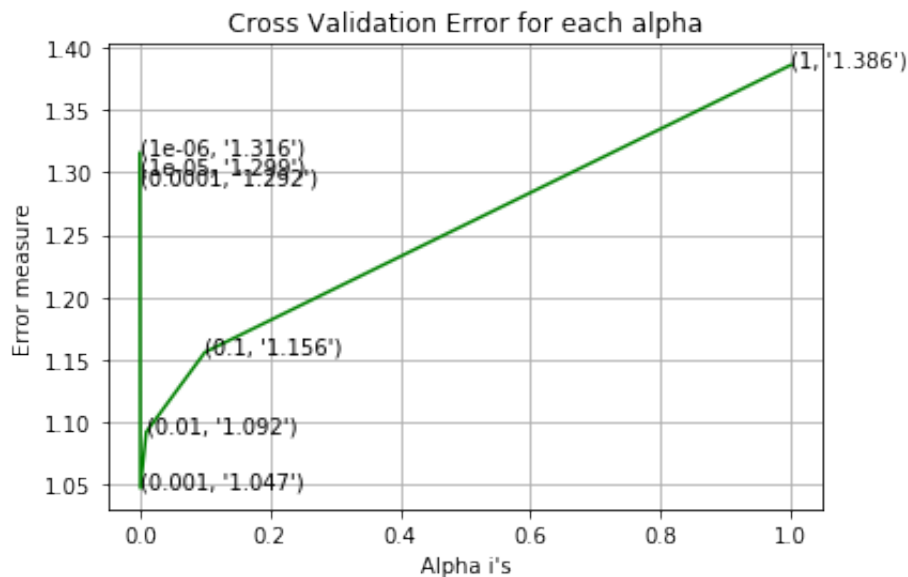
for alpha = 1e-06
Log Loss : 1.315665084862542
for alpha = 1e-05
Log Loss : 1.2990183038026888
for alpha = 0.0001
Log Loss : 1.2918751074166468
for alpha = 0.001
Log Loss : 1.046811494104433

```

```

for alpha = 0.01
Log Loss : 1.0918017199507595
for alpha = 0.1
Log Loss : 1.1557502760686225
for alpha = 1
Log Loss : 1.386269444170255

```



```

For values of best alpha = 0.001 The train log loss is: 0.580254125
4256129
For values of best alpha = 0.001 The cross validation log loss is:
1.046811494104433
For values of best alpha = 0.001 The test log loss is: 1.1484827308
926593

```

4.3.2.2. Testing model with best hyper parameters

```

In [83]: # read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_c

```

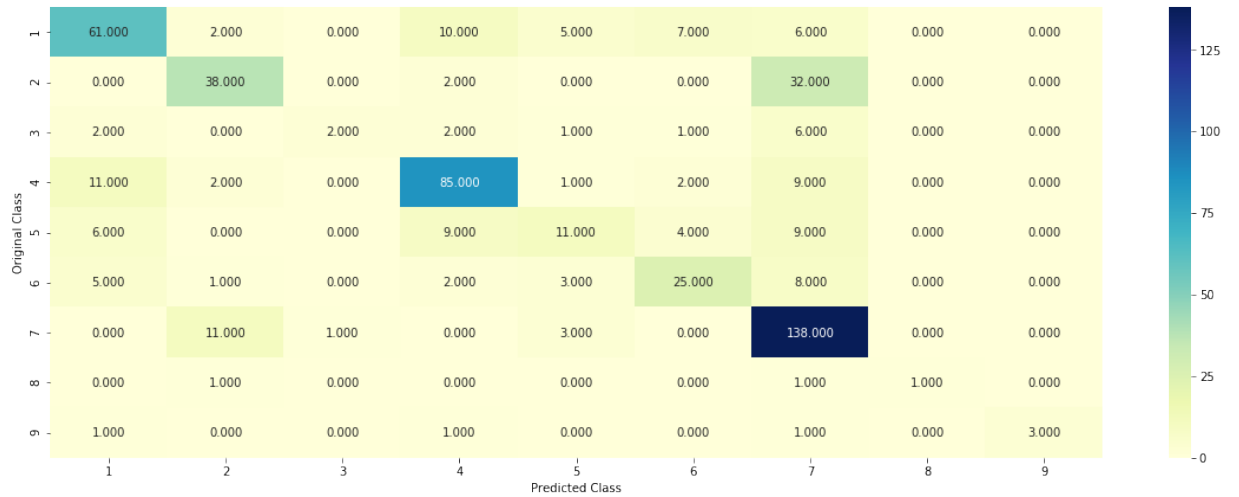
```

Log loss : 1.046811494104433

```

Number of mis-classified points : 0.3157894736842105

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



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



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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [86]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
      clf.fit(train_x_onehotCoding,train_y)
      test_point_index = 30
      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
      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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0621 0.0662 0.0096 0.7126 0.0343
0.0203 0.0883 0.0031 0.0035]]
Actual Class : 4
-----
167 Text feature [germline] present in test data point [True]
226 Text feature [suppressor] present in test data point [True]
270 Text feature [inactivating] present in test data point [True]
296 Text feature [young] present in test data point [True]
418 Text feature [dental] present in test data point [True]
464 Text feature [pten] present in test data point [True]
Out of the top 500 features 6 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [87]: 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
      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
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0379 0.3237 0.0038 0.0367 0.5583
0.0069 0.0299 0.0022 0.0007]]
Actual Class : 2
-----
331 Text feature [cystectomy] present in test data point [True]
Out of the top 500 features 1 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

```
In [88]: # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/linear_svm.html

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='raw')

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

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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
    # clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2')
    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_))
    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],
                    random_state=0)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

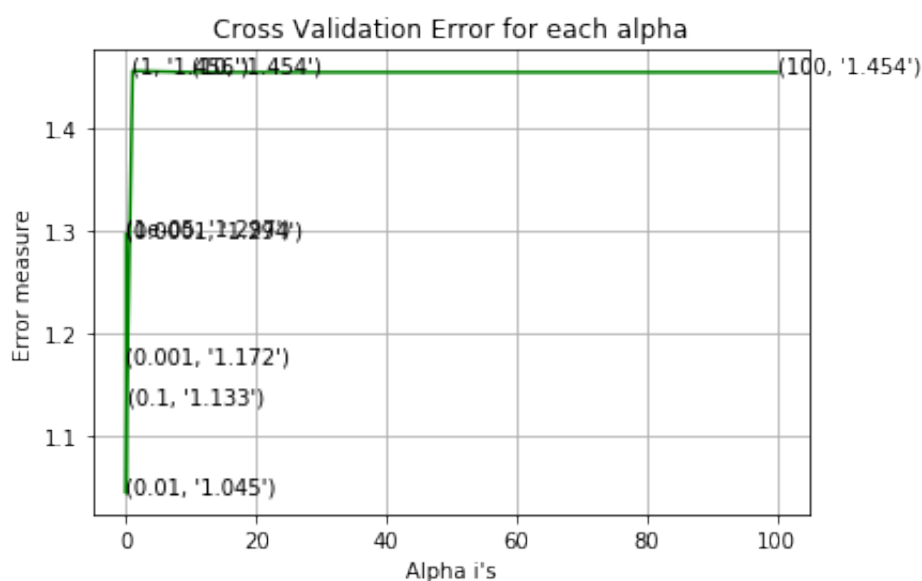
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is: ",
      sig_clf.loss_)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is: ",
      sig_clf.loss_)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is: ",
      sig_clf.loss_)

```

```

for C = 1e-05
Log Loss : 1.296645709421638
for C = 0.0001
Log Loss : 1.2943736784472655
for C = 0.001
Log Loss : 1.1720847970282955
for C = 0.01
Log Loss : 1.044608998084493
for C = 0.1
Log Loss : 1.133478945158311
for C = 1
Log Loss : 1.4557024737597957
for C = 10
Log Loss : 1.4543287762111374
for C = 100
Log Loss : 1.4543288029949224

```



For values of best alpha = 0.01 The train log loss is: 0.6785558661343886

For values of best alpha = 0.01 The cross validation log loss is: 1.044608998084493
 For values of best alpha = 0.01 The test log loss is: 1.1803869748509268

4.4.2. Testing model with best hyper parameters

```
In [89]: # read more about support vector machines with linear kernels here http://www.appliedaicourse.com/course/applied-ai-course

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_scaling=1, probability=True)

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

# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight=None)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding)
```

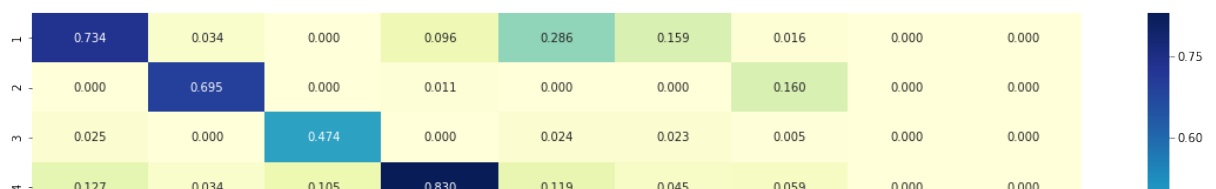
Log loss : 1.044608998084493

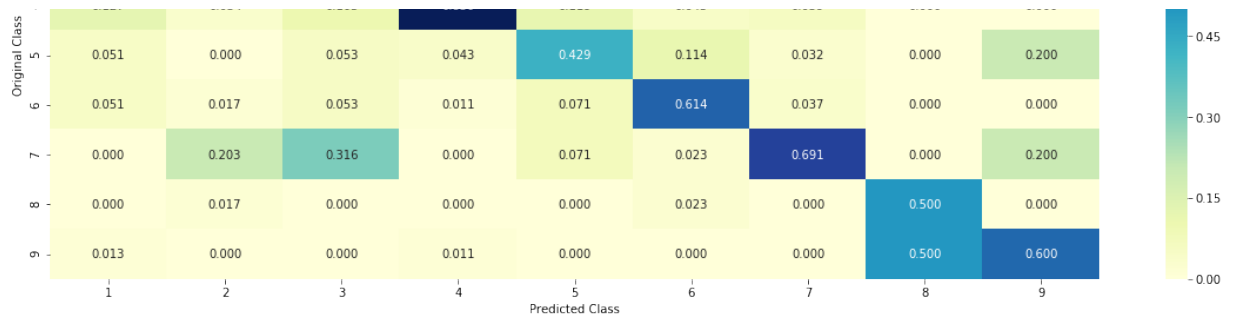
Number of mis-classified points : 0.31390977443609025

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

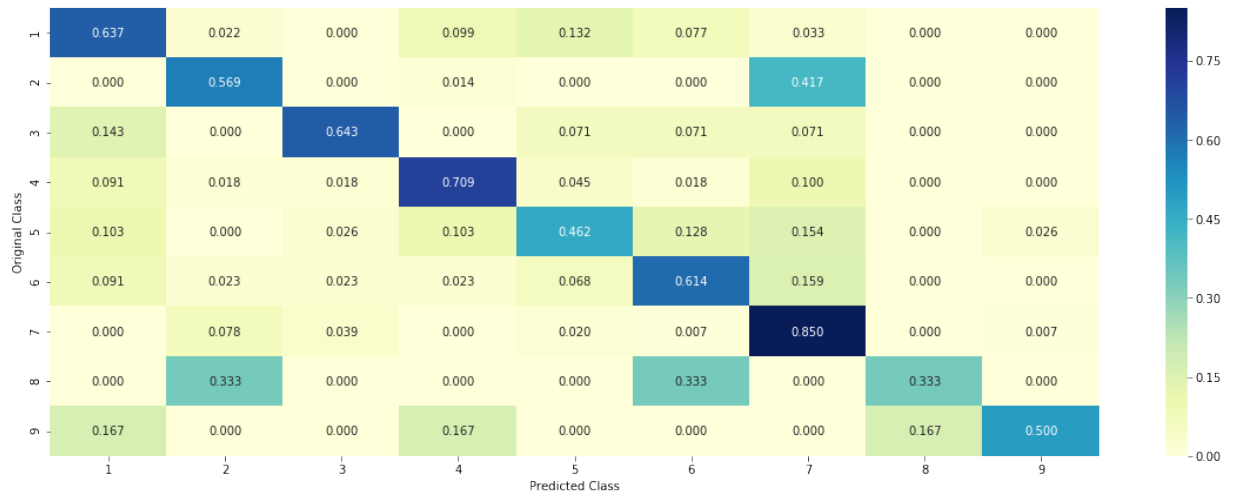


----- Precision matrix (Column Sum=1) -----





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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
In [90]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge')
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 25
# 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]), 5))
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])
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.035  0.0335 0.0087 0.8322 0.0235
0.0123 0.0483 0.0038 0.0027]]
Actual Class : 4
-----
20 Text feature [germline] present in test data point [True]
34 Text feature [suppressor] present in test data point [True]
92 Text feature [microscopy] present in test data point [True]
114 Text feature [hamartoma] present in test data point [True]
156 Text feature [pten] present in test data point [True]
163 Text feature [localization] present in test data point [True]
481 Text feature [trisphosphate] present in test data point [True]
482 Text feature [nonsense] present in test data point [True]
Out of the top 500 features 8 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
In [91]: 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]), 5))
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])
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.054  0.2815 0.012  0.0481 0.53
0.0085 0.0568 0.0056 0.0036]]
Actual Class : 2
-----
478 Text feature [vm] present in test data point [True]
Out of the top 500 features 1 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

```
In [92]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
# class_weight=None)

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

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

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
        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
        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]).r
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)), (featu
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)], cr
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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The

for n_estimators = 100 and max depth = 5
Log Loss : 1.278536413129168
for n_estimators = 100 and max depth = 10
Log Loss : 1.1936957783836775
for n_estimators = 200 and max depth = 5
Log Loss : 1.2672669742961473
for n_estimators = 200 and max depth = 10
Log Loss : 1.1823023628422664
for n_estimators = 500 and max depth = 5
Log Loss : 1.2594308891564394
for n_estimators = 500 and max depth = 10
Log Loss : 1.1749141280805797
for n_estimators = 1000 and max depth = 5
Log Loss : 1.2566089552316233
for n_estimators = 1000 and max depth = 10
Log Loss : 1.173769166650029
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2525757901929266
for n_estimators = 2000 and max depth = 10
Log Loss : 1.1727199691358927
For values of best estimator = 2000 The train log loss is: 0.679046
3535264001
For values of best estimator = 2000 The cross validation log loss i
s: 1.1727199691358925
For values of best estimator = 2000 The test log loss is: 1.1654561
540303974

```

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

```
In [93]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
# class_weight=None)

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

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

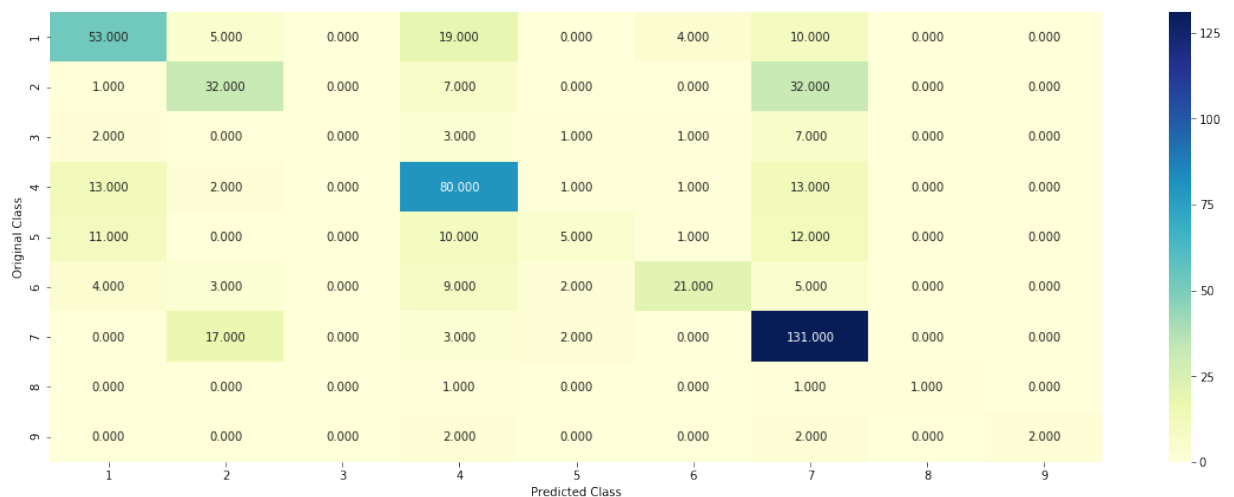
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cr
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
```

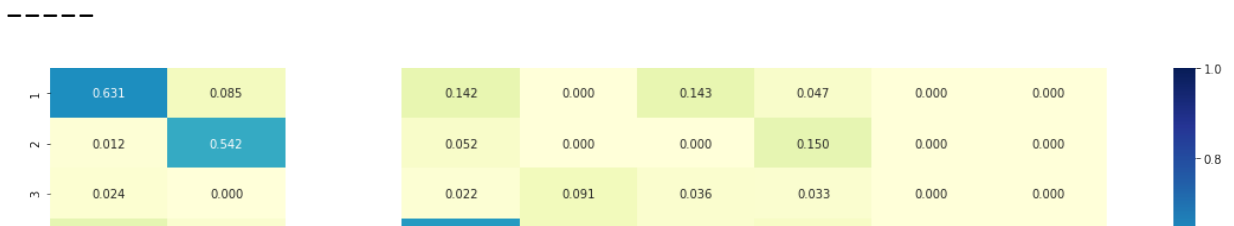
Log loss : 1.1727199691358925

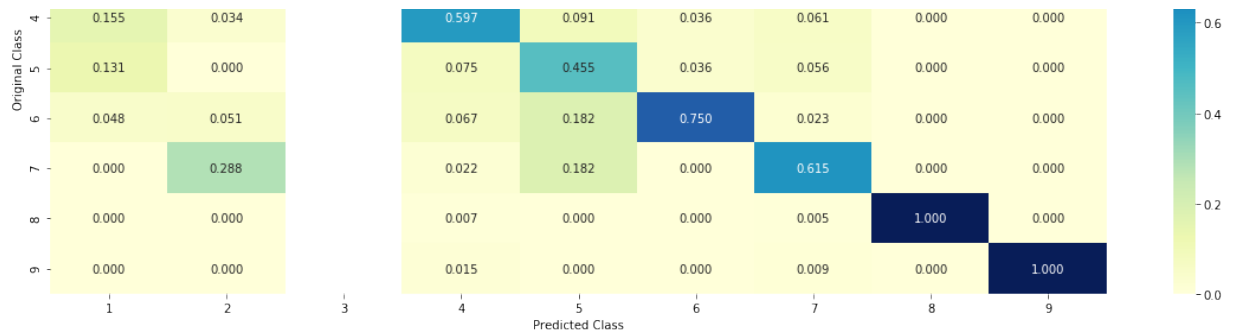
Number of mis-classified points : 0.3890977443609023

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

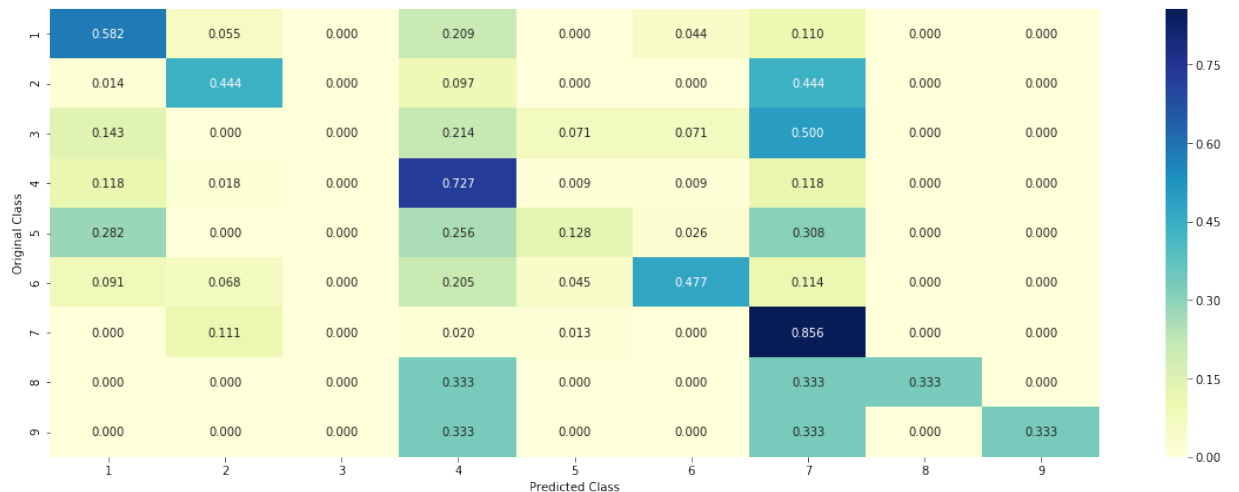


----- Precision matrix (Column Sum=1) -----





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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
In [95]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cr
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 25
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
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_p

Predicted Class : 4
Predicted Class Probabilities: [[0.0257 0.0148 0.0484 0.8378 0.0298
0.0222 0.0159 0.0027 0.0028]]
Actual Class : 4
```

```

-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] 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 [tyrosine] present in test data point [True]
10 Text feature [missense] present in test data point [True]
11 Text feature [oncogenic] present in test data point [True]
12 Text feature [signaling] present in test data point [True]
13 Text feature [function] present in test data point [True]
15 Text feature [growth] present in test data point [True]
17 Text feature [therapy] present in test data point [True]
19 Text feature [downstream] present in test data point [True]
22 Text feature [kinases] present in test data point [True]
23 Text feature [suppressor] present in test data point [True]
25 Text feature [loss] present in test data point [True]
29 Text feature [nonsense] present in test data point [True]
30 Text feature [cells] present in test data point [True]
32 Text feature [akt] present in test data point [True]
35 Text feature [pathogenic] present in test data point [True]
36 Text feature [phospho] present in test data point [True]
37 Text feature [protein] present in test data point [True]
38 Text feature [deleterious] present in test data point [True]
40 Text feature [stability] present in test data point [True]
42 Text feature [lines] present in test data point [True]
43 Text feature [inhibition] present in test data point [True]
45 Text feature [patients] present in test data point [True]
57 Text feature [affected] present in test data point [True]
58 Text feature [active] present in test data point [True]
59 Text feature [functional] present in test data point [True]
61 Text feature [expressing] present in test data point [True]
69 Text feature [yeast] present in test data point [True]
70 Text feature [cell] present in test data point [True]
72 Text feature [phosphorylated] present in test data point [True]
80 Text feature [pten] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
90 Text feature [retained] present in test data point [True]
96 Text feature [activity] present in test data point [True]
97 Text feature [information] present in test data point [True]
99 Text feature [presence] present in test data point [True]
Out of the top 100 features 39 are present in query point

```

4.5.3.2. Inorrectly Classified point

```

In [96]: 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
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_p
```

Predicted Class : 7

Predicted Class Probabilities: [[0.0606 0.3002 0.0252 0.0532 0.0959
0.044 0.4083 0.0063 0.0064]]

Actual Class : 2

```
-----
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 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 [tyrosine] present in test data point [True]
8 Text feature [treatment] present in test data point [True]
10 Text feature [missense] present in test data point [True]
12 Text feature [signaling] present in test data point [True]
13 Text feature [function] present in test data point [True]
14 Text feature [inhibitor] present in test data point [True]
15 Text feature [growth] present in test data point [True]
17 Text feature [therapy] present in test data point [True]
18 Text feature [receptor] present in test data point [True]
19 Text feature [downstream] present in test data point [True]
20 Text feature [treated] present in test data point [True]
21 Text feature [drug] present in test data point [True]
22 Text feature [kinases] present in test data point [True]
24 Text feature [therapeutic] present in test data point [True]
27 Text feature [inhibited] present in test data point [True]
29 Text feature [nonsense] present in test data point [True]
30 Text feature [cells] present in test data point [True]
31 Text feature [brca1] present in test data point [True]
32 Text feature [akt] present in test data point [True]
34 Text feature [variants] present in test data point [True]
37 Text feature [protein] present in test data point [True]
39 Text feature [proliferation] present in test data point [True]
42 Text feature [lines] present in test data point [True]
43 Text feature [inhibition] present in test data point [True]
45 Text feature [patients] present in test data point [True]
46 Text feature [extracellular] present in test data point [True]
53 Text feature [nsclc] present in test data point [True]
54 Text feature [survival] present in test data point [True]
55 Text feature [ic50] present in test data point [True]
56 Text feature [respond] present in test data point [True]
57 Text feature [affected] present in test data point [True]
59 Text feature [functional] present in test data point [True]
60 Text feature [egfr] present in test data point [True]
61 Text feature [expressing] present in test data point [True]
63 Text feature [advanced] present in test data point [True]
64 Text feature [erk1] present in test data point [True]
65 Text feature [mapk] present in test data point [True]
66 Text feature [ras] present in test data point [True]
67 Text feature [factor] present in test data point [True]
68 Text feature [sensitivity] present in test data point [True]
70 Text feature [cell] present in test data point [True]
72 Text feature [phosphorylated] present in test data point [True]
```

```

74 Text feature [response] present in test data point [True]
76 Text feature [efficacy] present in test data point [True]
77 Text feature [effective] present in test data point [True]
79 Text feature [potential] present in test data point [True]
81 Text feature [clinical] present in test data point [True]
82 Text feature [ligand] present in test data point [True]
83 Text feature [serum] present in test data point [True]
86 Text feature [metastatic] present in test data point [True]
88 Text feature [receptors] present in test data point [True]
89 Text feature [enhanced] present in test data point [True]
90 Text feature [retained] present in test data point [True]
93 Text feature [amplification] present in test data point [True]
96 Text feature [activity] present in test data point [True]
99 Text feature [presence] present in test data point [True]
Out of the top 100 features 61 are present in query point

```

4.5.3. Hyper paramter tuning (With Response Coding)

```

In [97]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
# class_weight=None)

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

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

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

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

```

```

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
        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))
        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]).r
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)), (featu
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)], cr
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 tr
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cr
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes

for n_estimators = 10 and max depth = 2
Log Loss : 2.0618601742436313
for n_estimators = 10 and max depth = 3
Log Loss : 1.5280489957196726
for n_estimators = 10 and max depth = 5
Log Loss : 1.3194905254305027
for n_estimators = 10 and max depth = 10
Log Loss : 1.6589707254126604
for n_estimators = 50 and max depth = 2
Log Loss : 1.6558253033395725
for n_estimators = 50 and max depth = 3
Log Loss : 1.3837287921875558
for n_estimators = 50 and max depth = 5
Log Loss : 1.3084523935130783

```

```

for n_estimators = 50 and max depth = 10
Log Loss : 1.6281506778575454
for n_estimators = 100 and max depth = 2
Log Loss : 1.5247183912949906
for n_estimators = 100 and max depth = 3
Log Loss : 1.3942531097748407
for n_estimators = 100 and max depth = 5
Log Loss : 1.3149027625563132
for n_estimators = 100 and max depth = 10
Log Loss : 1.6518720100240585
for n_estimators = 200 and max depth = 2
Log Loss : 1.5619331088728783
for n_estimators = 200 and max depth = 3
Log Loss : 1.3736511396978532
for n_estimators = 200 and max depth = 5
Log Loss : 1.3413553602251937
for n_estimators = 200 and max depth = 10
Log Loss : 1.6403609406032373
for n_estimators = 500 and max depth = 2
Log Loss : 1.5863226445670078
for n_estimators = 500 and max depth = 3
Log Loss : 1.4528280456979437
for n_estimators = 500 and max depth = 5
Log Loss : 1.3456748854119724
for n_estimators = 500 and max depth = 10
Log Loss : 1.5903842429011912
for n_estimators = 1000 and max depth = 2
Log Loss : 1.572415556238346
for n_estimators = 1000 and max depth = 3
Log Loss : 1.4587049480392362
for n_estimators = 1000 and max depth = 5
Log Loss : 1.353578768506442
for n_estimators = 1000 and max depth = 10
Log Loss : 1.5692347764959824
For values of best alpha = 50 The train log loss is: 0.068065047356
77832
For values of best alpha = 50 The cross validation log loss is: 1.3
084523935130783
For values of best alpha = 50 The test log loss is: 1.3437204205585
531

```

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

```

In [98]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
# class_weight=None)

# Some of methods of RandomForestClassifier()

```

```
# fit(X, y, [sample_weight])    Fit the SVM model according to the given
# predict(X)    Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

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

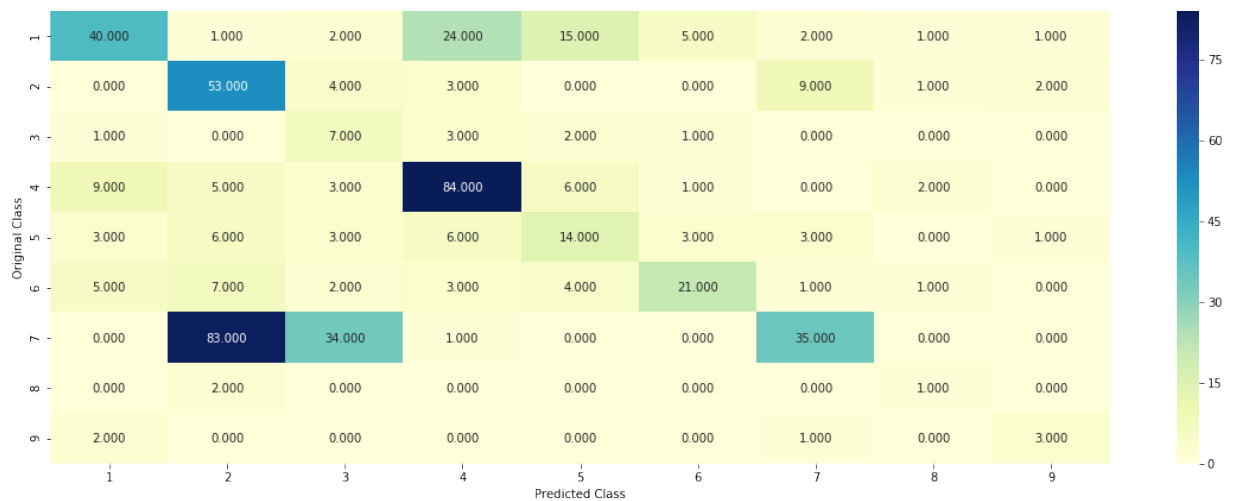
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=100,
                             predict_and_plot_confusion_matrix=train_x_responseCoding, train_y=cv_y, cv_x=cv_x)
```

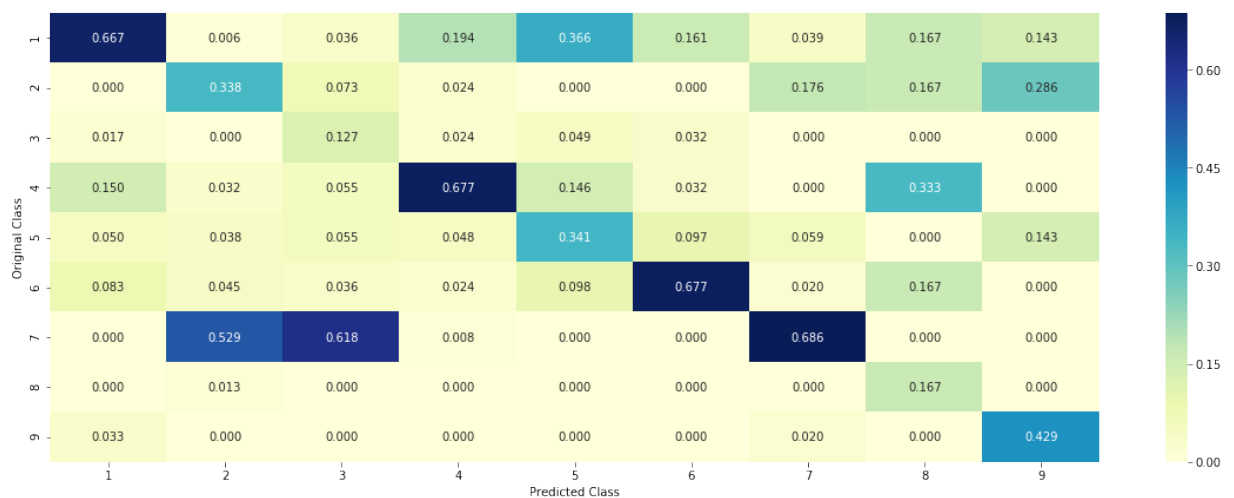
Log loss : 1.3084523935130783

Number of mis-classified points : 0.5150375939849624

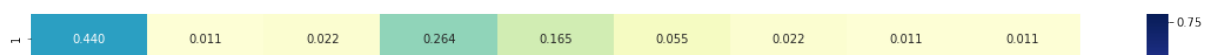
----- Confusion matrix -----

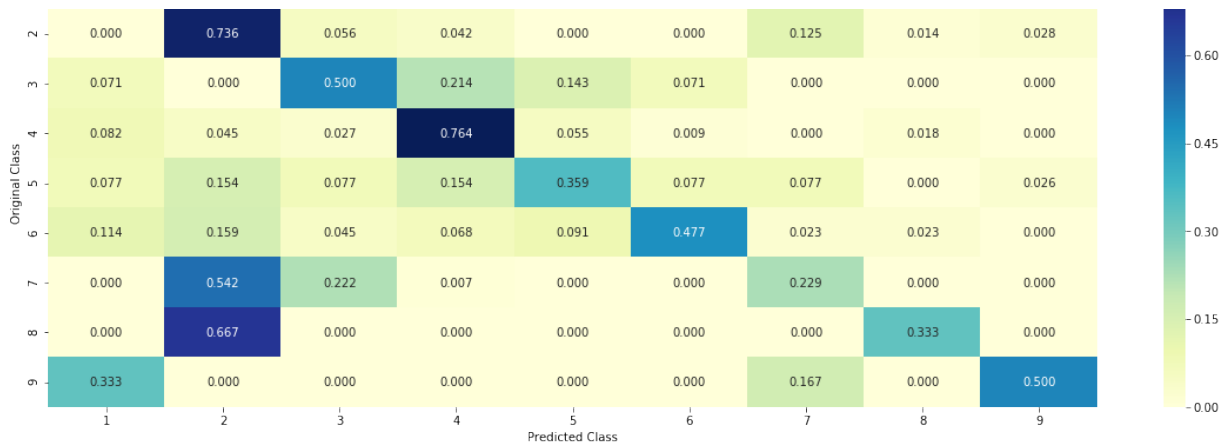


----- Precision matrix (Column Sum=1) -----



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
In [99]: lf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], crit
lf.fit(train_x_responseCoding, train_y)
ig_clf = CalibratedClassifierCV(clf, method="sigmoid")
ig_clf.fit(train_x_responseCoding, train_y)
```

```
est_point_index = 1
o_feature = 27
redicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]).
rint("Predicted Class :", predicted_cls[0])
rint("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(t
rint("Actual Class :", test_y[test_point_index])
ndices = np.argsort(-clf.feature_importances_)
rint("-"*50)
or 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: [[0.0097 0.5834 0.0826 0.0143 0.031
0.0313 0.2323 0.008 0.0073]]
Actual Class : 2
```

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


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

4.5.5.2. Incorrectly Classified point

```
In [104]: test_point_index = 42
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index]), 5))
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: [[0.0109 0.4808 0.125 0.0167 0.0154
0.0318 0.3027 0.0085 0.0082]]

Actual Class : 7

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```
In [105]: # read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-----

# read more about support vector machines with linear kernels here htt
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinkin
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decis

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the giv
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----

# read more about support vector machines with linear kernels here htt
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
# class_weight=None)

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

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

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course
# -----
```

```

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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced')
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: %.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: %.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: %.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)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %f" % (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: 1.01
Support vector machines : Log Loss: 1.46
Naive Bayes : Log Loss: 1.25

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.035
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.488
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.057
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.094
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.283

```

4.7.2 testing the model with the best hyper parameters

```
In [107]: lr = LogisticRegression(C=0.1)
scf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3],
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(
plot_confusion_matrix(test_y=test_y, predict_y=scf.predict(test_x_one
```

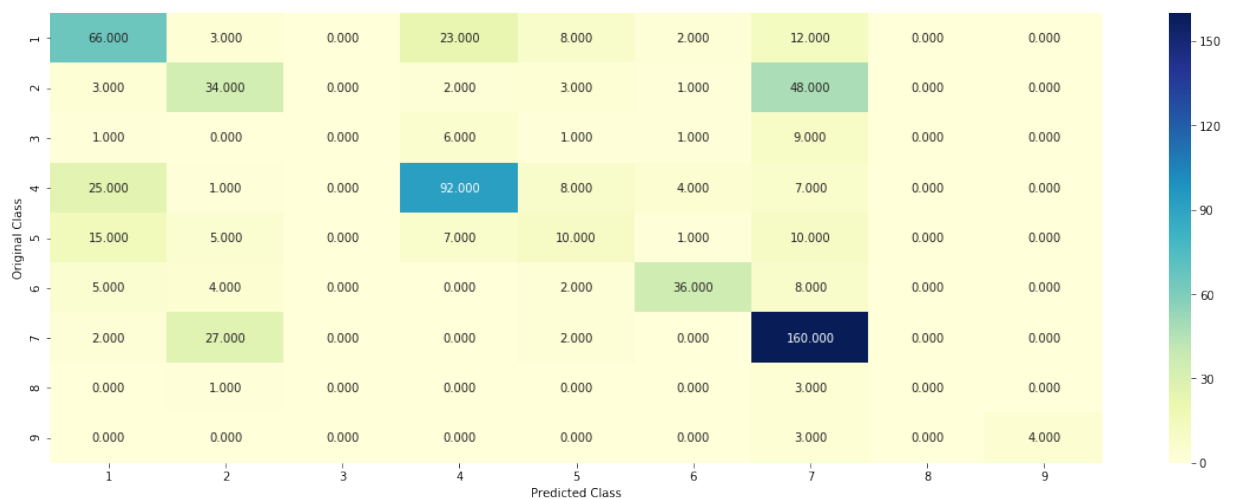
Log loss (train) on the stacking classifier : 0.6460147880855395

Log loss (CV) on the stacking classifier : 1.0569310880395937

Log loss (test) on the stacking classifier : 1.178268583701092

Number of missclassified point : 0.3954887218045113

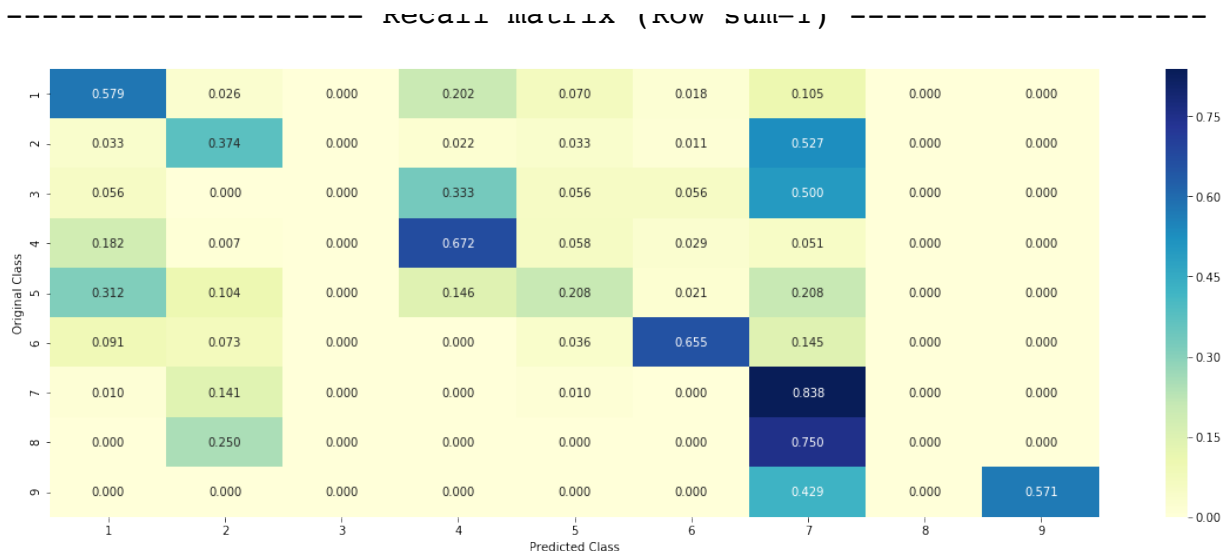
----- Confusion matrix -----



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



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



4.7.3 Maximum Voting classifier

```
In [108]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2)])
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero(vclf.predict(test_x_onehotCoding) != test_y))
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

Log loss (train) on the VotingClassifier : 0.8652420438827987

Log loss (CV) on the VotingClassifier : 1.128267956005765

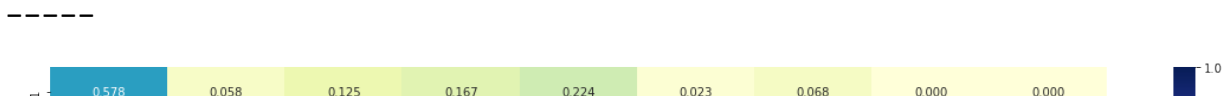
Log loss (test) on the VotingClassifier : 1.192585126401401

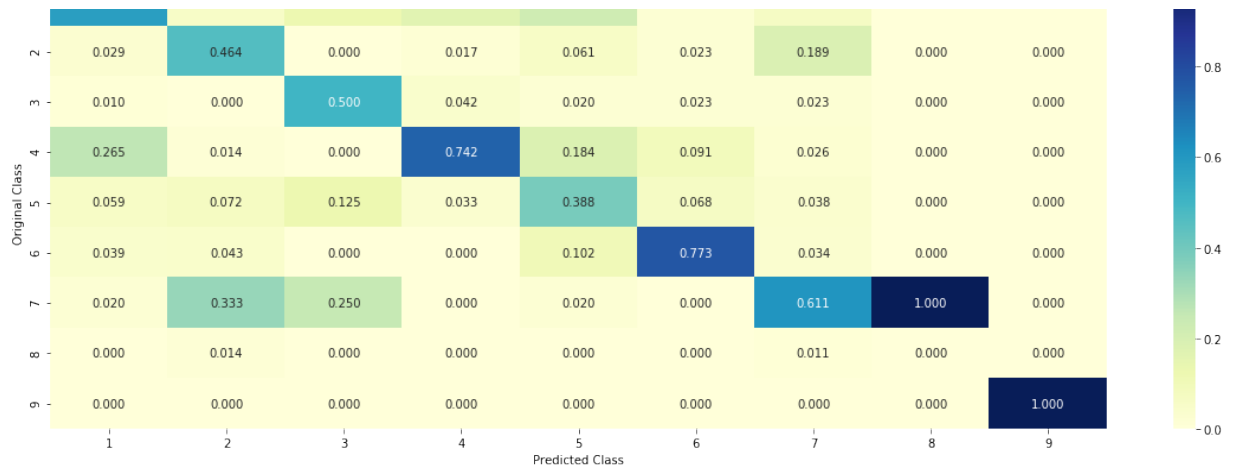
Number of missclassified point : 0.3894736842105263

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

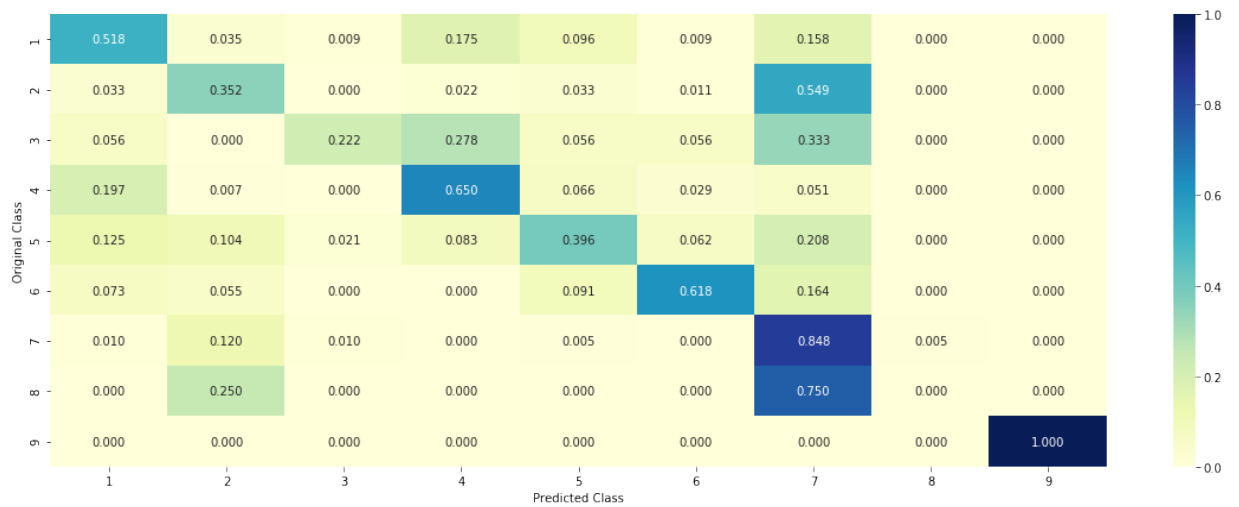


----- Precision matrix (Column Sum=1) -----





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



5. OBSERVATION

```
In [2]: from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Algorithm used", "Train Score", "Test Score", "CV score"]

x.add_row(["Naive Bayes", 0.83029, 1.20768, 1.24489, 40.22])
x.add_row(["KNN", 0.47751, 1.01647, 1.11440, 35.15])
x.add_row(["LR (Balanced data)", 0.58336, 1.02190, 1.12188, 31.39])
x.add_row(["LR (Without Balanced data)", 0.58025, 1.04681, 1.14848, 31.57])
x.add_row(["Linear SVM", 0.67855, 1.04460, 1.18038, 31.39])
x.add_row(["Random Forest (one hot encoding)", 0.67904, 1.17271, 1.16545, 38.9])
x.add_row(["Random Forest (Response Coding)", 0.06806, 1.30845, 1.34372, 51.5])
x.add_row(["Stacking model (LR, SVM, NB)", 0.64601, 1.05693, 1.17826, 39.54])
x.add_row(["Maximum Voting Classifier", 0.86524, 1.12826, 1.19258, 0.38947])
print(x)
```

	Algorithm used	Train Score	Test Score	CV score
	% misclassified			
	Naive Bayes	0.83029	1.20768	1.24489
489	40.22			
	KNN	0.47751	1.01647	1.11440
144	35.15			
	LR (Balanced data)	0.58336	1.02190	1.12188
188	31.39			
	LR (Without Balanced data)	0.58025	1.04681	1.14848
848	31.57			
	Linear SVM	0.67855	1.04460	1.18038
038	31.39			
	Random Forest (one hot encoding)	0.67904	1.17271	1.16545
545	38.9			
	Random Forest (Response Coding)	0.06806	1.30845	1.34372
372	51.5			
	Stacking model (LR, SVM, NB)	0.64601	1.05693	1.17826
826	39.54			
	Maximum Voting Classifier	0.86524	1.12826	1.19258
258	0.38947			