Personalized Medicine: Redefining Cancer Treatment

PROBLEM STATEMENT:

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

WHAT PROBLEM WE ARE TRYING TO SOLVE?

Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers).

The workflow is as follows

1. A molecular pathologist selects a list of genetic variations of interest that he/she want to analyze
2. The molecular pathologist searches for evidence in the medical literature that somehow are relevant to the genetic variations of interest
3. Finally this molecular pathologist spends a huge amount of time analyzing the evidence related to each of the variations to classify them

Our goal here is to replace step 3 by a machine learning model. The molecular pathologist will still have to decide which variations are of interest, and also collect the relevant evidence for them. But the last step, which is also the most time consuming, will be fully automated.

Dataset Description:

training_variants - 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)

training_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

test_variants - 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)

test_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)

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.

How our data looks like?

- training_variants (ID , Gene, Variations, Class)
- training_text (ID, Text)

ID,Gene,Variation,Class

0,FAM58A,Truncating Mutations,1
1,CBL,W802*,2
2,CBL,Q249E,2
.

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…….. still goes on.

WHAT TYPE OF PROBLEM IS THIS?

There are nine different classes of a genetic mutation. So, this can be posed as a Multi class classification problem.

1.EXPLORATORY DATA ANALYSIS:

LIBRARIES USED:

- **NLTK – Natural Language Toolkit**
- **Matplotlib**
- **Numpy**
- **Scikit-Learn**
- **Seaborn**
- **Pandas**

Training_variants:

Number of data points : 3321

Number of features : 4

Features : ['ID' 'Gene' 'Variation' 'Class']

Training_text

Number of features : 2

Features : ['ID' 'TEXT']

Preprocessing of text

nlp_preprocessing(total_text, index, column):

Purpose: This function prepares text data for natural language processing (NLP) tasks.

Parameters:

1. total_text: Text data to clean and preprocess.
2. index: Position of the text data in the dataset.
3. column: Location in the dataset to store the cleaned text.

Process:

1. Type Check: Ensures the provided total_text is not an integer.
2. Text Cleaning: Replaces special characters with spaces. Condenses multiple spaces into single spaces. Converts all text to lowercase.
3. Stop Word Removal: Eliminates common stop words from the text.

NOTE:

Stop words are a set of commonly used words in a language that are filtered out before or after processing of natural language data because they are insignificant.

Returns: Updates the specified column in the dataset (data_text) with the cleaned text.

AFTER PREPROCESSING OUR TEXT DATA WILL LOOK LIKE THIS:

abstract background non small cell lung cancer nsclc heterogeneous group disorders number genetic proteomic alterations c cbl e3 ubiquitin ligase adaptor molecule important normal homeostasis cancer determined genetic variations c cbl relationship receptor tyrosine kinases egfr met functionality nsclc methods findings using archival formalin fixed paraffin embedded ffpe extracted genomic dna show c cbl mutations occur somatic fashion lung cancers c cbl mutations mutually exclusive met egfr mutations however independent p53 kras mutations normal tumor pairwise analysis significant loss heterozygosity loh c cbl locus 22 n 8 37 none samples revealed mutation remaining copy c cbl c cbl loh also positively correlated egfr met mutations observed samples using select c cbl somatic mutations s80n h94y q249e w802 obtained caucasian taiwanese african american samples respectively transfected nsclc cell lines increased cell viability cell motility conclusions taking overall mutation rate c cbl co….

HANDLING NULL OR MISSING VALUES;

Assigns the concatenated string of 'Gene' and 'Variation' to the 'TEXT' column for rows where 'TEXT' was originally null.

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

	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

```
# replacing the missing text values by concatinating the its gene and variation text.
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

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

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

SPLITTING THE DATA FOR TRAINING, TESTING:

```
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
```

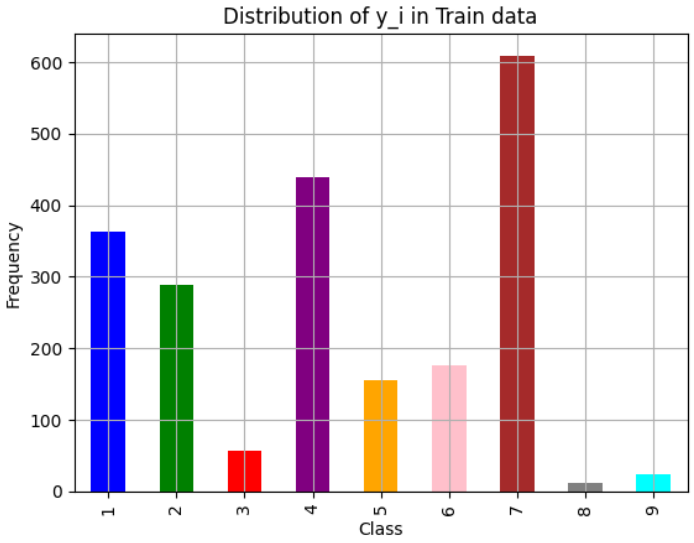
```
X_train, test_df, Y_train, y_test = train_test_split(result, y_class, test_size = 0.2, stratify = y_class)
train_df, cv_df, y_train, y_cv = train_test_split(X_train, Y_train, test_size = 0.2, stratify = Y_train)
```

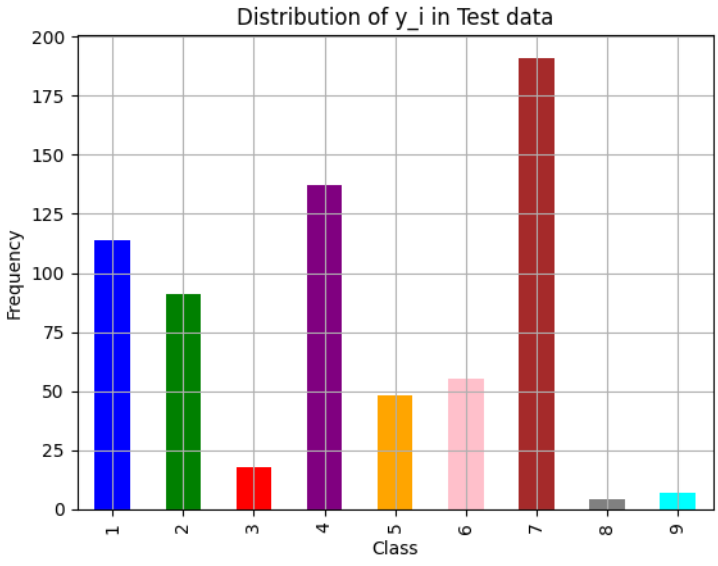
The above code first modifies the 'Gene' and 'Variation' columns in the dataset by replacing any spaces with underscores.

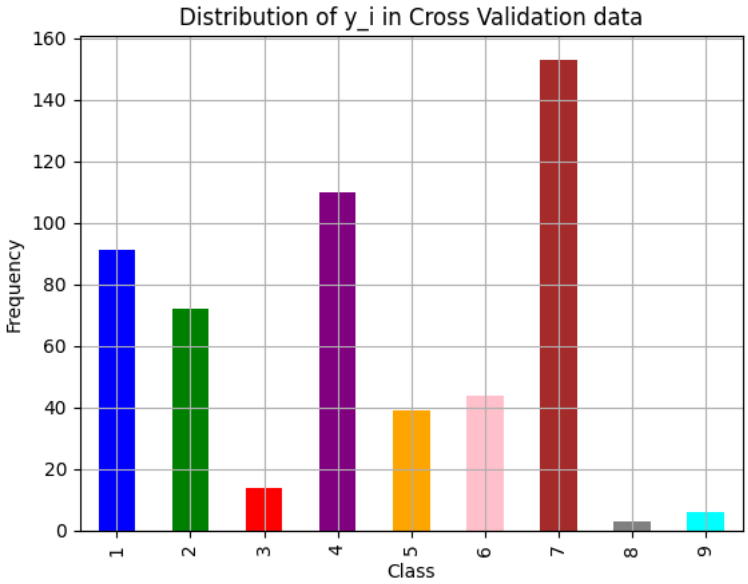
After this modification, it splits the dataset into three subsets: a training set, a testing set, and a cross-validation set.

This split is designed to ensure that the distribution of the output variable ('y_true' or 'y_train') remains similar across these subsets, aiding in the fair evaluation and validation of machine learning models.

Plotting the Distribution of Class labels:







INFERENCE:

Class Imbalance:

Across all three datasets, there's a significant class imbalance. Some classes have notably fewer instances compared to others. For instance, in all sets, Class 8 and Class 9 have a very small number of data points compared to the majority classes (Class 1, Class 2, Class 4, and Class 7).

Consistency in Class Distribution:

The percentages of each class within the datasets are relatively consistent across the three subsets. This consistency indicates that the initial split has maintained the proportions of classes in each subset(because of using strarify)

Relative Proportions:

Classes 1, 2, 4, and 7 generally constitute a larger percentage of the datasets compared to other classes. This skew might influence the model's learning behavior, especially for the minority classes (Class 8 and Class 9), which have significantly fewer samples.

WHAT IS LOG LOSS?

REFERENCE : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

Log loss, aka logistic loss or cross-entropy loss.

This is the loss function used in (multinomial) logistic regression and extensions of it such as neural networks, defined as the negative log-likelihood of a logistic model that returns `y_pred` probabilities for its training data `y_true`. The log loss is only defined for two or more labels. For a single sample with true label $y \in \{0, 1\}$ and a probability estimate $p = \Pr(y = 1)$, the log loss is:

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p))$$

WHAT IS A RANDOM MODEL?

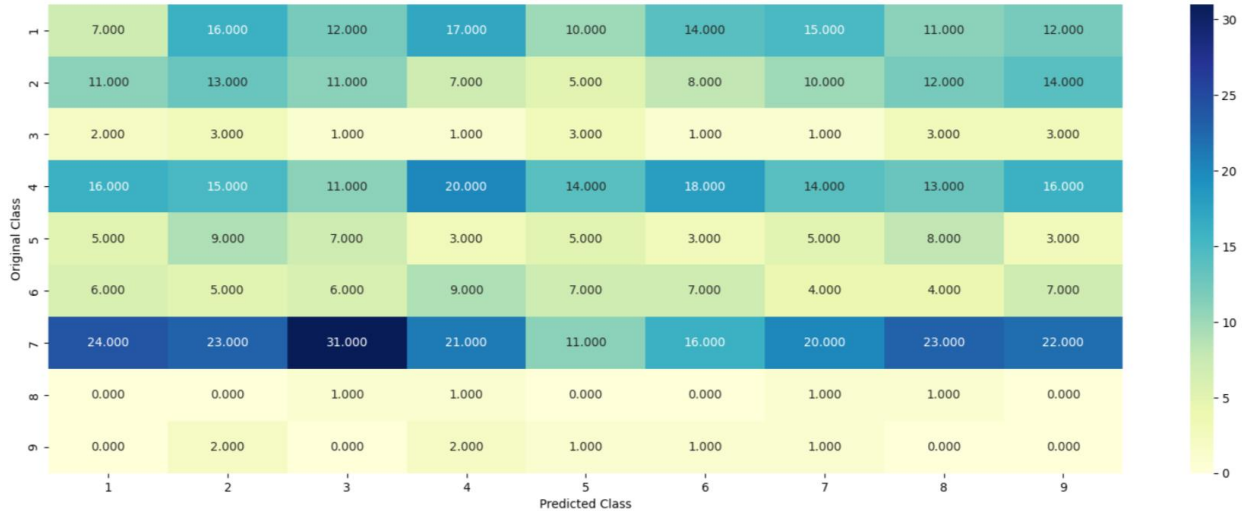
A random model, also known as a baseline model or a random classifier, is a simple model used as a benchmark or a point of reference for evaluating the performance of more sophisticated models.

For multiclass problems, it assigns labels with equal probability across all classes.

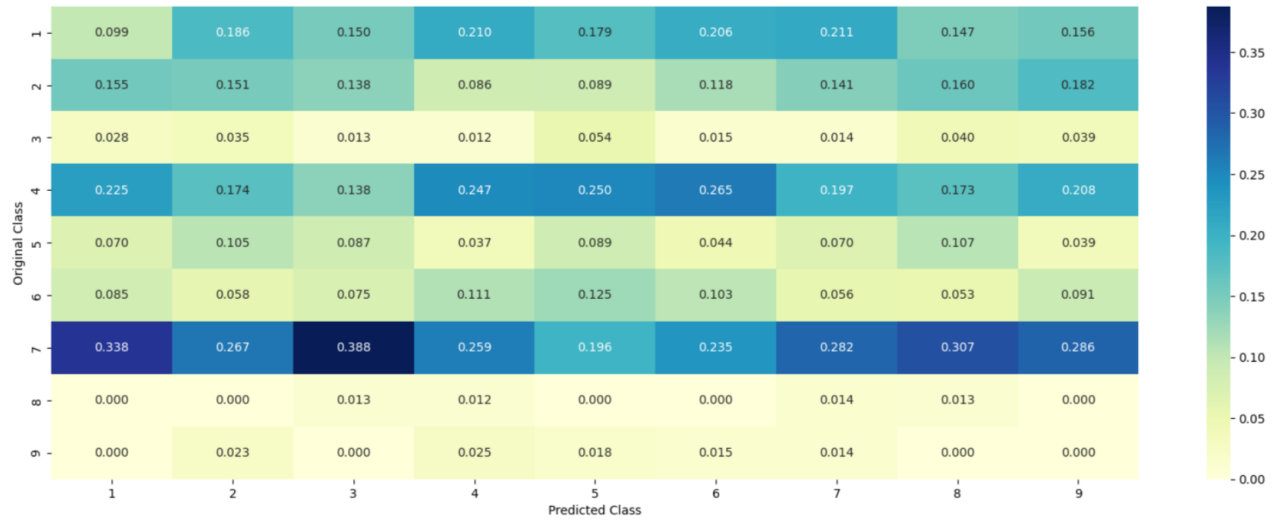
WHAT DOES OUR RANDOM MODEL DO AND HOW IT WORKS?

Our random classifier in this code randomly assigns probabilities to different classes without considering any information from the dataset. It generates random probabilities for each class and normalizes them so that their sum equals 1, resembling a probability distribution.

Log loss on Cross Validation Data using Random Model : 2.1972245773362196
Log loss on Test Data using Random Model 2.447349583769534
---------- Confusion matrix ----------



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



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

UNIVARIATE ANALYSIS ON ALL THE FEATURES

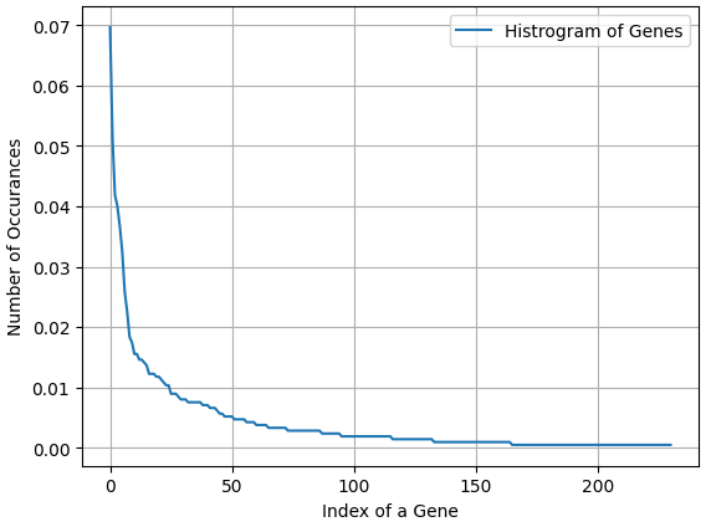
UNIVARIATE ANALYSIS ON GENE FEATURE:

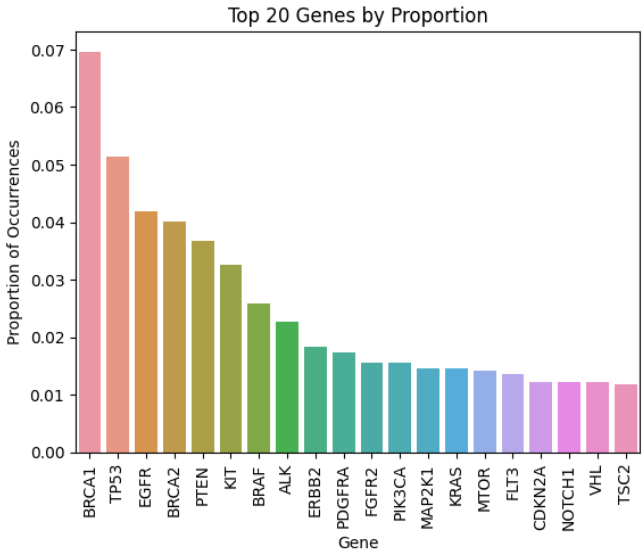
1.What type of feature is GENE?

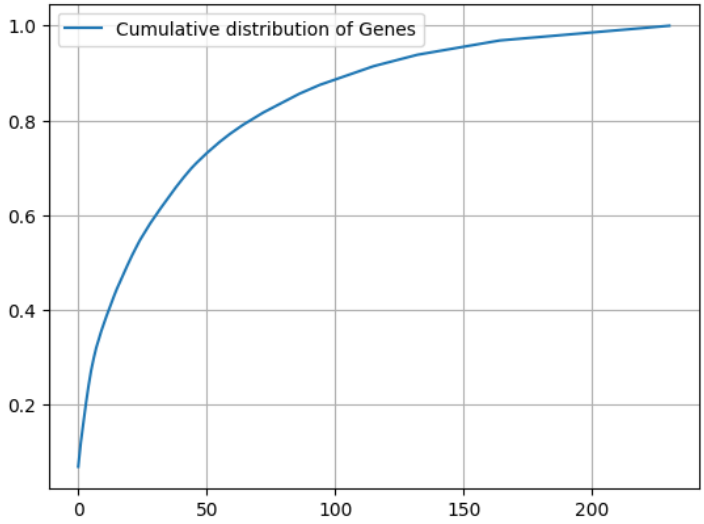
Gene is a categorical variable

2.How many categories are there and how they are distributed?

```
Number of Unique Genes or different categories of gene : 231
BRCA1     148
TP53      109
EGFR       89
BRCA2      85
PTEN       78
KIT        69
BRAF       55
ALK        48
ERBB2      39
PDGFRA     37
Name: Gene, dtype: int64
```





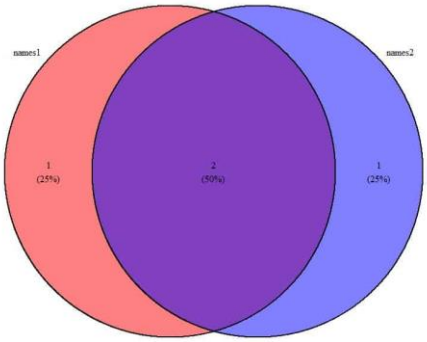


OBSERVATIONS:

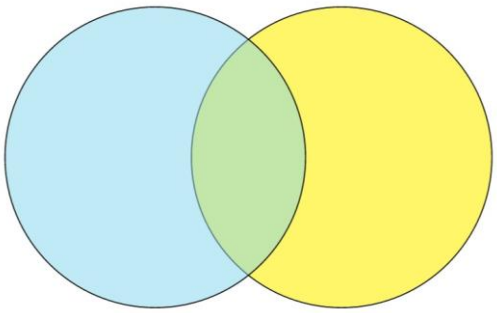
1. Here, we can clearly notice the distribution of genes are very skewed.
2. Almost, 200 Genes occur less than 1 percentage out of all the genes.
3. Top 50 genes contribute to 75% of the total occurrence of gene in the data.
4. It means, there are some genes(around 50) that occur very frequently and lots of genes that occur very less frequent(around 180).

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

How to determine a feature is stable or not?



STABLE



NOT STABLE

```
How many data points in Test and CV datasets are covered by the  238  genes in train dataset?
Ans
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 513 out of  532 : 96.42857142857143
```

CONCLUSION:

This indicates that there is a strong or high overlapping between set of genes in train and test, set of genes in train and cross validation dataset.

Hence, gene must be one the important feature.

4. How to featurize this Gene feature?

There are two ways we can featurize this gene variable,

1. One hot Encoding
2. Response coding

ONE HOT ENCODING:

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Refernce Link : https://en.wikipedia.org/wiki/One-hot#Natural_language_processing

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 229)
```

train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 229)

RESPONSE CODING:

Here, we will create a vector of probabilities. Where, the size or length of the vector will be equal to total number of classes = 9.

Reference link : Response Coding for Categorical Data

Represent each gene category as a set of probabilities corresponding to different classes. For instance, if you have classes A, B, and C, and a gene feature G with categories G1, G2, and G3, your representation might look like:

G1: [P(A|G1), P(B|G1), P(C|G1)]

G2: [P(A|G2), P(B|G2), P(C|G2)]

G3: [P(A|G3), P(B|G3), P(C|G3)]

Output vector=[$p1$,$p2$,$p3$,$p4$,$p5$,$p6$,$p7$,$p8$,$p9$]

- 1$p1$ represents the probability or score that the input belongs to Class 1.
- $p2$ represents the probability or score that the input belongs to Class 2.
- ...and so on until *p9* represents the probability or score for Class 9.

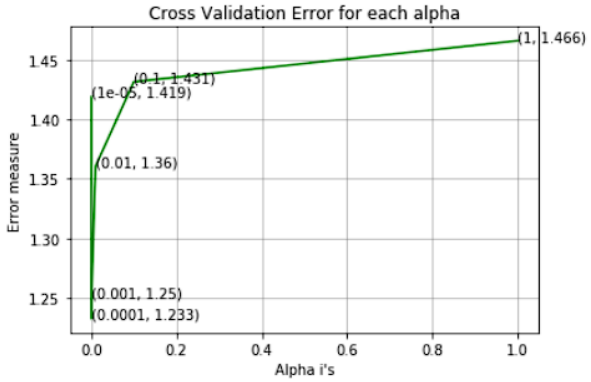
```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape of gene feature: (2124, 9)
```

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

5.How good is this gene feature in predicting our class labels?

There are many ways to estimate how good a feature is, in predicting y'i. One of the simple method 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.

```
For values of alpha =  1e-05 The log loss is: 1.418841767162939
For values of alpha =  0.0001 The log loss is: 1.2325868001617826
For values of alpha =  0.001 The log loss is: 1.2503129272158073
For values of alpha =  0.01 The log loss is: 1.360379976757511
For values of alpha =  0.1 The log loss is: 1.4314392521126913
For values of alpha =  1 The log loss is: 1.4659143358159061
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0425604300119806
For values of best alpha =  0.0001 The cross validation log loss is: 1.2325868001617826
For values of best alpha =  0.0001 The test log loss is: 1.200905436534172
```

Univariate Analysis on Variation Feature

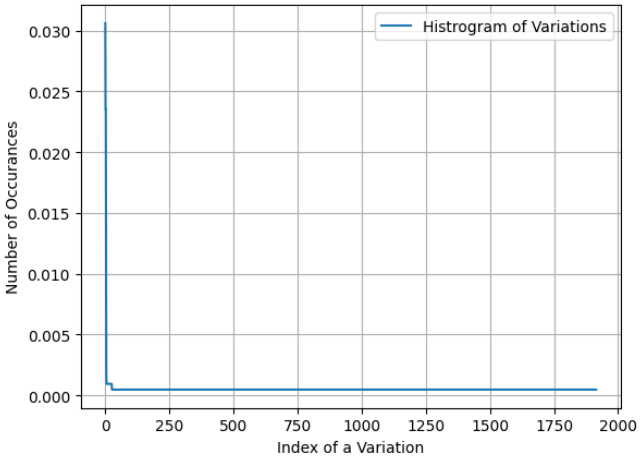
1.What type of feature is Variation?

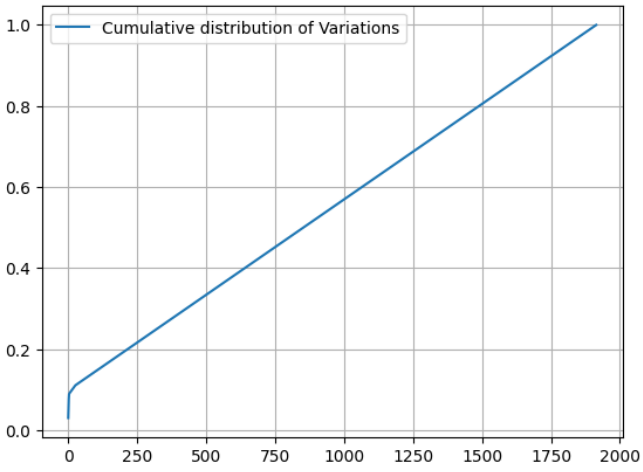
Variation is a categorical variable.

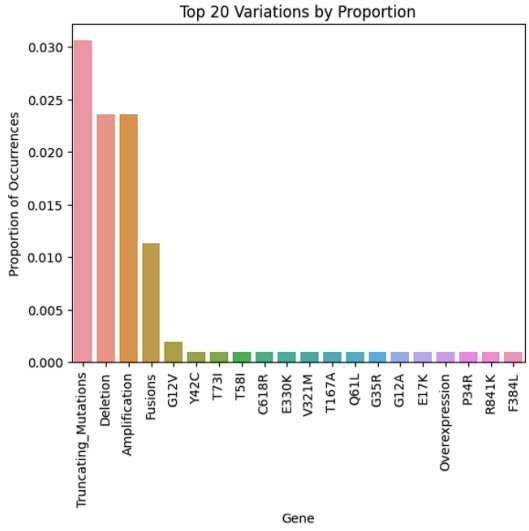
2. How many categories are there and how are they distributed?

There are 1915 different categories of variations in the train data

```
Number of Unique Variations : 1915
Truncating_Mutations    65
Deletion                50
Amplification           50
Fusions                 24
G12V                     4
Y42C                     2
T73I                     2
T58I                     2
C618R                    2
E330K                    2
Name: Variation, dtype: int64
```





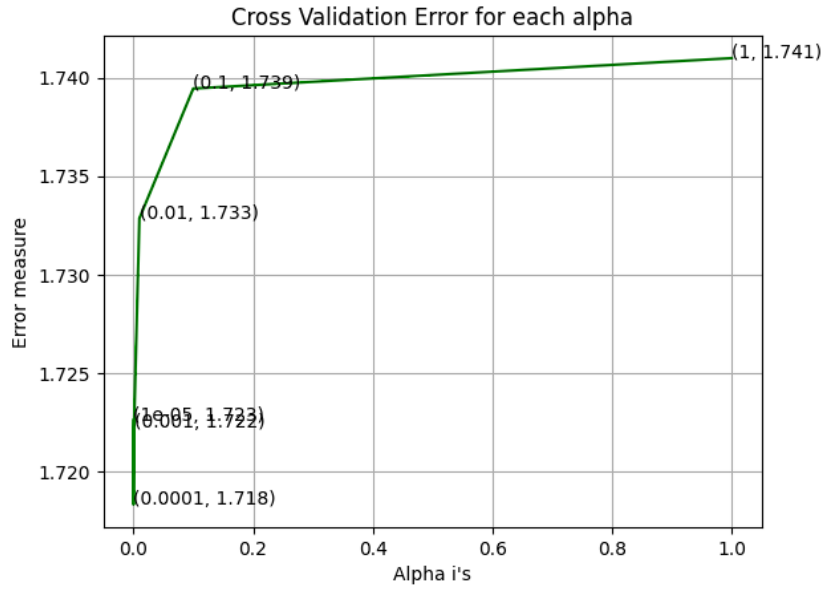


OBSERVATIONS:

The Gene Variation is super skewed.

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

```
For values of alpha =  1e-05 The log loss is: 1.7226140620564314
For values of alpha =  0.0001 The log loss is: 1.718328166553091
For values of alpha =  0.001 The log loss is: 1.7222333410497603
For values of alpha =  0.01 The log loss is: 1.7328615746467173
For values of alpha =  0.1 The log loss is: 1.739450043706098
For values of alpha =  1 The log loss is: 1.7409941717099298
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7117957433361833
For values of best alpha =  0.0001 The cross validation log loss is: 1.718328166553091
For values of best alpha =  0.0001 The test log loss is: 1.7289449506562453
```

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

```
How many data points are covered by total  1915  genes in test and cross validation data sets?
Ans
1. In test data 65 out of 665 : 9.774436090225564
2. In cross validation data 48 out of  532 : 9.022556390977442
```

5.How are we going to featurize this variation feature?

There are two ways we can featurize this variable.

1. **One hot Encoding**
2. **Response coding**

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

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1960)
```

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

train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1960).

UNIVARIATE ANALYSIS ON TEXT FEATURE:

1.How many unique words are present in train data?

Total number of unique words in train data : 53396

2.How to featurize this text feature?

There are various methods to featurize (represent) text data for machine learning models. Here are some commonly used techniques:

a. **Bag of Words (BoW):**
   CountVectorizer: Converts text into a matrix representing the count of each word occurrence in the document. Each column represents a unique word, and values are the word counts.
b. **Term Frequency-Inverse Document Frequency (TF-IDF):**
   TfidfVectorizer: Similar to CountVectorizer but also considers the importance of words by penalizing common words occurring in multiple documents.
c. **Word Embeddings:**
   Word2Vec, GloVe, FastText: Create dense vector representations of words in a continuous vector space. Words with similar meanings are closer together in the vector space.
d. **N-grams:**
   CountVectorizer/TF-IDF with N-grams: Captures sequences of N words instead of single words, providing context information.

We will be using the simple and elegant Bag of Words Approach and Response Coding.

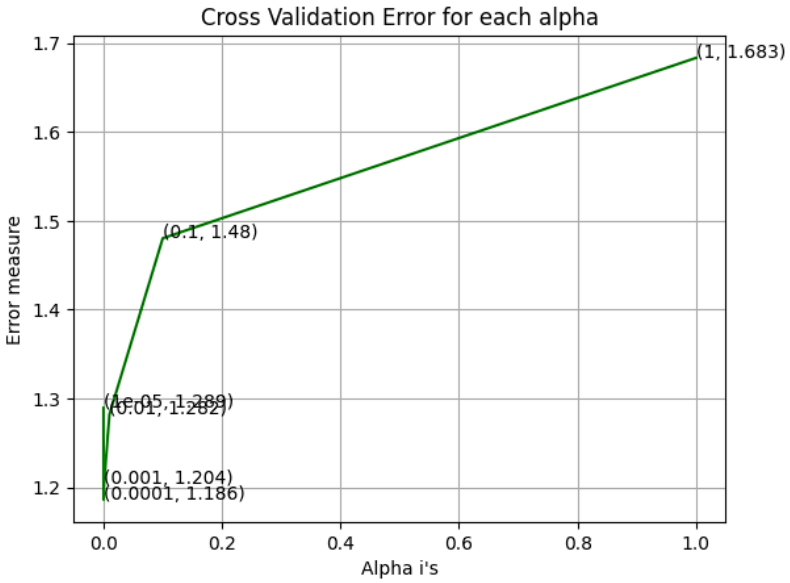
After merging/stacking all our features based on our encoding method, this is how our dimensions of our dataset looks like.

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55578)
(number of data points * number of features) in test data =  (665, 55578)
(number of data points * number of features) in cross validation data = (532, 55578)
```

```
 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)
```

3.Is the text feature useful in predicitng y_i?

```
For values of alpha =  1e-05 The log loss is: 1.2893436276926484
For values of alpha =  0.0001 The log loss is: 1.186354725550688
For values of alpha =  0.001 The log loss is: 1.204472508468598
For values of alpha =  0.01 The log loss is: 1.2820583477278986
For values of alpha =  0.1 The log loss is: 1.480357877547436
For values of alpha =  1 The log loss is: 1.6832974439446222
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6541547774124429
For values of best alpha =  0.0001 The cross validation log loss is: 1.186354725550688
For values of best alpha =  0.0001 The test log loss is: 1.1446366674824648
```

We can say that from all the above features that we discussed text feature has the lowest log loss in Training and cross validation. Hence this is the most important feature that we should not miss out.

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

```
95.55 % of word of test data appeared in train data
98.209 % of word of Cross Validation appeared in train data
```

Yes, the text feature is highly stable.

CONCLUSION OF UNIVARIATE ANALYSIS ON ALL THE FEATURES:

NOTE:

All these inferences are performed basis on the logistic regression model.

FEATURE	TRAIN	CROSS VALIDATION	TEST	STABILITY
GENE	**0.99137**	**1.22572**	**1.17379**	**STABLE**
VARIATION	**0.68471**	**1.74398**	**1.71288**	**LESS STABLE**
TEXT	**0.65425**	**1.18635**	**1.14463**	**STABLE**

BASE LINE MODEL : NAÏVE BAYES

HYPER-PARAMETER TUNING:

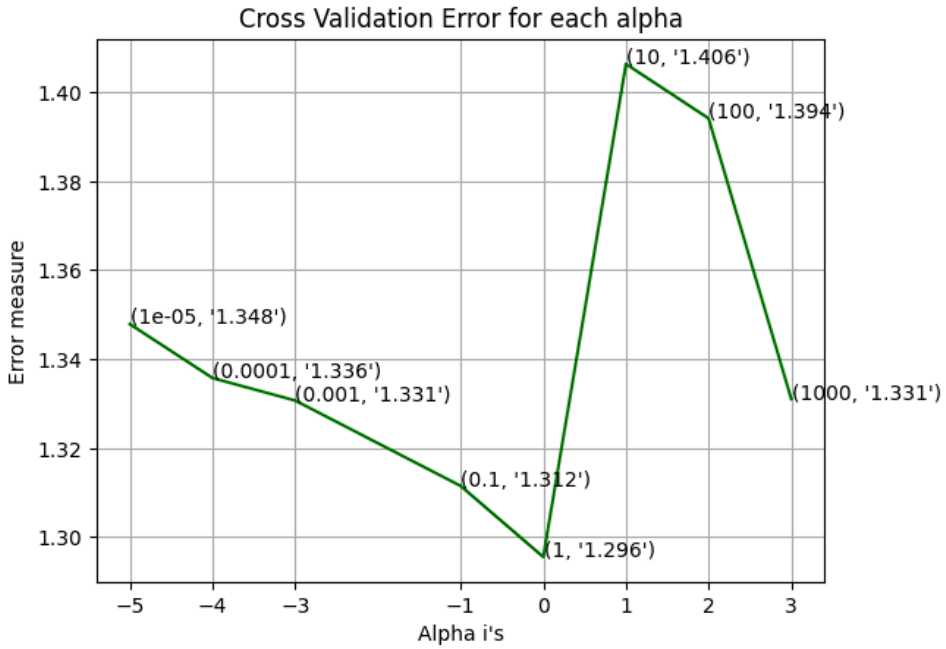
Alpha (α) in Multinomial Naive Bayes:

Smoothing Parameter (α):

- In Multinomial Naive Bayes, the alpha hyperparameter is used for **Laplace smoothing** (additive smoothing). It's a non-negative value that helps handle the ***issue of zero probabilities*** for words not present in the training data.
- A smaller alpha value results in less smoothing, which might lead to overfitting if the training data is sparse or if some features have zero probabilities.
- A larger alpha value results in more smoothing, reducing the risk of overfitting but potentially affecting the model's ability to capture fine-grained details in the data**.**

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

```
for alpha = 1e-05
Log Loss : 1.3478711165034103
for alpha = 0.0001
Log Loss : 1.3357599339704822
for alpha = 0.001
Log Loss : 1.3307083782299027
for alpha = 0.1
Log Loss : 1.3115325993713889
for alpha = 1
Log Loss : 1.2955668601530128
for alpha = 10
Log Loss : 1.4063339148457792
for alpha = 100
Log Loss : 1.394109014204986
for alpha = 1000
Log Loss : 1.3310441823571844
```



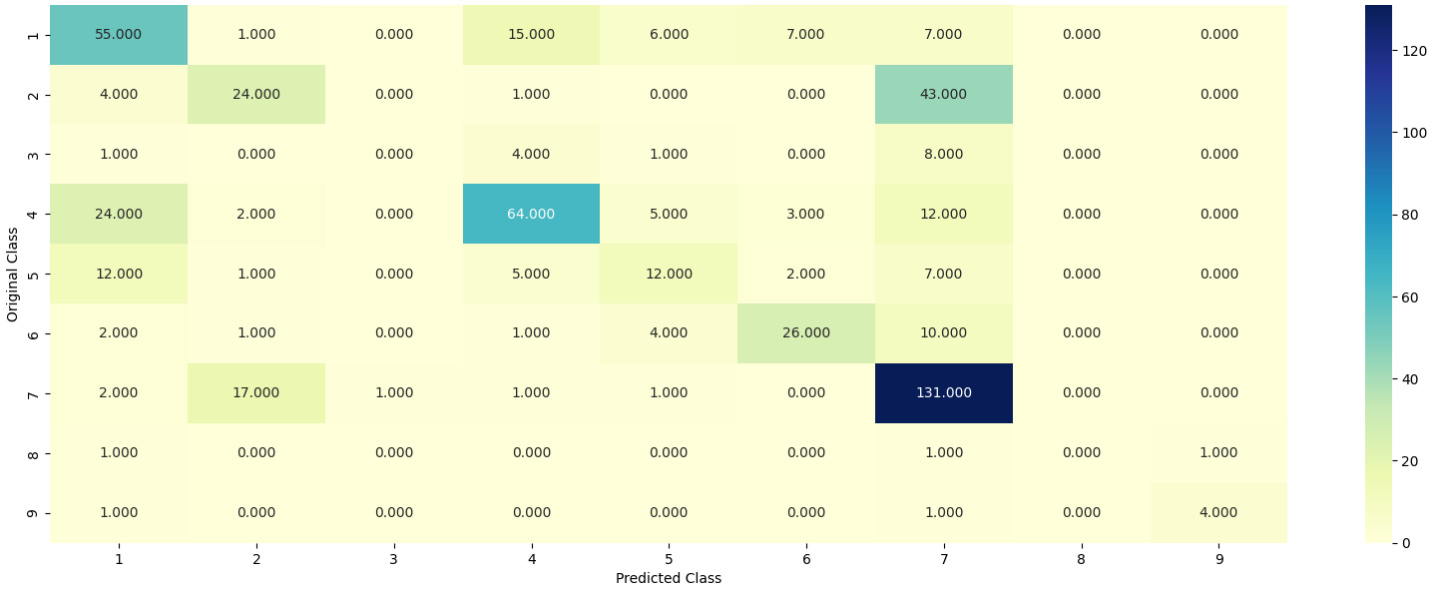

```
For values of best alpha =  1 The train log loss is: 0.8863914452163685
For values of best alpha =  1 The cross validation log loss is: 1.2955668601530128
For values of best alpha =  1 The test log loss is: 1.2437576523794762
```

TESTING THE MODEL WITH BEST HYPER-PARAMETER:

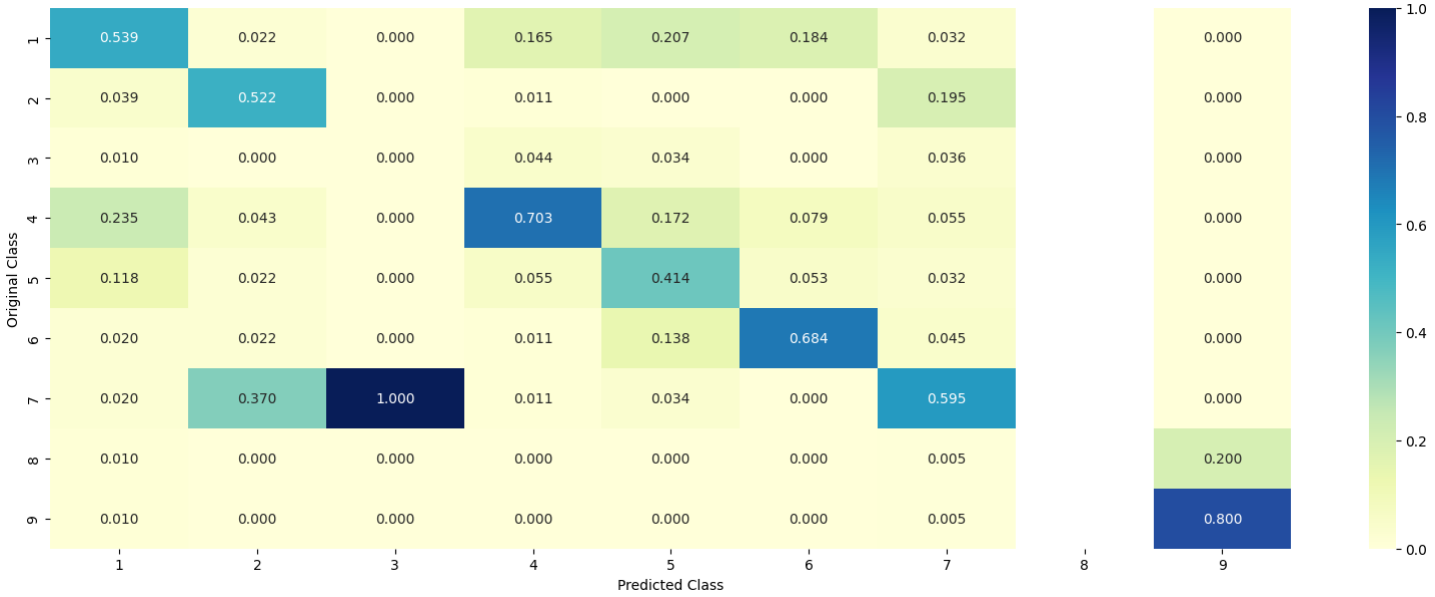
Log Loss : 1.2955668601530128

Number of missclassified point : 0.40601503759398494

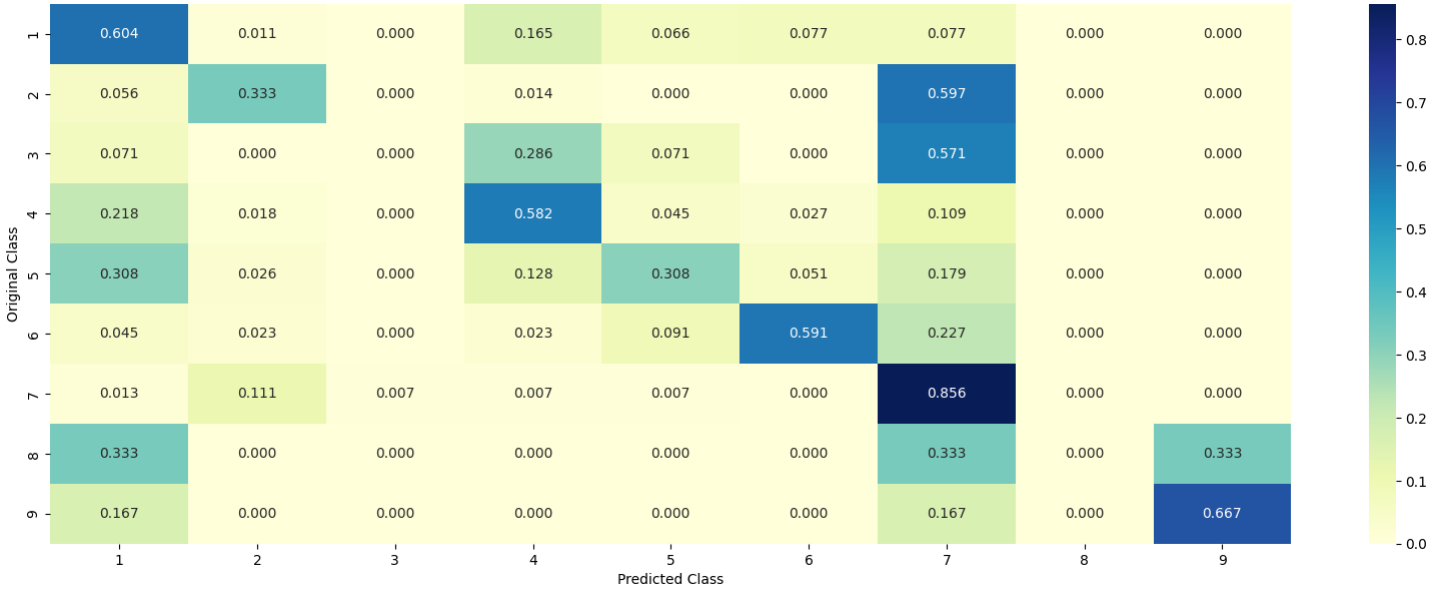
CONFUSION MATRIX:



PRECISION MATRIX: COLUMN SUM = 1



RECALL MATRIX : ROW SUM = 1




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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.071  0.0823 0.0218 0.0876 0.044  0.0369 0.6461 0.0057 0.0046]]
Actual Class : 5
```

```
test_point_index = 3
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.071  0.0823 0.0218 0.0876 0.044  0.0369 0.6461 0.0057 0.0046]]
Actual Class : 7
```

K Nearest Neighbour Classification

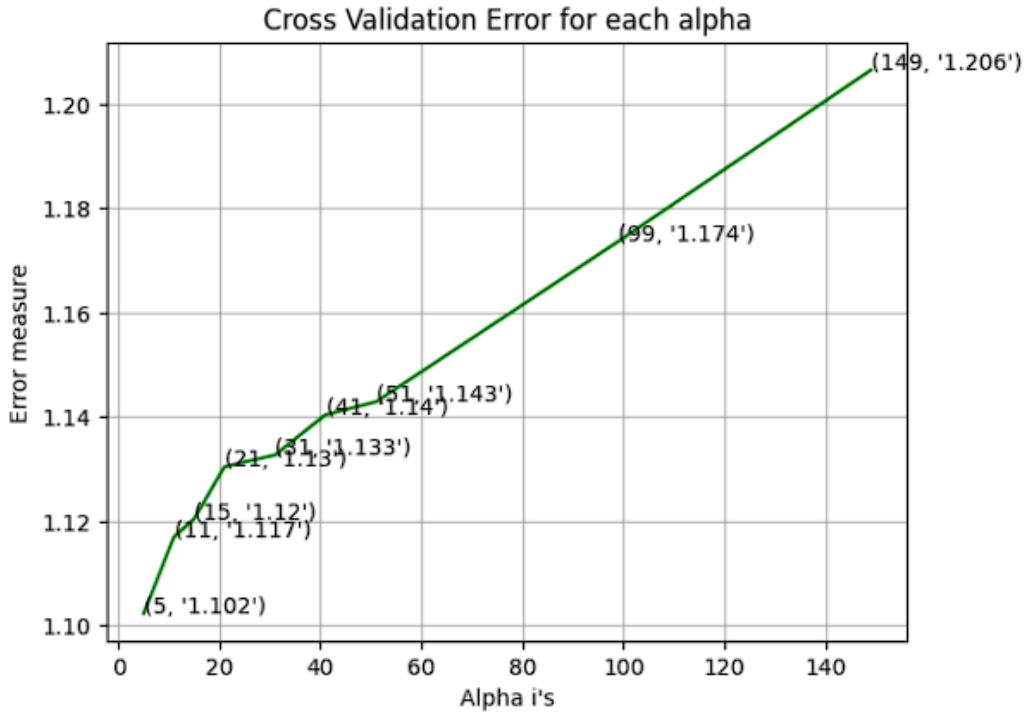
Hyper-Parameter Tuning:

Hyperparameters in KNN : Number of Neighbors (k):

- The most critical hyperparameter in KNN.
- It determines the number of neighbors used to make predictions for a new data point.

alpha = [5, 11, 15, 21, 31, 41, 51, 99, 149]

```
for alpha = 5
Log Loss : 1.1023234355091387
for alpha = 11
Log Loss : 1.116998161540611
for alpha = 15
Log Loss : 1.120469729949882
for alpha = 21
Log Loss : 1.1304548640838943
for alpha = 31
Log Loss : 1.132686892443943
for alpha = 41
Log Loss : 1.1404188682208716
for alpha = 51
Log Loss : 1.1428812067090108
for alpha = 99
Log Loss : 1.1736430240734905
for alpha = 149
Log Loss : 1.2064726042329563
```



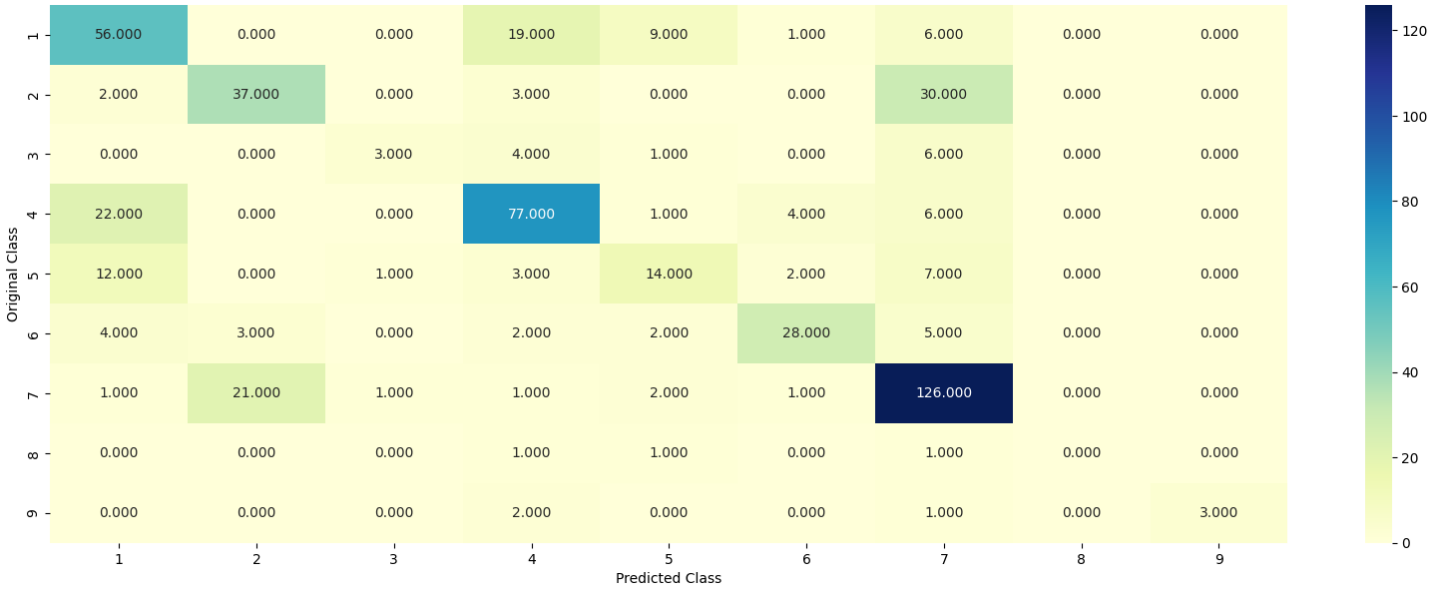

```
For values of best alpha =  5 The train log loss is: 0.43795670272419124
For values of best alpha =  5 The cross validation log loss is: 1.1023234355091387
For values of best alpha =  5 The test log loss is: 1.0500180977435405
```

Testing the model with best hyper parameter:

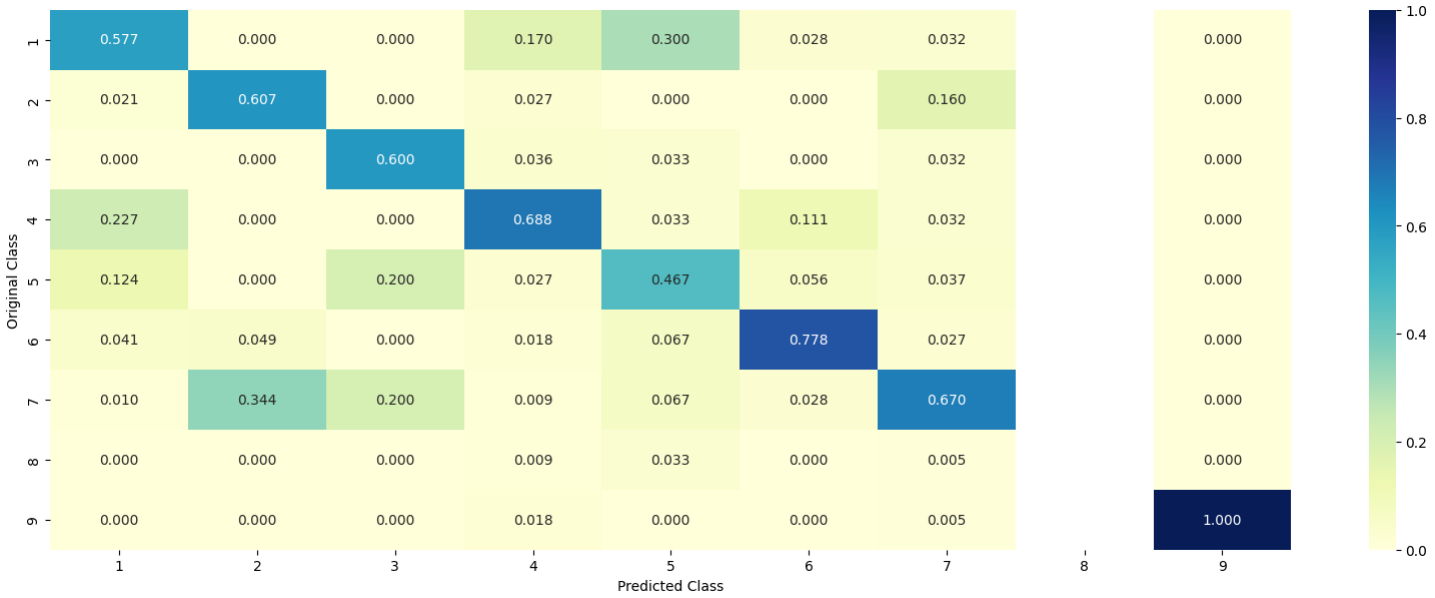
Log loss : 1.1023234355091387

Number of mis-classified points : 0.3533834586466165

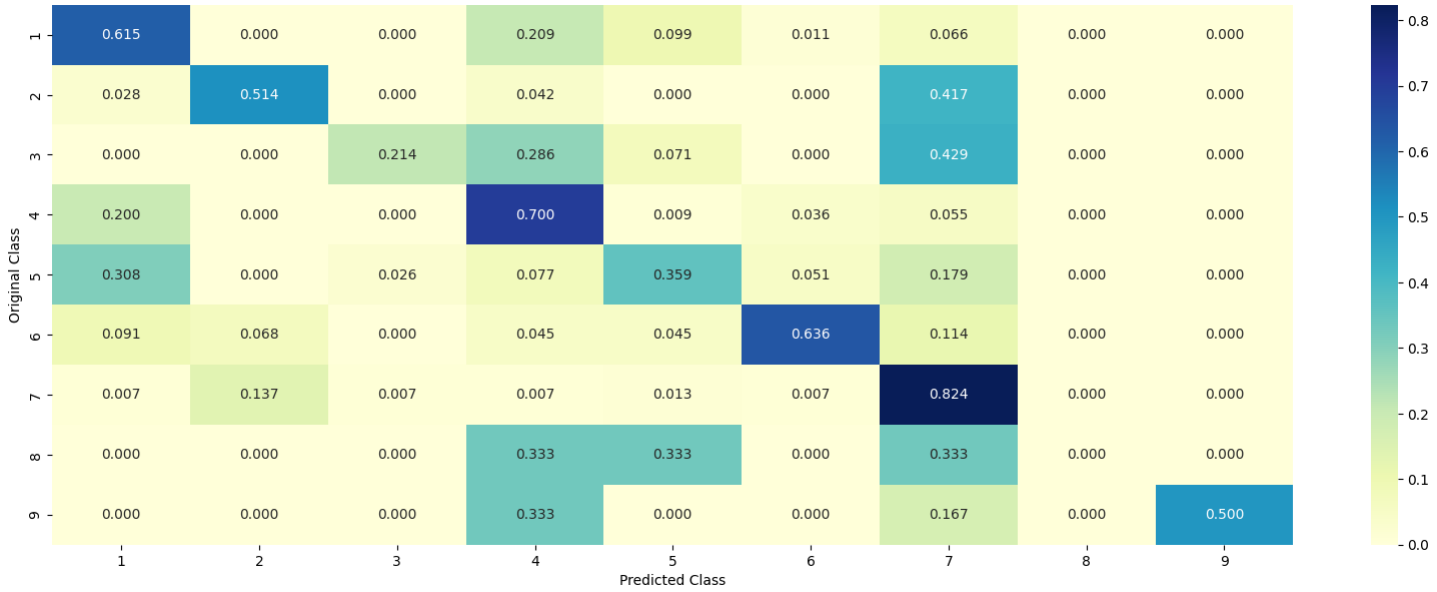
CONFUSION MATRIX :



PRECISION MATRIX :



RECALL MATRIX:




```
test_point_index = 40

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

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

```
test_point_index = 143

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

```
Predicted Class : 7
Actual Class : 1
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [1 7 1 3 7]
Fequency of nearest points : Counter({1: 2, 7: 2, 3: 1})
```