Redefining Cancer Treatment

Gaurav Pandey

Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research
Institute Belur Math, Howrah
Pin - 711 202, West Bengal

A thesis submitted to
Ramakrishna Mission Vivekananda Educational and Research Institute
in partial fulfillment of the requirements for the degree of
MSc in Big Data
Analytics 2019-2021

Dedicated to all the researchers working hard to find the modern ways to treat cancer

Acknowledgements

I take immense pleasure in thanking the great many people who helped in completion of my project associated with it. My deepest thanks to my faculty guide i.e, Dr./Br. Dripta Mj. for guiding and correcting me with attention and care. I greatly appreciate the efforts he took to go through my data and make necessary corrections as and when needed. I also want to express my thanks to Dr./Br. Mrinmay Mj., Professor and Swathy Prabhu Mj., H.O.D for extending their support. Words are inadequate in offering my thanks to the various helpful people of RKMVERI for their encouragement and cooperation in carrying out the project work. I would also thank my Institution and my faculty members without whom this project would have been distant reality. Finally, yet importantly I would like to express my thanks to my beloved parents for their blessings my friends/classmates for their help and wishes for the successful completion of this project.

Ramakrishna Mission Vivekananda Educational and Research Institute, Belur Math, West Bengal

Gaurav

June 30, 2020

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled *'Redefining cancer treatement'* submitted by *Mr. Gaurav Pandey*, who has been registered for the award of MSc in Big Data Analytics degree of Ramakrishna Mission Vivekananda Educational and Research Institute, Belur Math, Howrah, West Bengal is absolutely based upon his/her own work under the supervision of *Prof./Dr./Br. Dripta Mj.* of Department of Computer Science, Ramakrishna Mission Vivekananda Educational and Research Institute and that neither his/her thesis nor any part of the thesis has been submit- ted for any degree/diploma or any other academic award anywhere before.

Dr./Br. Dripta Mj.
Emeritus Professor/Professor/Associate Professor/AssistantProfessor
Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, Howrah, 711 202, West Bengal

Abstract

A lot has been said during the past several years about how precision medicine and, more concretely, how genetic testing is going to disrupt the way diseases like cancer are treated.

But this is only partially happening due to the huge amount of manual work still required.

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

Currently this interpretation of genetic mutations is being done manually. This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

We are going to build machine learning algorithm using this knowledge base as a baseline, automatically classifies genetic variations.

Contents

Chapter 1

Introduction

Understanding the dataset is very important to for further evaluation and algorithm building we are going to do with this dataset.

This problem of “medical treatment on cancer” is generally faced a lot in medical industry that is the institution named MEMORIAL SLOAN KETTERING CANCER CENTER launched the dataset opensource and asked the aspiring data scientists to provide their solution using different-different techniques and models for the concrete solution of this problem.

The problem is there is a lot of manual work is happening by the researchers and they wanted to replace it by using the machine learning techniques.

In simple words, every human body have some sort of gene’s and genetic sequence in their body and if your genetic sequence lead to any variation due to various reasons like smoking, hereditary etc.

Those variations will lead to the cancer in human body but definitely it is not always the case that all the tumours lead to cancer but it is most times leads to cancer.

Now, we have 9 classes provided which means that if some sort of gene gone through with some sort of variation then it will lead to one of those classes (those classes are maybe the type of cancer like lung, blood, bone cancer etc.)

The thing here is that we do have genes, variations and classes. So, the problem faced by the pathologists (these are the people who have vast knowledge on the genes and the type of variations happen) are that there are so many genes and variations occur in the body and to understand the type of variations there are research papers available to them which they need to refer whenever any variation occur amongst the gene and going through with all the research papers takes so much time and efforts and because it is an issue of medical so the minimal mistakes are acceptable.

Hence, they need to be very careful as well with the handling because it is very sensitive case due to which it is very time consuming. To make the work easy for pathologists we are going to take help of the machine learning for classification of these gene’s and variations with respect to the relevant classes.

Chapter 2

Data

The dataset we have been provided to solve the problem stated above would be in two parts:

- Variant file- This contains the information about the genes, variations and the classes. These are the features present in the datafile.
- Text data file- This contains the information regarding the research papers available to identify the type of cancer.

Text data file:

```
]: data_text.head()
```

]:

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

Variant data file:

```
: data_variants.head()
```

:

	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

In class column, unique no. of classes we have:

```
: data_variants.Class.unique()
: array([1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64)
```

Now, the problem we are going to solve is that we have gene, we have variation and we have research papers (in text file). By using these we need to predict the class.

In text dataset, I have noticed one unusual thing with the dataset while describing it the following output shows that we do have some missing values in our dataset.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 2 columns):
ID      3321 non-null int64
TEXT    3316 non-null object
dtypes: int64(1), object(1)
memory usage: 52.0+ KB
```

As you can see there are 3321 entries from which in text column only 3316 are defined rest are the missing values.

Chapter 3

Methodology

Since, we are dealing with the problem where errors can be very costly we need to be very careful while dealing with this kind of problem. This can be proven costly to researcher I can explain this via short example, let's supposed a patient gone to the doctor for his evaluation/screening process and due to some errors researcher predicted the wrong class of cancer then it can be proven dangerous for patient because patient then go through with that sort of cancer treatment procedure. So, here we have life and death situation here that is to contain these errors we would like to have results for each class in terms of probability.

In terms of probability means, for example, we have

$P(y=3|x)=0.99$, where x-> variation|gene|text

But if I have the probability something like that

$P(y=3|x)=0.6$

$P(y=2|x)=0.4$, where x-> variation|gene|text

Then sure it has predicted 3 but the probability is still a factor and we may need to look further evidence to improve the chances.

This means that prediction y in class 3 have 99% chances of being there.

Hence, we might more concerned about the accuracy of the results latency is not the issue here the time machine learning algorithm will take to predict would be seem reasonable.

This is a multi-class classification problem we are going to solve.

One more important thing is we have been asked by the institution to find the results with respect to multiclass log-loss and the minimal log-loss will shows the accuracy of the model.

Chapter 4

Experimental Evaluation

4.1 Dealing with the text data

Here we have text file in our project report on cancer.

So, the project report contains text data as we discussed earlier, which is nothing just contains the brief information about the type of cancer and genes associated with the cancer.

Now, form one perspective this is very important for the researcher's point of view but here this report we cannot directly feed to the ML-based algorithm. So, here first we need to cleanse the data like we do in the NLP.

In cleansing now what needs to be removed/ replaced and what to keep so that we can smooth the data as per our requirement without losing any information.

- Remove the double spaces and tabs present in the dataset.
- There are so many unnecessary numbers present in the dataset which we do not need right now for our processing the data.
- There are so many words in the data set where same words use in different format like gene and Gene.
- There are so many stop words present in the text data file like an, a, the etc…

Now, after cleansing we are going to merge our dataset with the variants file, there is common component needed to merge and that common component here is "ID".

After merging we create a new data frame, so the question arises that is there any missing value present in the dataset and if there is how to deal with it??

There are many methods to deal with the missing values but just because we do not have very large dataset so I opt to impute the missing values with the genes and variations.

Elsewise, if I had a very large data, I could have dropped out the missing values.

Now, this data is ready for the next step….. i.e., Training, Testing the dataset etc…...

Output:

	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...

The above output shows the data frame after merging the two dataset and creating one data frame.

4.2 Splitting the data

Now, here we come to the most important part of this project.

General procedure we use in machine learning is to simply divide the data into training set and testing set and then create model on training set and test it on testing set but sometimes this is not a good thing to do because when we play with the things called hyperparameter tuning then these things may cause a problem in a way that we may be become more dependent on the testing dataset. The result comes out with the test dataset and we start looking for accuracy on training dataset this creates the dependency on the test dataset.

In order to handle that we come up with the validation dataset, here we divide the dataset into 3 parts rather than two parts.

Here, those three parts are:

1. Training set
2. Cross-validation set
3. Testing set

In training set, we are going to build our model.

In cross-validation, we treat it as a test set where we test our hyperparameter tuning and all the other stuff. We going to know that what the best parameter we get out of it.

In test set, we do not touch it yet, when we completely build our model then we use this test dataset for predictions and testing stuff of our dataset.

Here, we can adopt any method like 70-20-10 distribution etc.

I have adopted first split 80-20 in training and test respectively. Then again splitting the training into 80-20 ratio in training and cross-validation respectively.

Now, after splitting it is very important that we do notice that distribution of the data happened properly because if the distribution does not happen properly many kinds of problem can occur like:

- Here we have 9 classes, suppose class 1 components are stayed in the training dataset and there is a chance that they haven't gotten into the cross-validation and testing set.

If this happens then there is no way to bringing in and verify that whether my model is to bringing the class 1 is good or not.

- It can also happen that suppose class 7 components we have here is not much present in the training set, so if that happens then we do not have much information in training set to train our model and get accurate predictions because not much patterns available to study or to learn from the training set.

So, classes distribution from the raw data is very difficult to compare. Hence, as a data scientist I try to visualize this dataset distribution.

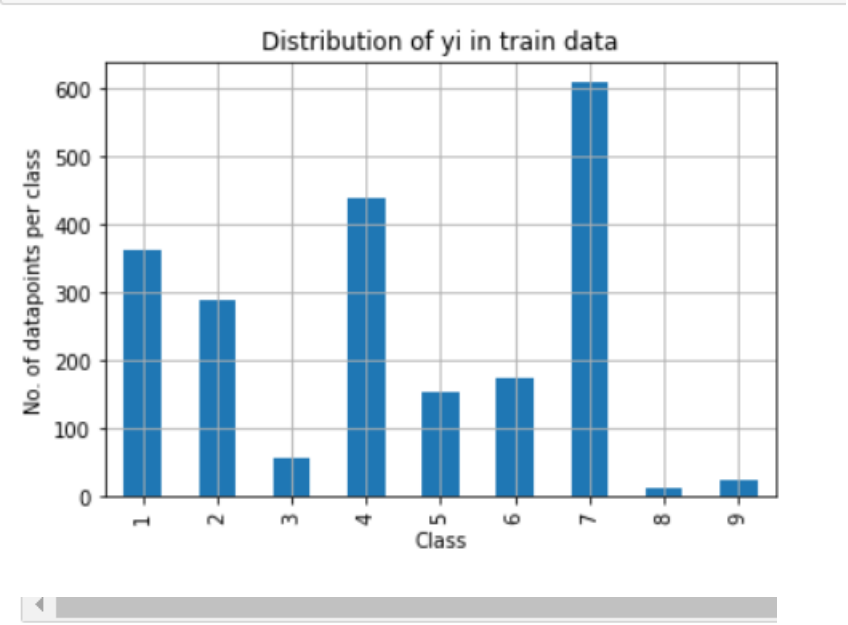
Output:

```
No. of data points in train data: 2124
No. of data points in test data: 665
No. of data points in cross-validation data: 532
```

This output shows how much data distributed amongst the three datasets.

Distribution output:

Training data

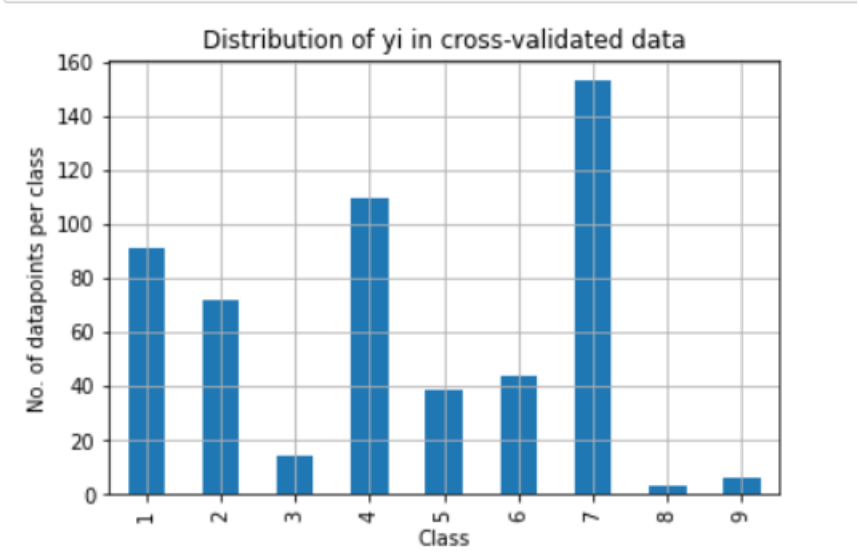



```
No. of datapoints in class: 7 : 609 ( 28.672 %)
No. of datapoints in class: 4 : 439 ( 20.669 %)
No. of datapoints in class: 1 : 363 ( 17.09 %)
No. of datapoints in class: 2 : 289 ( 13.606 %)
No. of datapoints in class: 6 : 176 ( 8.286 %)
No. of datapoints in class: 5 : 155 ( 7.298 %)
No. of datapoints in class: 3 : 57 ( 2.684 %)
No. of datapoints in class: 9 : 24 ( 1.13 %)
No. of datapoints in class: 8 : 12 ( 0.565 %)
```

This output shows the distribution classwise and distribution seems to be perfect.

Distribution output:

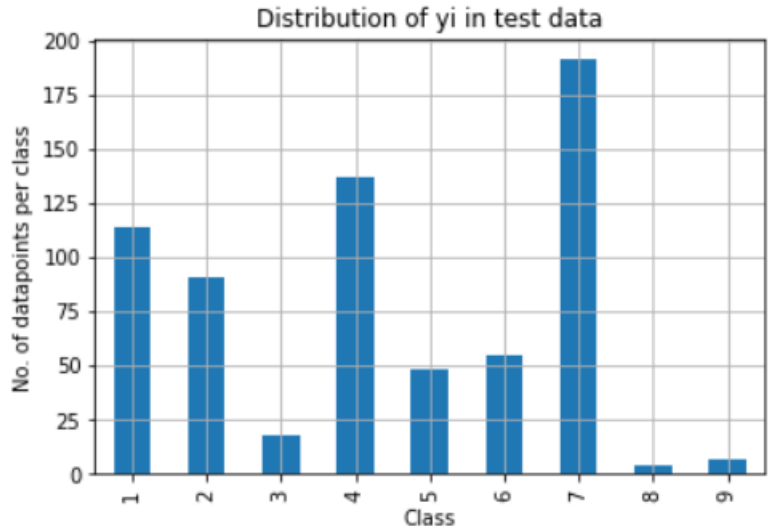
Cross-validated data




```
No. of datapoints in class: 7 : 153 ( 28.759 %)
No. of datapoints in class: 4 : 110 ( 20.677 %)
No. of datapoints in class: 1 : 91 ( 17.105 %)
No. of datapoints in class: 2 : 72 ( 13.534 %)
No. of datapoints in class: 6 : 44 ( 8.271 %)
No. of datapoints in class: 5 : 39 ( 7.331 %)
No. of datapoints in class: 3 : 14 ( 2.632 %)
No. of datapoints in class: 9 : 6 ( 1.128 %)
No. of datapoints in class: 8 : 3 ( 0.564 %)
```

Distribution output:

Test data



```
No. of datapoints in class: 7 : 191 ( 28.722 %)
No. of datapoints in class: 4 : 137 ( 20.602 %)
No. of datapoints in class: 1 : 114 ( 17.143 %)
No. of datapoints in class: 2 : 91 ( 13.684 %)
No. of datapoints in class: 6 : 55 ( 8.271 %)
No. of datapoints in class: 5 : 48 ( 7.218 %)
No. of datapoints in class: 3 : 18 ( 2.707 %)
No. of datapoints in class: 9 : 7 ( 1.053 %)
No. of datapoints in class: 8 : 4 ( 0.602 %)
```

4.3 Log-loss

Log loss is nothing just to check accuracy of the model. Just like we do after every model we check accuracy; it is also one of that method.

I am using this log loss method because in my Kaggle evaluation it is asked that project is evaluated on the basis of Multiclass log loss but in real case scenario we can use any method to check the model accuracy.

Now, the question arises what is log loss?

First, I will talk about the binary log loss function:

Mathematically we can write it as followed:

$$\text{log-loss}= -\frac{1}{n}\sum_{i=1}^{n}[\left(\log\left(p_i\right)\times y_i\right)+\left(\log\left(1-p_i\right)\times\left(1-y_i\right)\right)$$

Let's suppose we have x-> input and y-> output and p-> probability of that model's output.

For p= 0.9 and y=1

We have, log-loss= 0.0457

Similarly, for p=0.4 and y=0

We have, log-loss= 0.22

This shows that the lower the log-loss the more accurate our model and vice versa.

A perfect log-loss would have 0 log-loss.

And now, the main thing multi-class log loss which is quite similar to binary log loss and I am going to use in this project.

Mathematically we can write it as followed:

$$\text{log}-\text{loss}=-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}y_{ij}\times\log\left(p_{ij}\right)$$

Where, 1= if x_i belongs to class j

0= o/w

So, taking an example of this project as we know that we have 9 classes here in the project.

To check its model accuracy, it is similarly work like our previous binary classification. The difference is that there we just had 2 classes and here we have 9 different classes to classify. Else the results will be provided in the similar manner.

In our case for each row we have some probability from our model and from that probability we decide that where which class lies more importantly log-loss plays that our model's prediction is how much accurate for the classification here in the project.

NOTE- In log-loss the value can go from 0 to infinity.

4.4 Worst model/Random model

As I have explained previously that we are using the log-loss function in our report and log-loss function can be any value between 0 and infinity.

So, it is very difficult to say that which log-loss is better or which is worse. For ex- in our model we get log loss value of 1.5. How could we tell that this 1.5 represents the better model or the worst?

Hence, for comparison I am going to create a random model which also would be the worst model which is completely created by the random numbers. So that we can compare our model with this worst model.

Now, I am going to explain how am I going to generate the random model?

To generate this random model, I'd have to find out probability for each row and please note that in the end sum of that probability could not exceed to 1. It should be equal to 1.

Now, simply I have created the random model by generating the random probabilities and ensuring that sum would be equal to 1.

What I have observed from my random model?

I simply run for the cross-validated data and test data and got the following log-loss value:

Cross-validated data: log-loss= **2.465**

Test data: log-loss= **2.436**

Now, these values represent the worst model. And if some model gives more value than these values above then it is the worst one and do not use that model.

There is one more question arises here is that how am I going to evaluate this random model, of course it is a worst model so the results it is going to provided are not going to be good. But this evaluation we can do by using the confusion matrix and the precision matrix.

There will be 9×9 matrix where there are 9 classes and on x-axis there would be a predicted class and on y-axis there would be an original class.

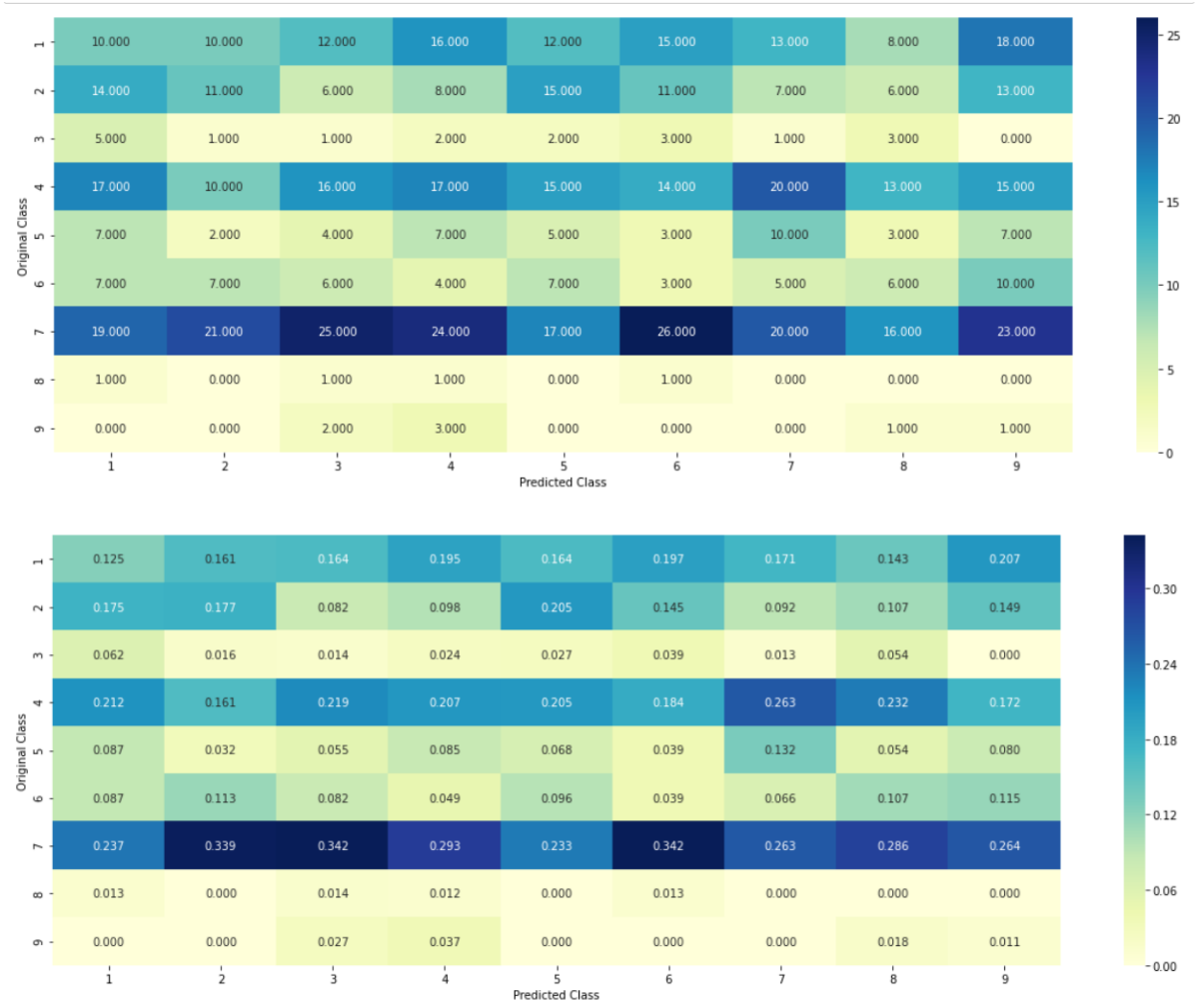
Numbers inside the confusion matrix tells us that this much of data values in the class are lies. Obviously, here we will consider the diagonal elements only because it will give us the actual values lies in the class rest is considered as the error values.

Similarly, from confusion matrix we can create the precision matrix which will tell us that that this much percent of the values lies in the particular class.

Precision matrix values are simply calculated as the value upon the sum of the values in a row.

Output:

Random model confusion and precision matrix:



As you can see there are misclassifications and confusions happening in this model that is why it is also the worst model we have created.

4.5 Dealing with categorical features in data frame

Now, here the problem arises that how to deal with the categorical variable present in our dataset. Because we cannot feed the categorical variables directly to our machine learning algorithm.

For example, in our dataset we have 4 features where 3 are independent and 1 is dependent. Class is dependent and gene, variation and text are independent. Here gene column is categorical so we need to assign some value to this column to feed this data to ML algorithm.

How do we transform the categorical variables?

There are two methods to do that transformation would be as followed:

- One-hot encoding method
- Response encoding

What is one-hot encoding and how do we implement that method in our dataset?

To answer this question first we need to understand what is one hot encoding method. It is a method in which we create extra columns with the name of unique values is our categorical variables. And we create sparse matrix from that.

Like here in our dataset we have 229 unique genes which means we need to create a sparse matrix like structure adding the 229 new columns.

There lies problem with the one-hot encoding method i.e., it increases the dimensionality of the dataset. Few algorithms have big trouble with this type of high dimensionality of the data, specially trees kind of algorithms like Random forest and decision tree (CART) etc.

What is response encoding and how do we implement that method in our dataset?

It is very interesting stuff where we assign some values just like we do with the NaN missing values. For that assigning of some logical values to categorical feature some domain knowledge about the topic plays plus point.

Now, I am going to explain how this works in contrast of this gene feature in our dataset.

As we know we have 9 classes in our dataset and unique values of genes associated with the classes as we have shown in the code.

For each class we create a column and assign probabilities to the classes. Like probability of gene being in a class 3 given that the gene is BRCA1(name of the gene).

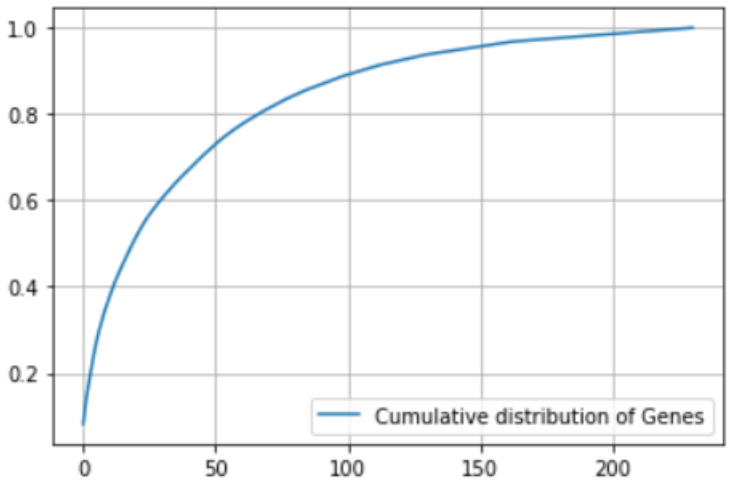
Similarly, we deal like this with all the gene values and assign some kind of posterior probabilities to them.

Supporting the claim with outputs we got:

Output:

For Gene Feature

```
No. of Unique Genes : 231
BRCA1     175
TP53      111
EGFR       83
BRCA2      74
PTEN       74
BRAF       64
KIT        54
ERBB2      48
ALK        44
CDKN2A     39
Name: Gene, dtype: int64
```



This output shows the no. of unique values in gene column.

Dimensions of training dataset in gene column with one-hot-encoding

```
train_gene_feature_onehotCoding.shape
```

```
(2124, 231)
```

Similarly, dimensions of training dataset in gene column with response encoding

```
train_gene_feature_responseCoding.shape
```

```
(2124, 9)
```

Similarly, we have also done for the variation column as well as text column in the data frame.

4.6 Machine learning models

NAÏVE BAYE'S MODEL

Here in this model we are going to use naïve bayes classifier for multinomial models.

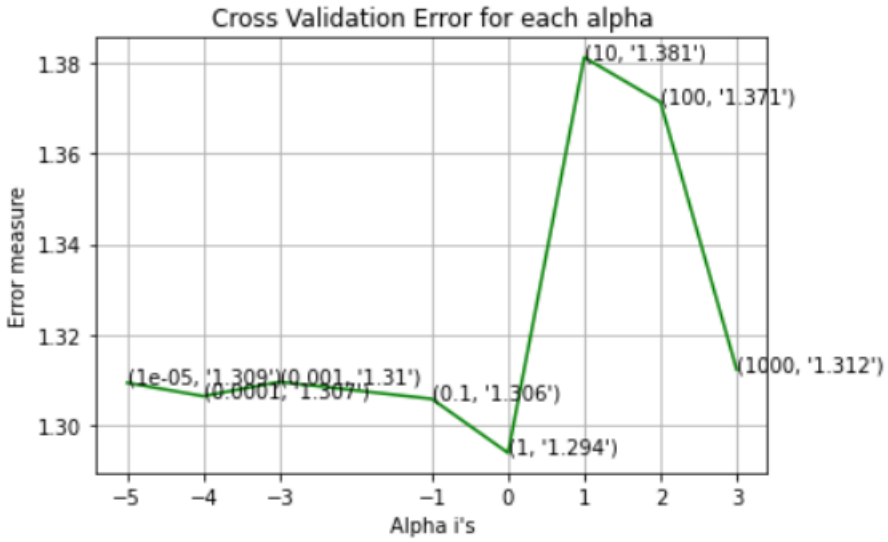
The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

But I have tried to perform modelling via naïve Bayes on my dataset. And it has shown me some great results with one hot encoding method.

It has shown higher interpretability amongst the entire dataset.

Here, are some best results we get from the naïve Bayes model:

```
for alpha = 1e-05
Log Loss : 1.309476619043843
for alpha = 0.0001
Log Loss : 1.3065501751788697
for alpha = 0.001
Log Loss : 1.3096846259534958
for alpha = 0.1
Log Loss : 1.3059348883509991
for alpha = 1
Log Loss : 1.294053984278091
for alpha = 10
Log Loss : 1.381090787451297
for alpha = 100
Log Loss : 1.3712513874082723
for alpha = 1000
Log Loss : 1.3123935984539896
```



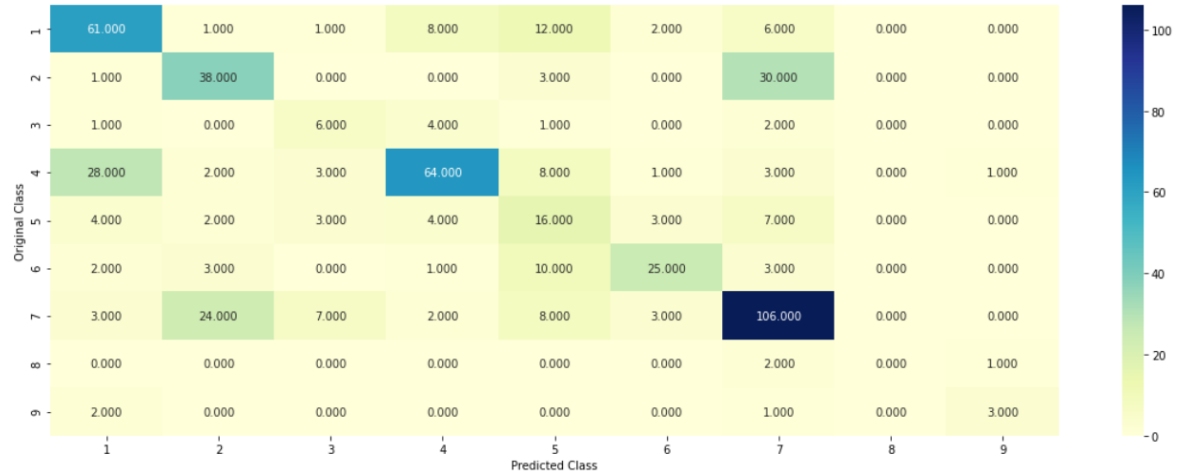
Now, as you can see in the above graph, we have created using the naïve bayes model in this graph it shows for different hyperparameter i.e., α the different log-loss (Error-value). By this we can select the best alpha-value (the lowest log loss value represents the best alpha) to proceed further.

Since, for the best alpha, log loss would be as followed for our train-cv-test dataset:

For values of best alpha = 1 The train log loss is: 0.9129210924813358
For values of best alpha = 1 The cross validation log loss is: 1.29405
3984278091
For values of best alpha = 1 The test log loss is: 1.249327991878956

To check overall performance of the model we need to draw confusion matrix and precision matrix:

```
Log Loss : 1.276018313538779
Number of missclassified point : 0.40037593984962405
-------------------- Confusion matrix --------------------
```




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

Interpretation of above results:

As you can see in the above confusion matrix diagonal elements in the matrix are performing quite well. Whereas, there is some confusion in happening in (2,7) and (3,7) and we can recheck that in precision matrix as well plus there is an entire column which is shown white here i.e., 8th column there reason behind that may be because of insufficient dataset present about the class 8 in the training dataset.

Moving further I am going to build KNN- model.

KNN (K-Nearest Neighbour)- Model

I have done the similar thing to build a model as I had done above the only difference is this time, I am going to use response encoding method because KNN does not perform better with the high dimensionality in the dataset. I had used one-hot encoder as well to run but the problem I have faced that it was taking too much ram memory and the provided results are the worst, I can say worst than my random model.

In naïve Bayes we do have quite good interpretability of our text dataset but in KNN this is one more demerit that interpretability is not so good we do have interpretability but it will be having at very minimal level. So, you should not call it as a interpretable model that's the issue with the KNN-model.

Obviously, there is difference between the code and method but the pattern I am going to follow the same as we have done above so that later on, we can compare these two models as well as the other models I am going to build further.

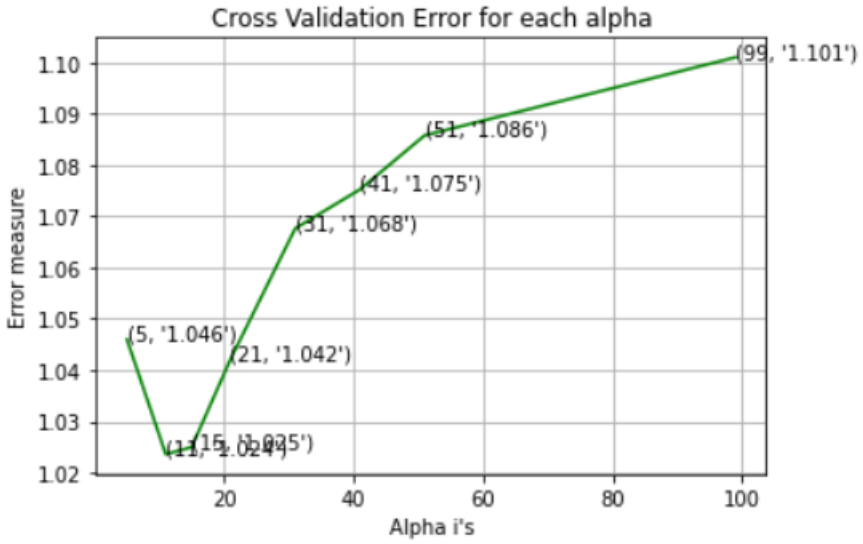
Similarly, I have selected the values of hyperparameter i.e., again alpha but this time my nearest neighbour would be different because here the regularization would be dependent on K, that should I look at 5 nearest neighbours, should I look at 11 nearest neighbour etc…

Now, for every value of alpha I have calculated log-loss here as followed output:

```
    # to avoid rounding error while mult
    print("Log Loss :",log_loss(cv_y, si
```

```
for alpha = 5
Log Loss : 1.0460143720082224
for alpha = 11
Log Loss : 1.0235718862555898
for alpha = 15
Log Loss : 1.0249354537472704
for alpha = 21
Log Loss : 1.0420902066843551
for alpha = 31
Log Loss : 1.067618833101152
for alpha = 41
Log Loss : 1.0752295216256702
for alpha = 51
Log Loss : 1.0857945156816597
for alpha = 99
Log Loss : 1.1010101144043807
```

I have also plotted this for better understanding and we can identify our best alpha as we have also done in the above model.



As you can see here the best alpha, we have is 11 with minimal log-loss.

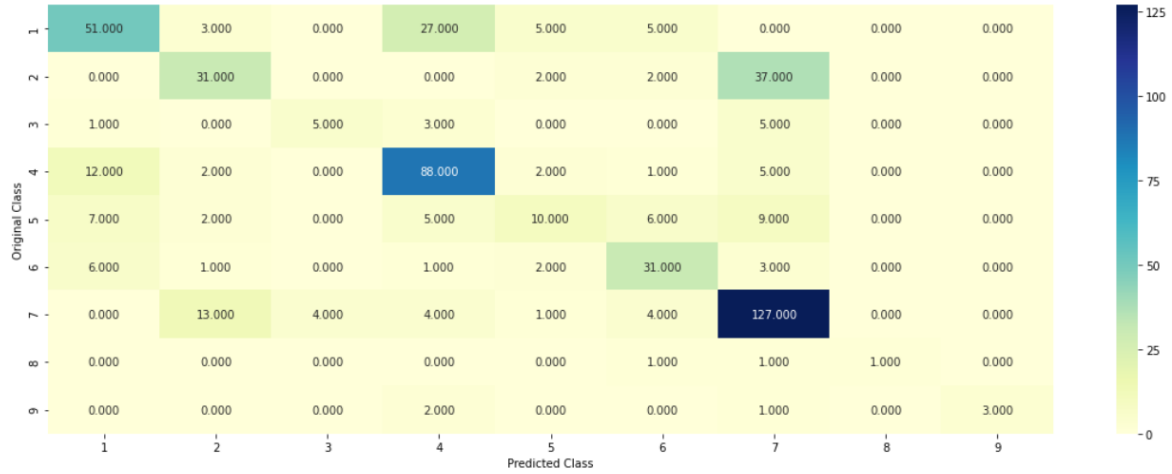
So, for further step I have calculated the log-loss for “train-cv-test” taking alpha is equal to 11.

For values of best alpha = 11 The train log loss is: 0.5905662945764782
For values of best alpha = 11 The cross validation log loss is: 1.0235718862555898
For values of best alpha = 11 The test log loss is: 1.0699959220092043

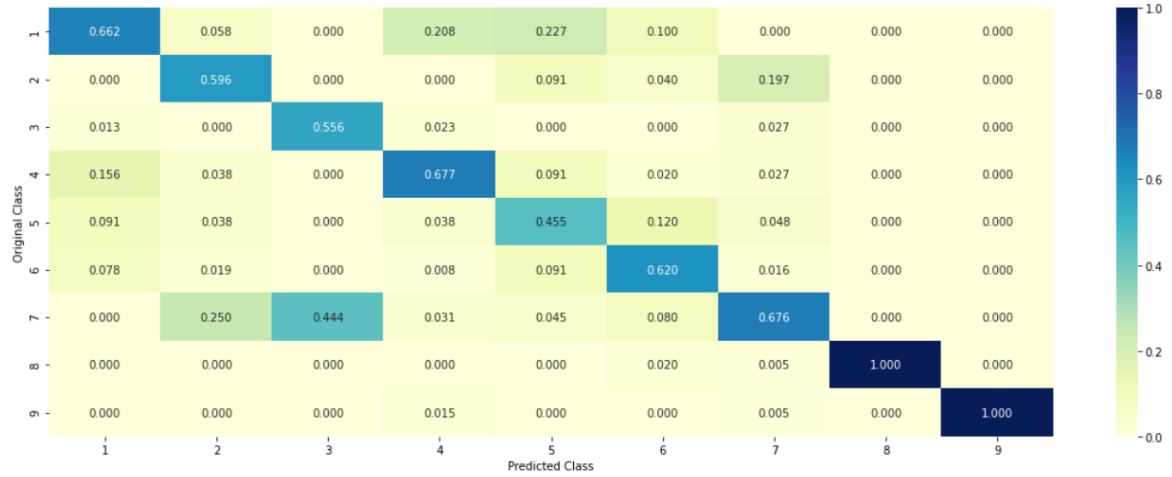
In [91]:

Again, to check the overall performance of the model, I have created the confusion matrix as well as precision matrix as followed:

```
Log loss : 1.0235718862555898
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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



Interpretation of the above results:

Again, if you look at the diagonal elements the entire model is working perfect the only problem I can see here that diagonal element of 8 and 9 in precision matrix have the value 1 and the rest of the column is 0 which shows that column 8 and 9 does not have sufficient data in regards of class 8.

Similarly, here as well there is some confusion in happening in (2,7) and (3,7) and we can recheck that in precision matrix as well plus there is an entire column which is shown white here i.e., 8th column there reason behind that may be because of insufficient dataset present about the class 8 in the training dataset.

After checking the accuracy I have done one more thing with this KNN model that is testing the points, like I took 1 of the query point where I am getting the wrong result and due to insufficient information in training dataset.

Follow the output below:

```
Actual Class : 2
Predicted Class : 1
The  11  nearest neighbours of the test points belongs to classes [1 1 1 1 1 1 1 1 1 1 1]
Fequency of nearest points : Counter({1: 11})
```

Interpretation:

As you can see 1 is occurring all the time so my algorithm thought it will be 1 whereas it should have been the class 2.

NOTE- here test point index was 1

Similarly, I have done this with right class as well where we are correctly predicting, output as followed:

```
Actual Class : 4
Predicted Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [4 4 4 4 4 4 4 4 4 4 1]
Fequency of nearest points : Counter({4: 10, 1: 1})
```

NOTE- here test point index was 80

Interpretation:

As you can see here 4 is predicting most of the times so, it has predicted the class 4.

Logistic Regression Model (with balancing)

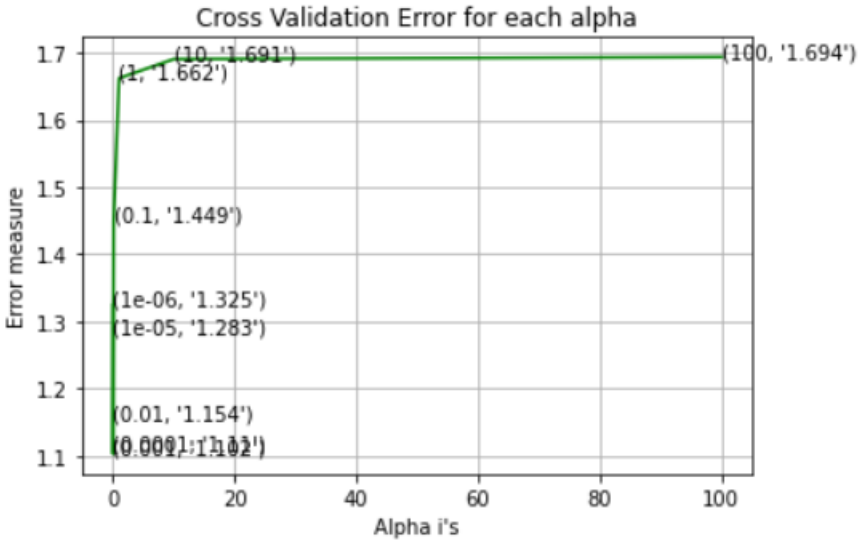
Now, we are gonna follow the same procedure as we have done above just this time while going with the logistic regression model this time I am going to balance all the class because as you have seen in the above models that there is some misbalancing happening in the class 8 and class 9.

Again, first selecting the hyperparametric values i.e., α. Finding the best alpha using the log-loss.

Outputs:

```
for alpha = 1e-06
Log Loss : 1.3247637313659408
for alpha = 1e-05
Log Loss : 1.2828463482455024
for alpha = 0.0001
Log Loss : 1.1096914051239748
for alpha = 0.001
Log Loss : 1.10249816746562
for alpha = 0.01
Log Loss : 1.154411273493875
for alpha = 0.1
Log Loss : 1.4488761919474742
for alpha = 1
Log Loss : 1.6622691602379784
for alpha = 10
Log Loss : 1.6906746602523517
for alpha = 100
Log Loss : 1.6938349992817676
```

Plotting:

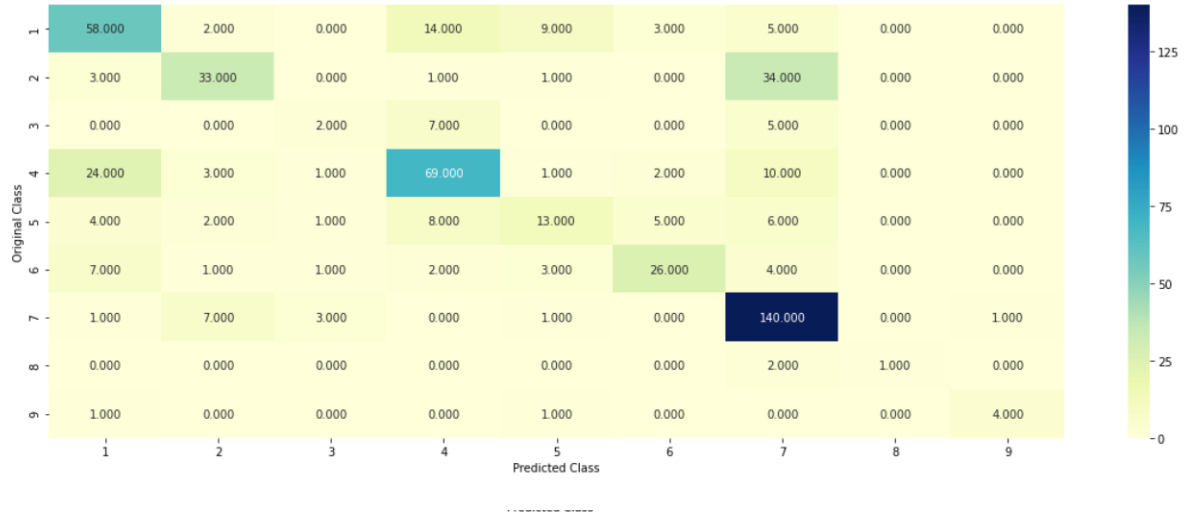


Finding the best alpha, whichever alpha has minimum log-loss will be considered as the best alpha here and we compute the log-loss for train-cv-test with respect to the best alpha.

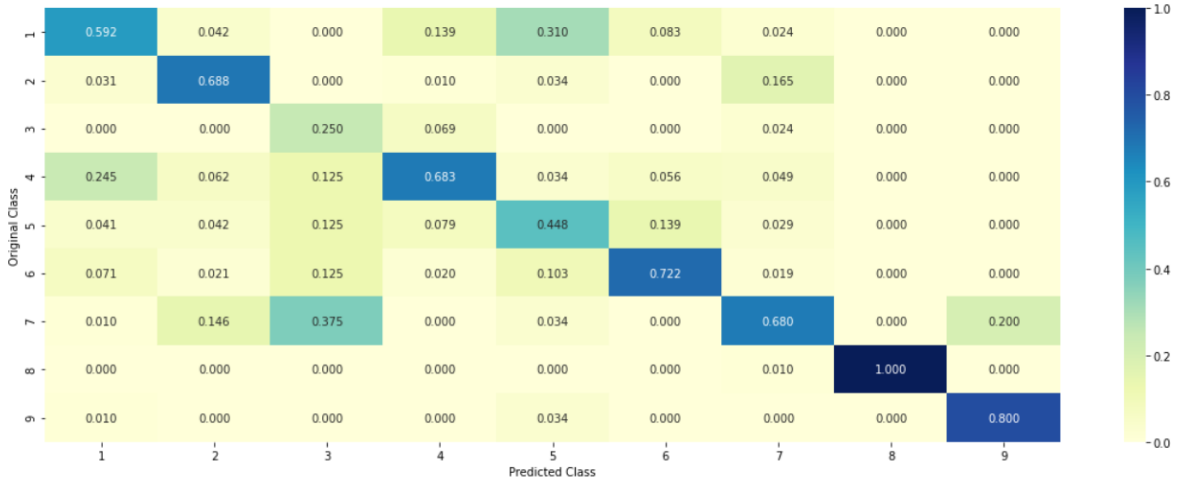
For values of best alpha = 0.001 The train log loss is: 0.5132708693608429
For values of best alpha = 0.001 The cross validation log loss is: 1.10249816746562
For values of best alpha = 0.001 The test log loss is: 1.0725617429162728

Again, to check the overall performance of the model, I have created the confusion matrix as well as precision matrix as followed:

```
Log loss : 1.10249816746562
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




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



Interpretation:

Here not much change we can see here with balancing all classes still the same problems came up with the class (2,7). Not much have been change with balancing all the classes the interpretability is same as we have done for the above models.

Now, the one very good thing with the logistic regression is that it works very good with the interpretability of the model because it considers the weights and coefficients and all those things. Like here in our case,

```
Predicted Class : 1
Predicted Class Probabilities: [[0.8965 0.0244 0.0044 0.0223 0.0153 0.0097 0.0204 0.0045 0.0024]]
Actual Class : 1
--------------------------------------------------
154 Text feature [homophilic] present in test data point [True]
167 Text feature [neutravidin] present in test data point [True]
183 Text feature [sk18] present in test data point [True]
262 Text feature [pcp2] present in test data point [True]
363 Text feature [mam] present in test data point [True]
471 Text feature [frameshifts] present in test data point [True]
490 Text feature [deletion] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

Like here you can see that for the class 1st probability is around 89.65% that why my prediction is this much correct.

I have gone through with the wrong predictions as well where the results are as followed:

```
Predicted Class : 5
Predicted Class Probabilities: [[0.2244 0.0146 0.0214 0.2238 0.4818 0.0175 0.0091 0.0053 0.002 ]]
Actual Class : 1
--------------------------------------------------
227 Text feature [v1833m] present in test data point [True]
383 Text feature [pathogenicity] present in test data point [True]
445 Text feature [probabilities] present in test data point [True]
467 Text feature [unclassified] present in test data point [True]
476 Text feature [kconfab] present in test data point [True]
480 Text feature [likelihood] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

Here, the fifth-class probability is higher maybe due to lack of presence class 1 data in the training dataset it has predicted the most occurring class.

Logistic Regression Model (without balancing)

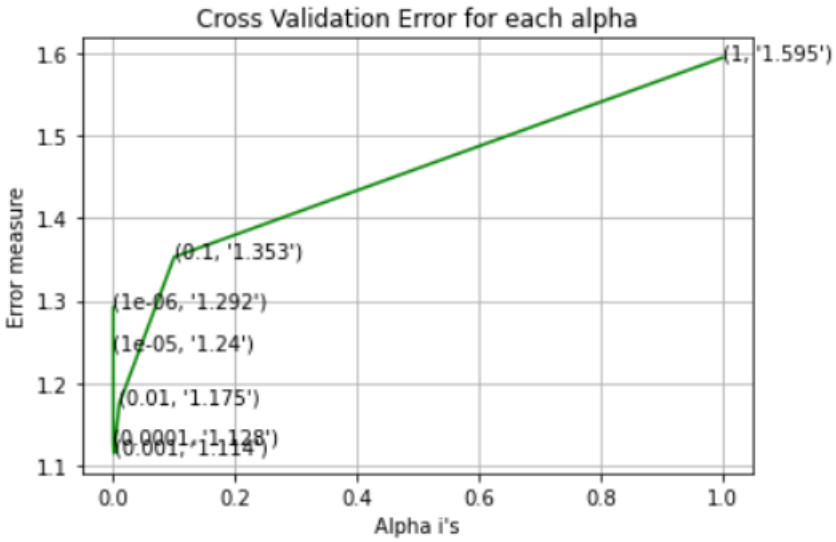
Now, we are going to do same thing as we have done in the logistic regression model above just there is only one difference in this model is that this time, we are not going to balance the classes here, rest of the procedure will be similar.

Again, first selecting the hyperparametric values i.e., α. Finding the best alpha using the log-loss.

Output:

```
for alpha = 1e-06
Log Loss : 1.2916402837895267
for alpha = 1e-05
Log Loss : 1.2402152001149287
for alpha = 0.0001
Log Loss : 1.127583334056951
for alpha = 0.001
Log Loss : 1.1143910352659776
for alpha = 0.01
Log Loss : 1.1747978222108189
for alpha = 0.1
Log Loss : 1.3529037749875312
for alpha = 1
Log Loss : 1.5948181604090124
```

Plot:



Finding the best alpha, whichever alpha has minimum log-loss will be considered as the best alpha here and we compute the log-loss for train-cv-test with respect to the best alpha.

```
For values of best alpha =  0.001 The train log loss is: 0.5160712097700929
For values of best alpha =  0.001 The cross validation log loss is: 1.1143910352659776
For values of best alpha =  0.001 The test log loss is: 1.0915155931611387
```

Again, to check the overall performance of the model, I have created the confusion matrix as well as precision matrix as followed:

```
Log loss : 1.1143910352659776
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```

Interpretation:

It is very strange for me to see here that the miss-classified points are less in the same model without balancing in comparison to with balancing but we can also here say that balancing is not a good choice always sometimes model performs better without balancing as well and our case is the live example here.

Now, I am going to comment about the interpretability of this model

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0998 0.6379 0.0025 0.09   0.0148 0.0109 0.1352 0.008  0.0009]]
Actual Class : 4
--------------------------------------------------
347 Text feature [eyelashes] present in test data point [True]
389 Text feature [necessitating] present in test data point [True]
404 Text feature [sagittal] present in test data point [True]
435 Text feature [knee] present in test data point [True]
462 Text feature [roberts] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

This is the output for wrong dataset where you can clearly see that probability of class 2 is very high that is why our model has predicted the class 2.

Now, again we do this with the correct dataset values:

```
Predicted Class : 6
Predicted Class Probabilities: [[2.300e-03 5.000e-03 3.200e-03 1.076e-01 2.800e-03 8.724e-01 2.200e-03
  3.900e-03 8.000e-04]]
Actual Class : 6
--------------------------------------------------
161 Text feature [s80] present in test data point [True]
201 Text feature [spermatogenesis] present in test data point [True]
244 Text feature [ccdc98] present in test data point [True]
294 Text feature [weakened] present in test data point [True]
326 Text feature [hospitals] present in test data point [True]
345 Text feature [sterility] present in test data point [True]
361 Text feature [zn2] present in test data point [True]
366 Text feature [c44r] present in test data point [True]
367 Text feature [g98r] present in test data point [True]
368 Text feature [c61l] present in test data point [True]
369 Text feature [k45e] present in test data point [True]
370 Text feature [r7c] present in test data point [True]
371 Text feature [l82p] present in test data point [True]
372 Text feature [l87v] present in test data point [True]
373 Text feature [7502] present in test data point [True]
374 Text feature [ddi476] present in test data point [True]
375 Text feature [l82d] present in test data point [True]
376 Text feature [k65m] present in test data point [True]
377 Text feature [l63f] present in test data point [True]
378 Text feature [k56n] present in test data point [True]
379 Text feature [q60l] present in test data point [True]
380 Text feature [k50l] present in test data point [True]
381 Text feature [kings] present in test data point [True]
382 Text feature [m18k] present in test data point [True]
383 Text feature [k38n] present in test data point [True]
384 Text feature [82579] present in test data point [True]
385 Text feature [l28p] present in test data point [True]
386 Text feature [q12y] present in test data point [True]
387 Text feature [c47r] present in test data point [True]
388 Text feature [c39r] present in test data point [True]
389 Text feature [c39s] present in test data point [True]
390 Text feature [brca1fh] present in test data point [True]
```

Here we have done with the correct where I got the predicted and actual class is same. I have done nothing played with the test_point_index in the code, it is very similar to bias-variance tradeoff just take control over the biasedness and variability and you'll get the best results otherwise it may lead to the problem of overfitting / underfitting.

Random Forest Model(one-hot encoding)

Now, further moving to our last model i.e., random forest. This time I am going to choose two hyperparameters for random forest because hyperparameters include the number of decision trees in the forest and the number of features considered by each tree when splitting a node. These parameters would be maximum depth and our regularization parameter (i.e., α).

We will apply the same procedure that we did in the above models. We are going to find out the best hyperparameter with a minimal log-loss and then with respect to that hyperparameter we will find our log-loss for train-cv-test.

One more thing is that this model is generally works well with both the method i.e., one-hot encoding method as well as response encoding method. Let's see how well did it perform with my dataset.

First, I am going with the outputs evaluated using one-hot encoding method:

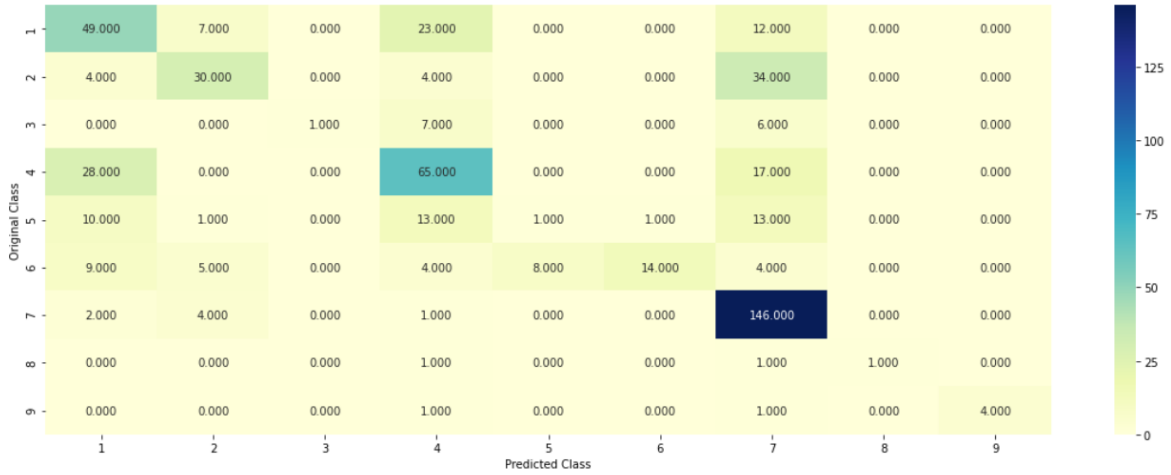
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.247803692220492
for n_estimators = 100 and max depth =  10
Log Loss : 1.1845022270056835
for n_estimators = 200 and max depth =  5
Log Loss : 1.233653803138955
for n_estimators = 200 and max depth =  10
Log Loss : 1.171495635569231
for n_estimators = 500 and max depth =  5
Log Loss : 1.228954888902214
for n_estimators = 500 and max depth =  10
Log Loss : 1.1697357159989832
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2294501466339345
for n_estimators = 1000 and max depth =  10
Log Loss : 1.169408152468762
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2296553014389262
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1664581386634412
```

The minimal log-loss will decide our best hyperparameter:

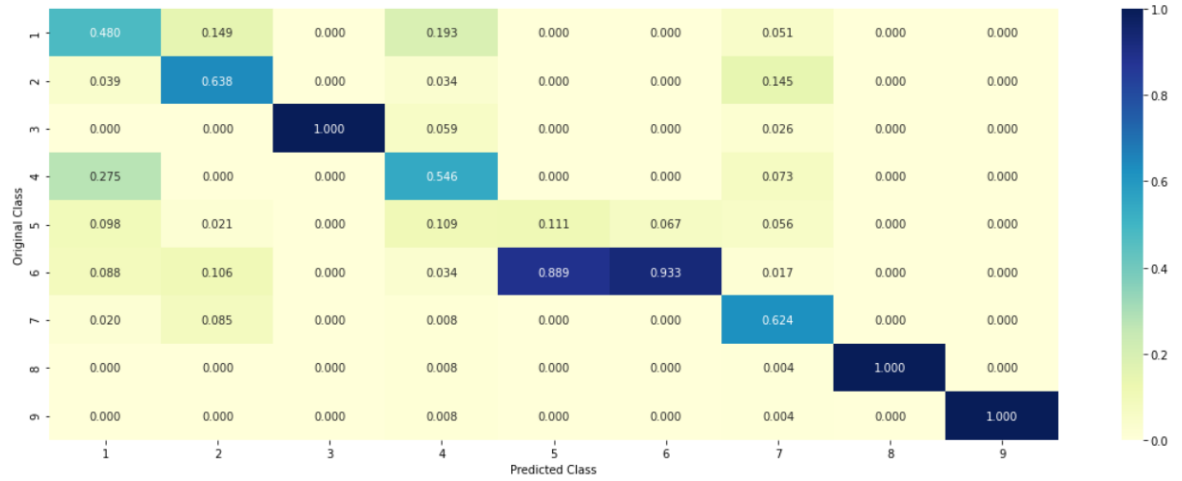
```
For values of best estimator =  2000 The train log loss is: 0.7085706760504246
For values of best estimator =  2000 The cross validation log loss is: 1.1664581386634412
For values of best estimator =  2000 The test log loss is: 1.1625527980101846
```

Again, to check the overall performance of the model, I have created the confusion matrix as well as precision matrix as followed:

```
Log loss : 1.1664581386634412
Number of mis-classified points : 0.41541353383458646
-------------------- Confusion matrix --------------------
```



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



Here, we can see less confusion occurring in the matrix but at the same time if you look at the misclassified points, the percentage is very high in comparison to the other model. So, definitely there is some problem occurring in this model.

Now, just like naïve Bayes and logistic regression random forest also shows the good interpretability amongst the dataset.

So, taking to the query points and some predicted class for the points.

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3818 0.1338 0.0253 0.2066 0.0797 0.0799 0.0759 0.0085 0.0084]]
Actual Class : 1
--------------------------------------------------
8 Text feature [treatment] present in test data point [True]
20 Text feature [loss] present in test data point [True]
31 Text feature [cells] present in test data point [True]
40 Text feature [pathogenic] present in test data point [True]
49 Text feature [deleterious] present in test data point [True]
51 Text feature [patients] present in test data point [True]
52 Text feature [variants] present in test data point [True]
60 Text feature [cell] present in test data point [True]
64 Text feature [expression] present in test data point [True]
67 Text feature [variant] present in test data point [True]
68 Text feature [neutral] present in test data point [True]
71 Text feature [unclassified] present in test data point [True]
77 Text feature [repair] present in test data point [True]
78 Text feature [advanced] present in test data point [True]
82 Text feature [wild] present in test data point [True]
84 Text feature [truncating] present in test data point [True]
86 Text feature [uncertain] present in test data point [True]
87 Text feature [protein] present in test data point [True]
89 Text feature [tumors] present in test data point [True]
91 Text feature [clinical] present in test data point [True]
96 Text feature [dna] present in test data point [True]
Out of the top  100  features  21 are present in query point
```

These are the query points with better interpretation of class 1 points. And it has accurately predicted the class 1 with the respective probabilities.

Random Forest Model (response encoding)

The procedure will the same just this time we going to use response encoding method.

We are going to find out the best hyperparameter with a minimal log-loss and then with respect to that hyperparameter we will find our log-loss for train-cv-test.

Output:

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.077550542176198
for n_estimators = 10 and max depth =  3
Log Loss : 1.587277140041427
for n_estimators = 10 and max depth =  5
Log Loss : 1.3838624937104267
for n_estimators = 10 and max depth =  10
Log Loss : 1.7545698105247023
for n_estimators = 50 and max depth =  2
Log Loss : 1.7060693056233291
for n_estimators = 50 and max depth =  3
Log Loss : 1.446202675141865
for n_estimators = 50 and max depth =  5
Log Loss : 1.3476695171647881
for n_estimators = 50 and max depth =  10
Log Loss : 1.635846748705175
for n_estimators = 100 and max depth =  2
Log Loss : 1.5424847647865376
for n_estimators = 100 and max depth =  3
Log Loss : 1.463691973435371
for n_estimators = 100 and max depth =  5
Log Loss : 1.2955229719356183
for n_estimators = 100 and max depth =  10
Log Loss : 1.6907989596526016
for n_estimators = 200 and max depth =  2
Log Loss : 1.572293392698602
for n_estimators = 200 and max depth =  3
Log Loss : 1.4684247456965966
for n_estimators = 200 and max depth =  5
Log Loss : 1.3319197256949469
for n_estimators = 200 and max depth =  10
Log Loss : 1.7046618358806598
for n_estimators = 500 and max depth =  2
Log Loss : 1.6359566563131112
for n_estimators = 500 and max depth =  3
Log Loss : 1.5190754531569863
for n_estimators = 500 and max depth =  5
Log Loss : 1.3823502863116464
for n_estimators = 500 and max depth =  10
Log Loss : 1.7258250139294513
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6203739019568477
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5138867221739112
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3794095082554572
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6953748266120692
For values of best alpha =  100 The train log loss is: 0.06263806376143385
For values of best alpha =  100 The cross validation log loss is: 1.2955229719356183
For values of best alpha =  100 The test log loss is: 1.301646045747502
```

Again, to check the overall performance of the model, I have created the confusion matrix as well as precision matrix as followed:

```
Log loss : 1.2955229719356183
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```


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

As you can see in the above matrix there are so many confusions arises and misclassified percentage is also too high, so from accuracy point of view this model is not performing well and we have much better models which performing far better than this.

Interpretability of this model:

Query point classification:

```
Predicted Class : 8
Predicted Class Probabilities: [[0.0507 0.2081 0.0805 0.0534 0.0402 0.0604 0.0288 0.3462 0.1318]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
```

Wrong prediction is happening here most of the time it is predicting 8 with respect to probabilities.

I can say this is the worst performing model in comparison to the all other models we have evaluated.

Chapter 5

Conclusion

After working with all the machine learning models, I have evaluated above I build a table to evaluate the best model working for this dataset and the worst model to this dataset.

The following table is:

Models	Train	Cross-Validation	Test	Missclassified %
Naïve Bayes (one-hot encoding)	0.863	1.25	1.265	39.28
KNN (response encoding)	0.455	1.05	1.13	36.28
Logistic Regression (with balancing)	0.513	1.102	1.073	34.962
Logistic Regression (without balancing)	0.516	1.114	1.092	33.835
Random Forest (one-hot encoding)	0.709	1.166	1.163	41.54
Random Forest (response encoding)	0.063	1.296	1.302	42.67

This table shows the log-loss for train-cv-test dataset and the percent of misclassified points in the models.

As per accuracy point of view this table clearly shows that **logistic regression without balancing** is the best model to consider where as for minimal log-loss point of view we go for **logistic regression with balancing** though there is not much difference in the prediction and precision. So, my suggestion would be the **logistic regression without balancing.**

Talking about the worst performing model is as you can see Random forest with response encoding gave the worst results so far and there is so much overfitting happening in the model as you can see.

What more I could have done?

The answer to this is I could have tried to minimize the log-loss more by using the deep learning algorithms as well as I could have gone with the bag of words for text data etc.

Bibliography

- An Introduction to Statistical learning by Gareth James
- GitHub
- Sci-kit learn
- Pattern Recognition and Machine Learning by Christopher M.