**BUILDING A SUITABLE MACHINE LEARNING ALGORITHM FOR OUR DATASET**

**NAÏVE BAYE’S MODEL**

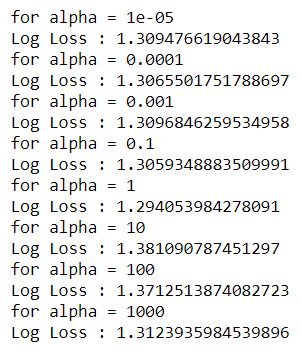
Here in this model we are going to use naïve bayes classifier for multinomial models.

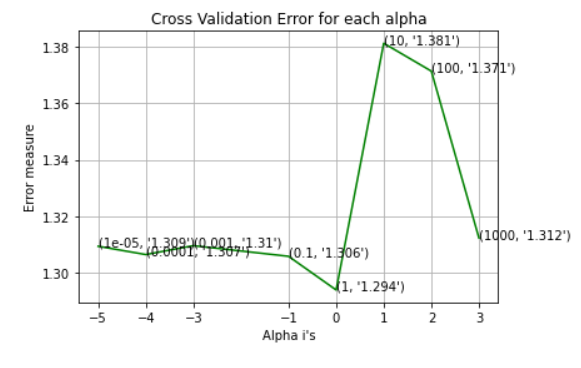
The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.

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:





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:

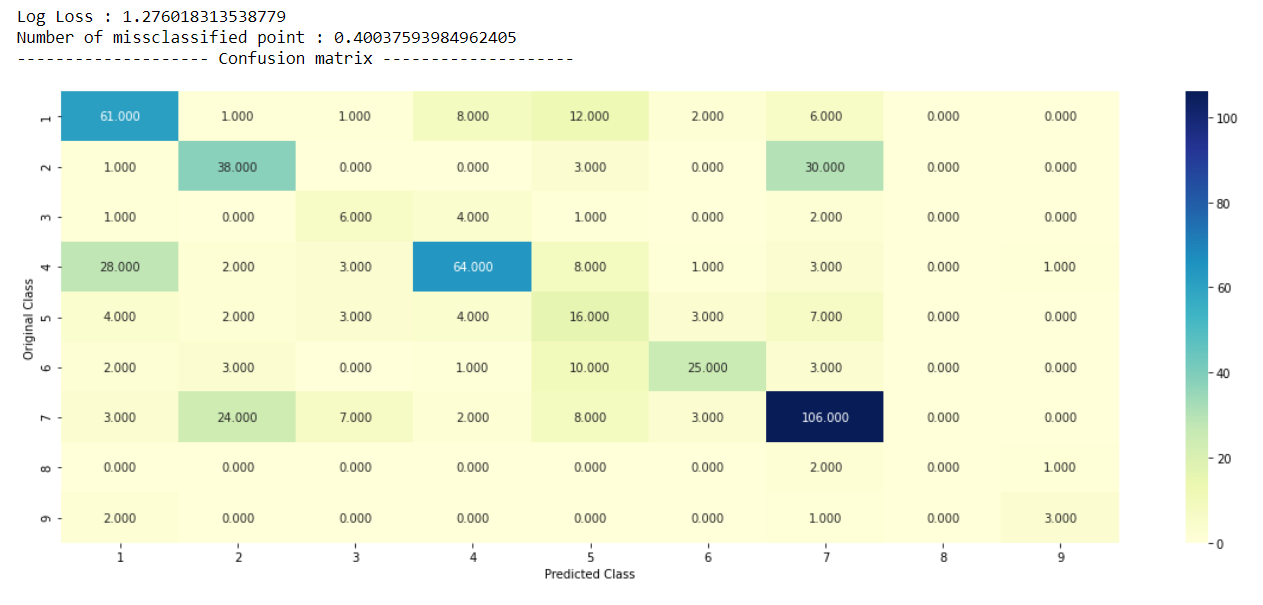
For values of best alpha = 1 The train log loss is: 0.9129210924813358

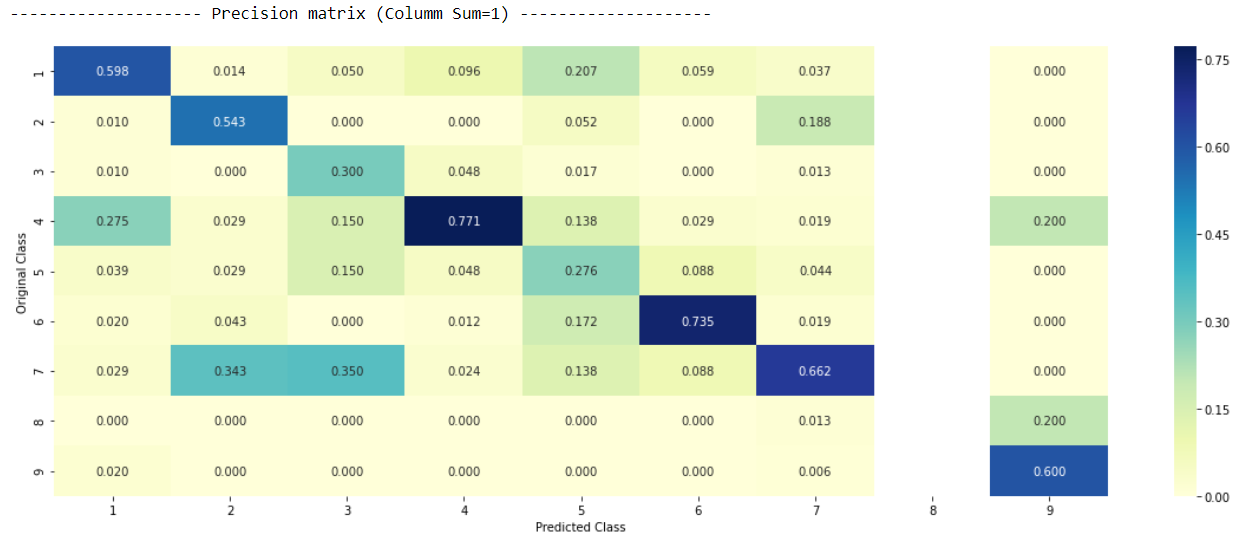
For values of best alpha = 1 The cross validation log loss is: 1.29405

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:





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**

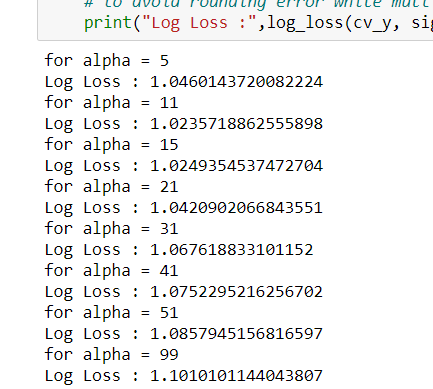
I have done the similar thing to build a model as I had done above the only difference is this time, I am going to use response encoding method because KNN does not perform better with the high dimensionality in the dataset. I had used one-hot encoder as well to run but the problem I have faced that 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.

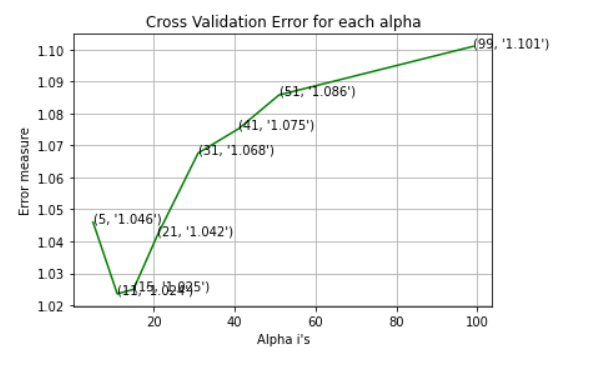
Obviously, there is difference between the code and method but the pattern I am going to follow the same as we have done above so that later on, we can compare these two models as well as the other models I am going to build further.

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:



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.

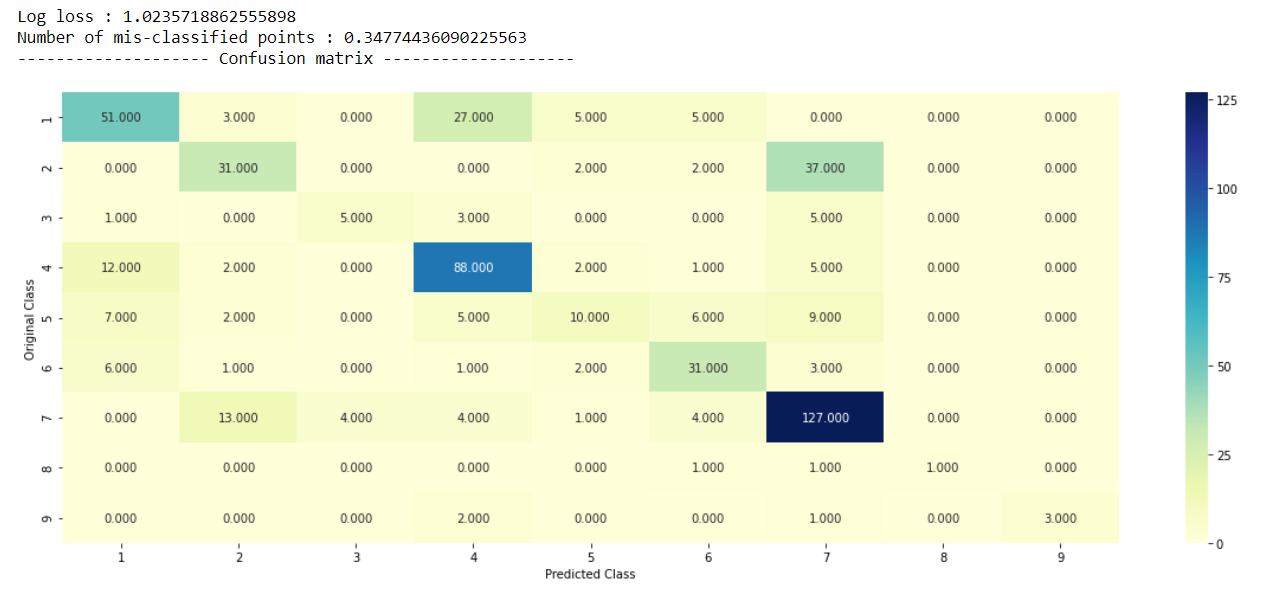
For values of best alpha = 11 The train log loss is: 0.5905662945764782

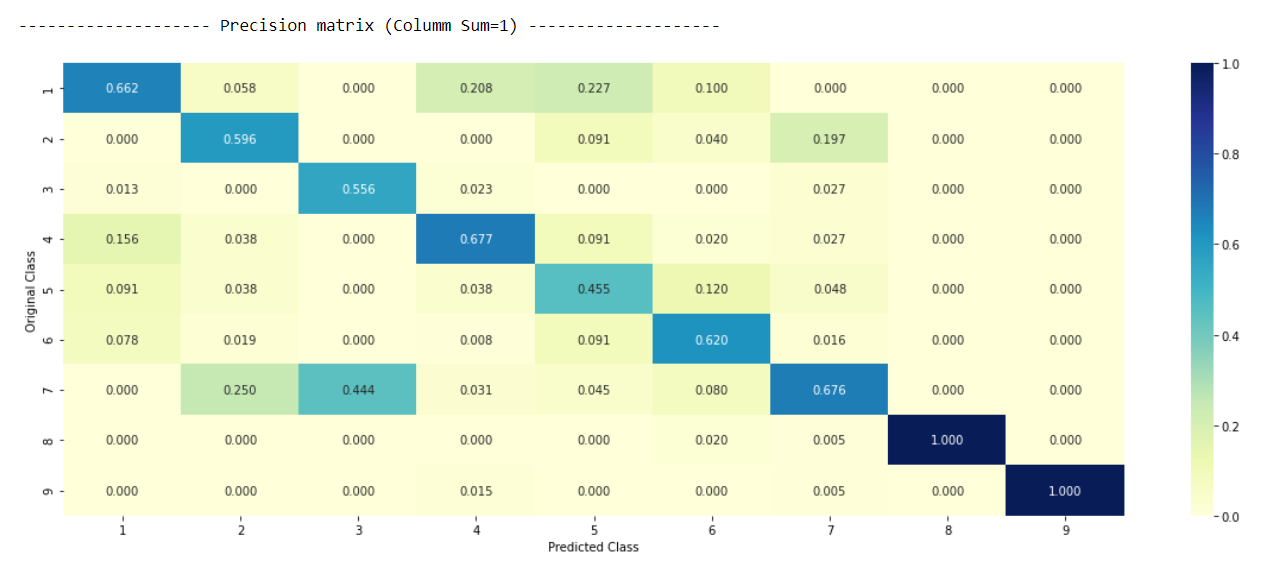
For values of best alpha = 11 The cross validation log loss is: 1.0235718862555898

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:



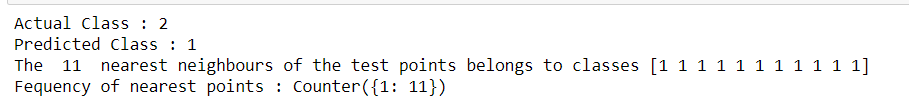


Interpretation of the above results:

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.

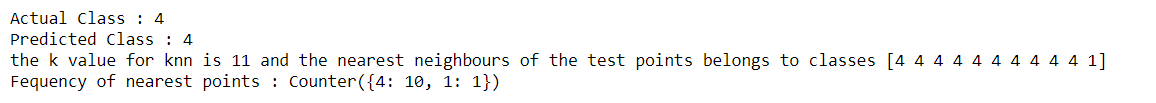
Follow the output below:

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:



NOTE- here test point index was 80

Interpretation:

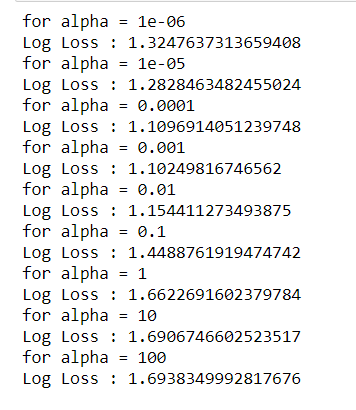
As you can see here 4 is predicting most of the times so, it has predicted the class 4.

**Logistic Regression Model (with balancing)**

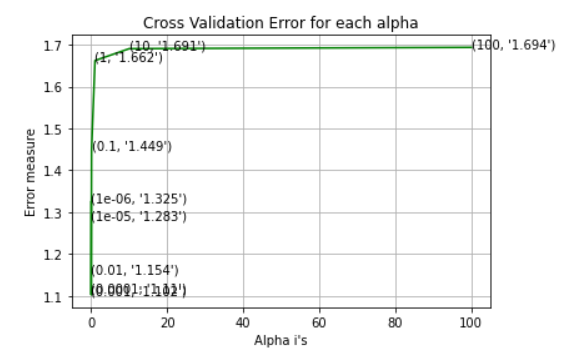
Now, we are gonna follow the same procedure as we have done above just this time while going with the logistic regression model this time I am going to balance all the class because as you have seen in the above models that there is some misbalancing happening in the class 8 and class 9.

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

Outputs:



Plotting:



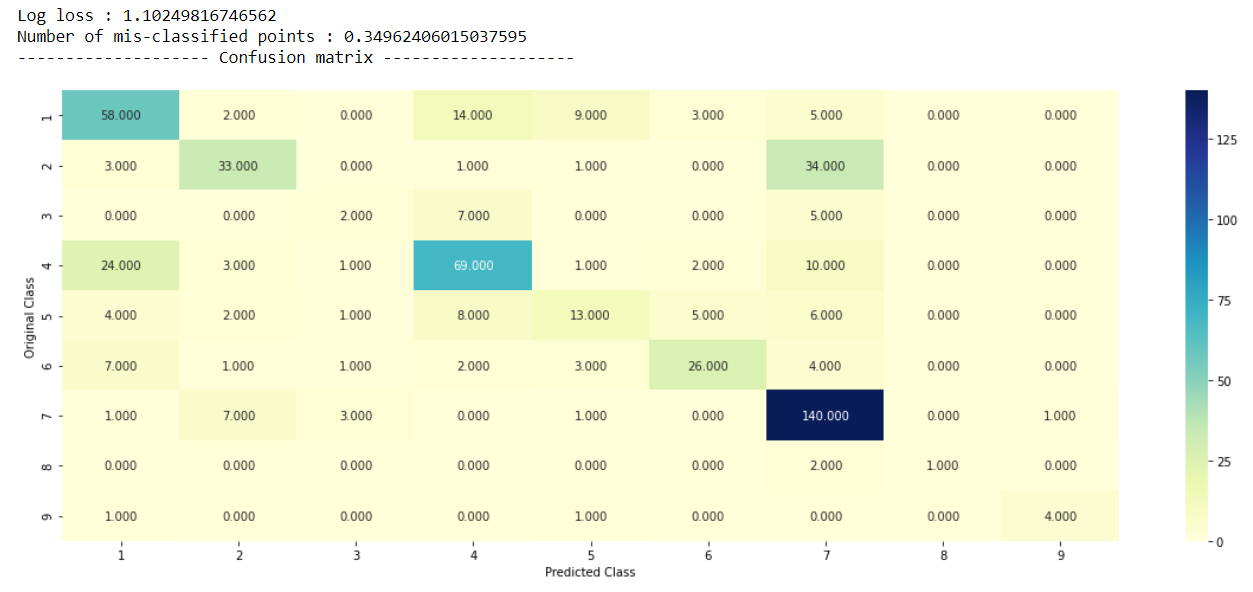
Finding the best alpha, whichever alpha has minimum log-loss will be considered as the best alpha here and we compute the log-loss for train-cv-test with respect to the best alpha.

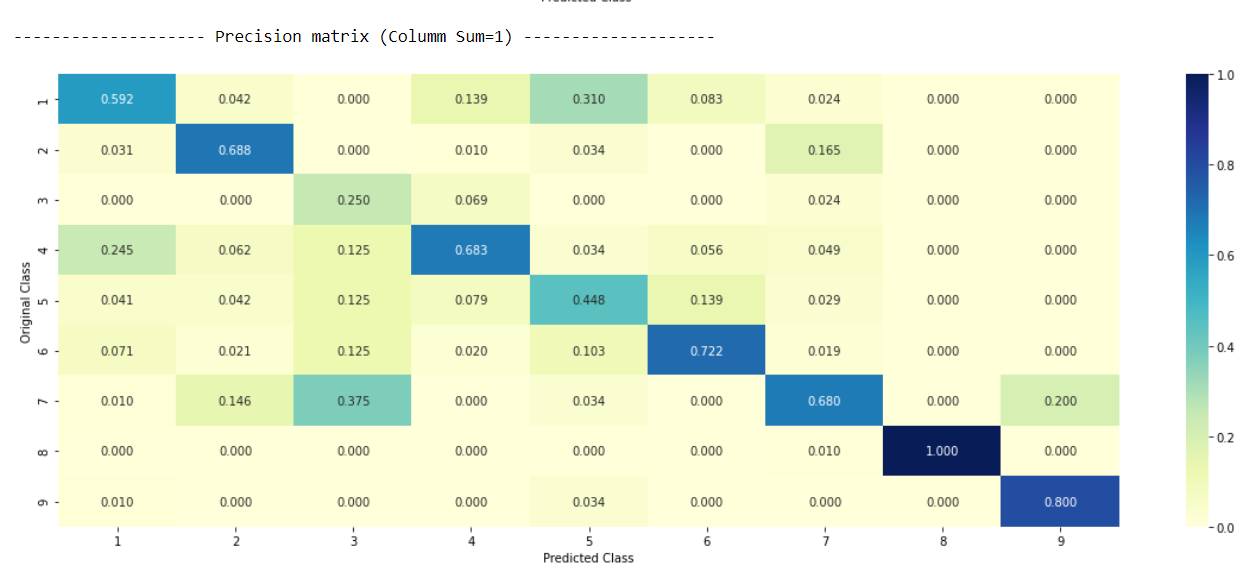
For values of best alpha = 0.001 The train log loss is: 0.5132708693608429

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:

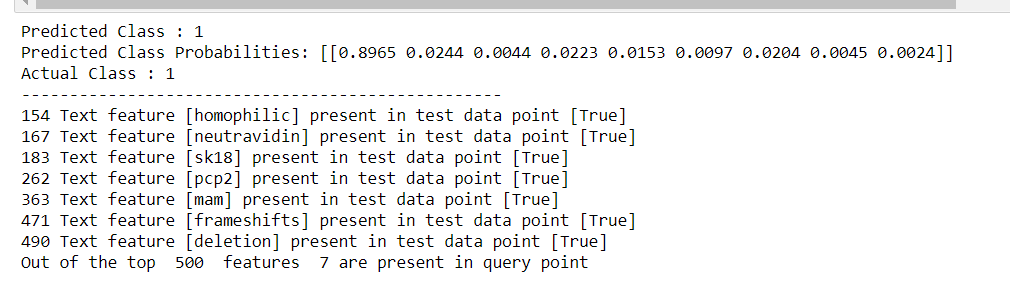




Interpretation:

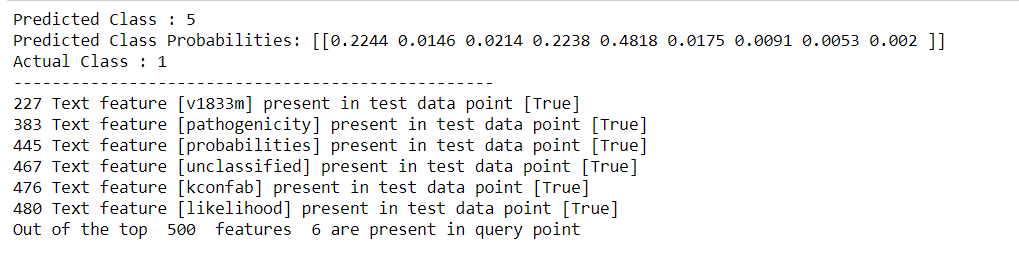
Here not much change we can see here with balancing all classes still the same problems came up with the class (2,7). Not much have been change with balancing all the classes the interpretability is same as we have done for the above models.

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,



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:



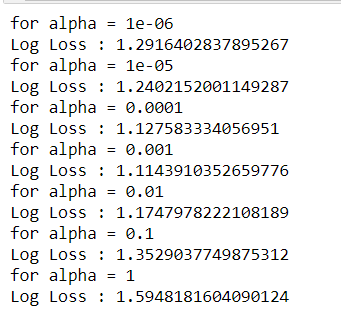
Here, the fifth-class probability is higher maybe due to lack of presence class 1 data in the training dataset it has predicted the most occurring class.

**Logistic Regression Model (without balancing)**

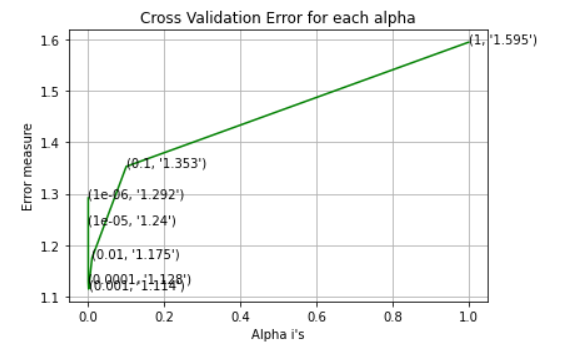
Now, we are going to do same thing as we have done in the logistic regression model above just there is only one difference in this model is that this time, we are not going to balance the classes here, rest of the procedure will be similar.

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

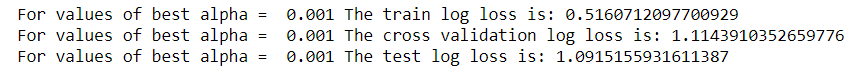
Output:



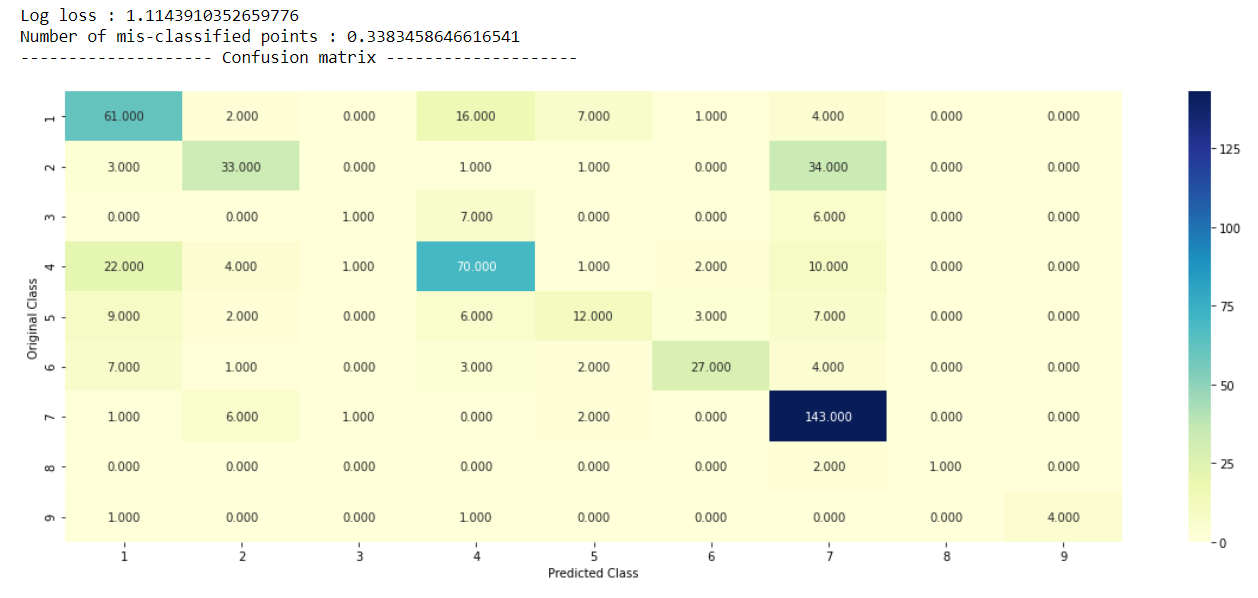
Plot:

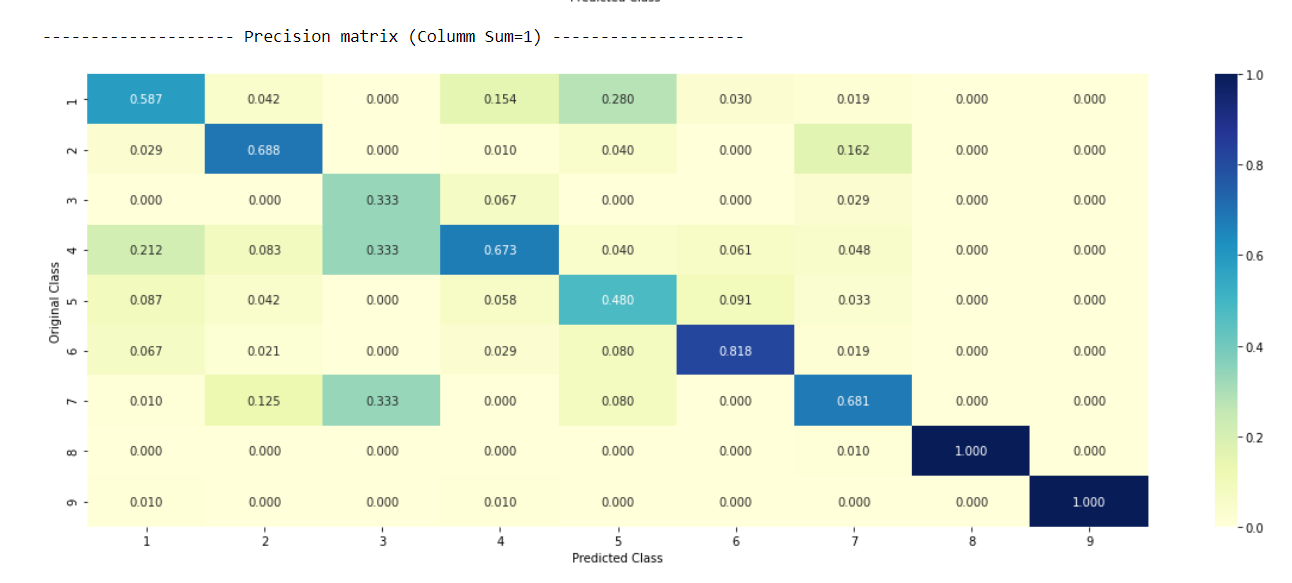


Finding the best alpha, whichever alpha has minimum log-loss will be considered as the best alpha here and we compute the log-loss for train-cv-test with respect to the best alpha.



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

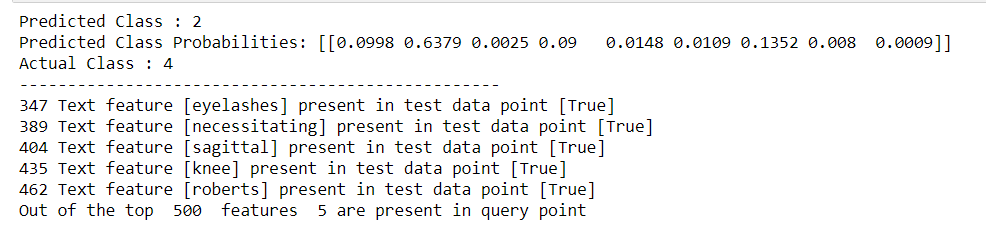




Interpretation:

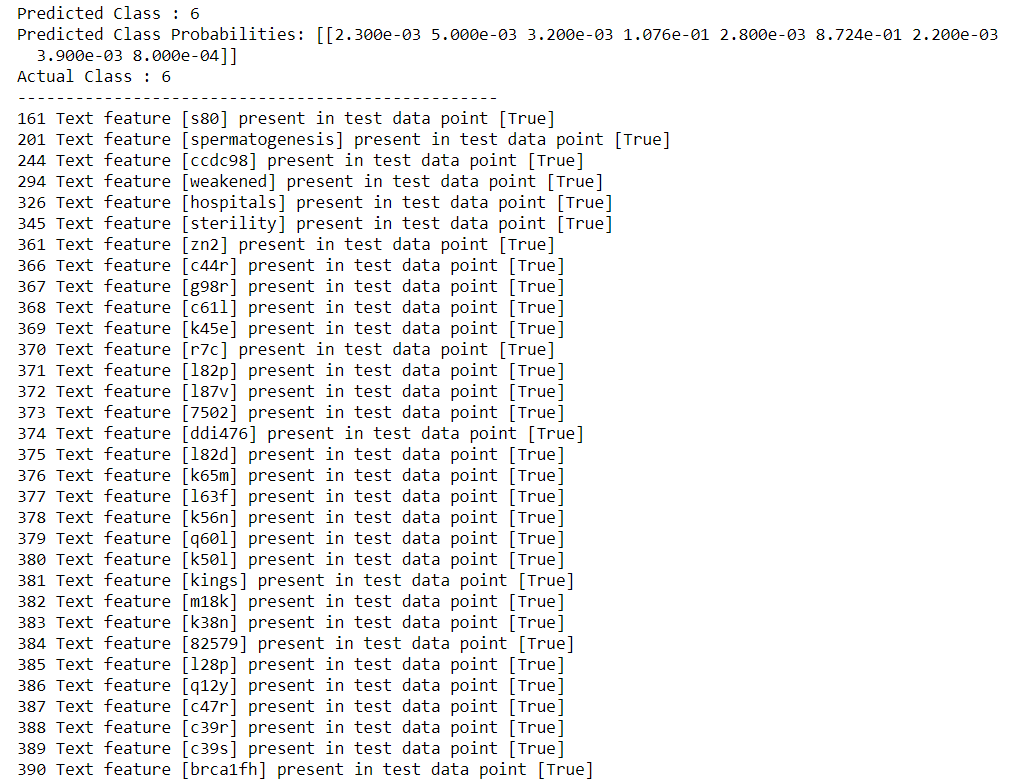
It is very strange for me to see here that the miss-classified points are less in the same model without balancing in comparison to with balancing but we can also here say that balancing is not a good choice always sometimes model performs better without balancing as well and our case is the live example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

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



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:



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.

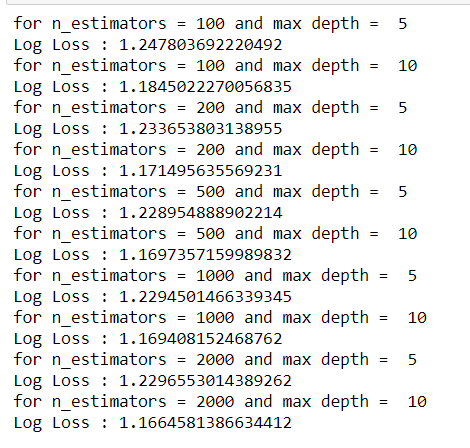
**Random Forest Model(one-hot encoding)**

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).

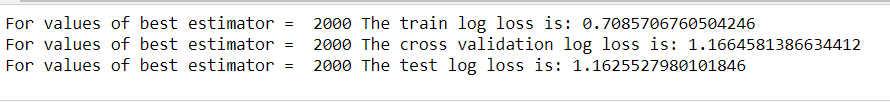
We will apply the same procedure that we did in the above models. We are going to find out the best hyperparameter with a minimal log-loss and then with respect to that hyperparameter we will find our log-loss for train-cv-test.

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.

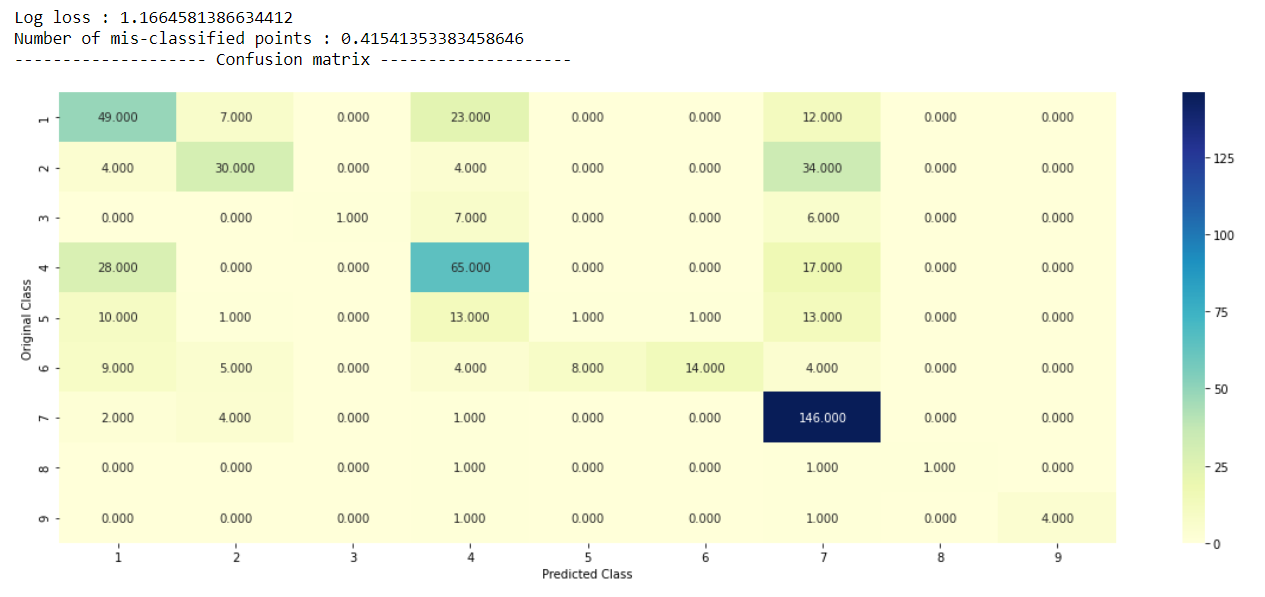
First, I am going with the outputs evaluated using one-hot encoding method:

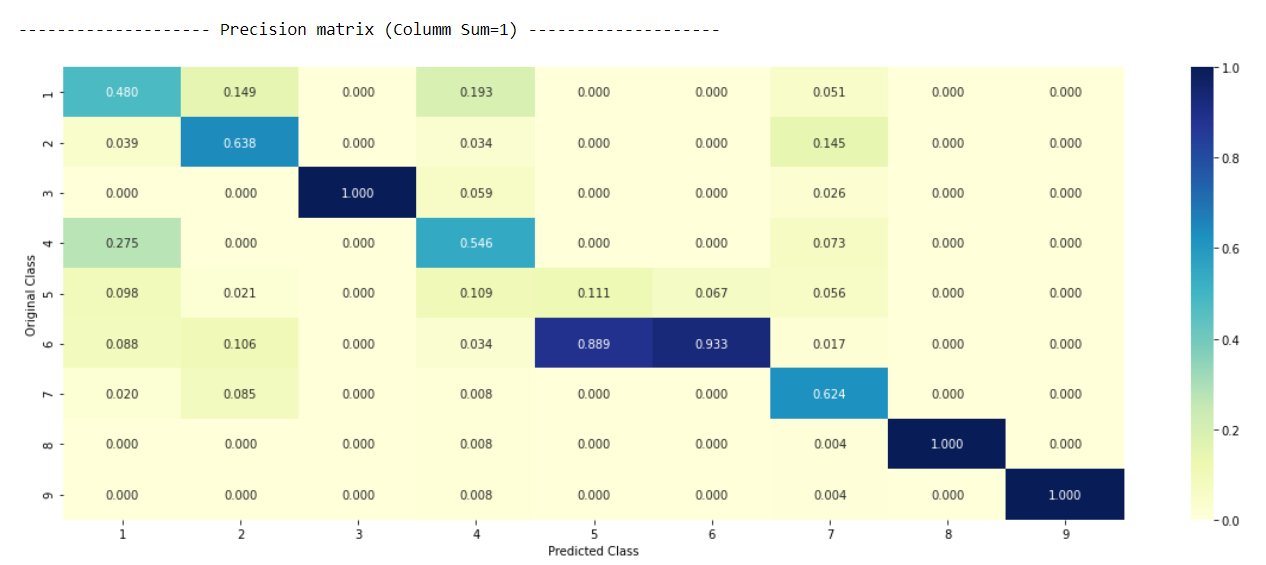


The minimal log-loss will decide our best hyperparameter:



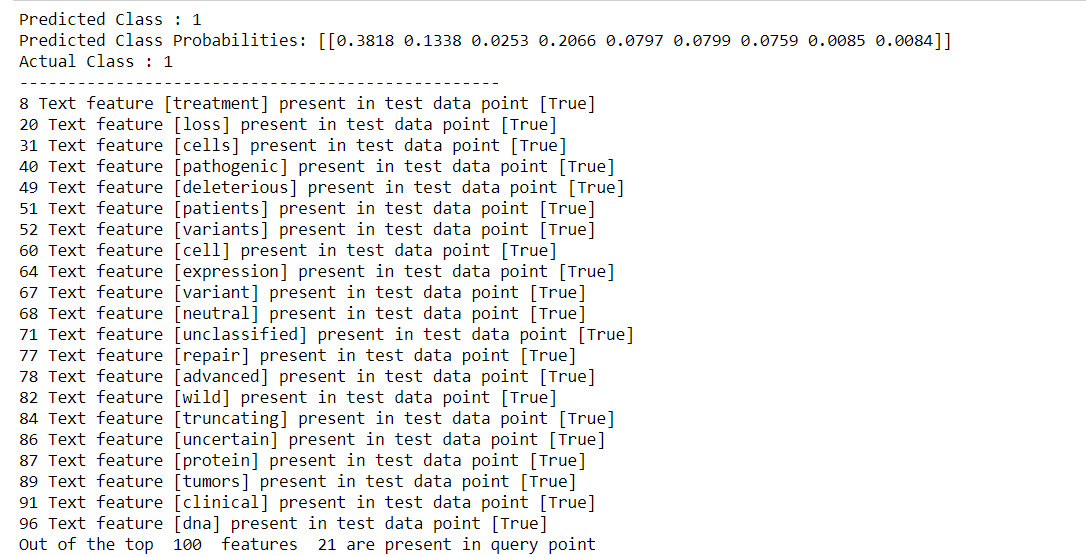
Again, to check the overall performance of the model, I have created the confusion matrix as well as precision matrix as followed:





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. 

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

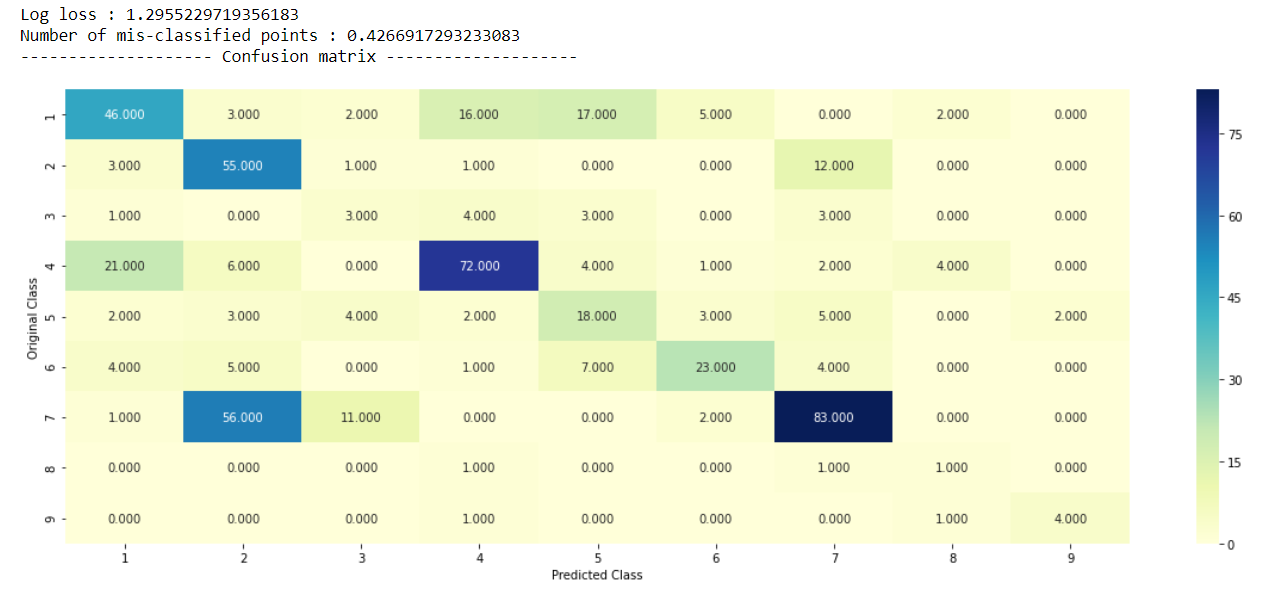
Log Loss : 1.6953748266120692

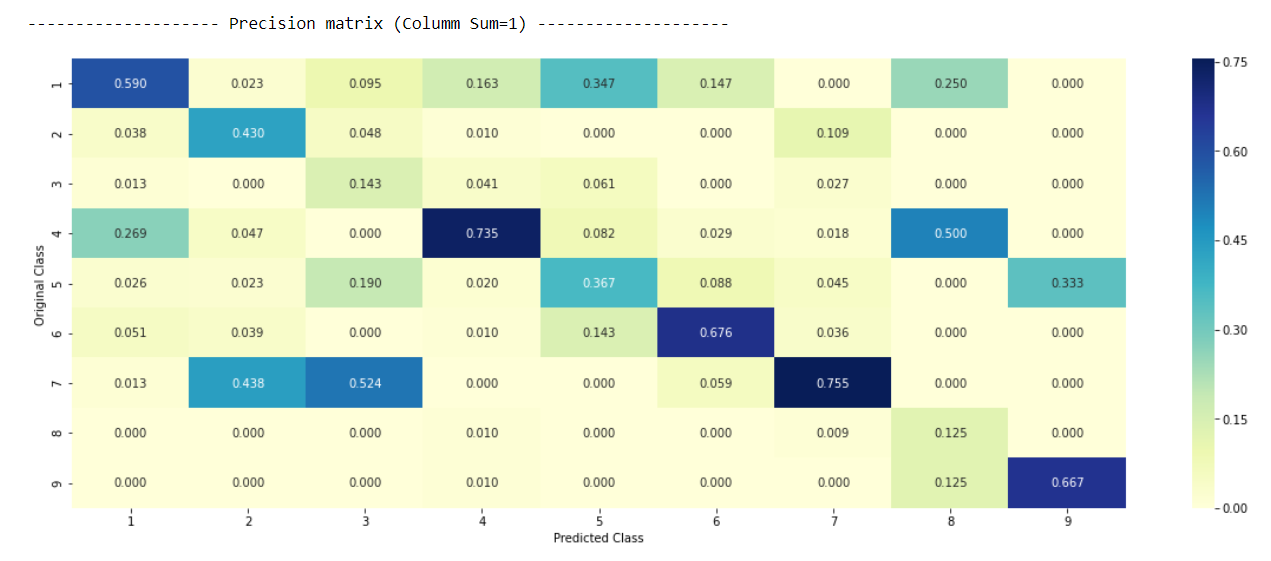
For values of best alpha = 100 The train log loss is: 0.06263806376143385

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:

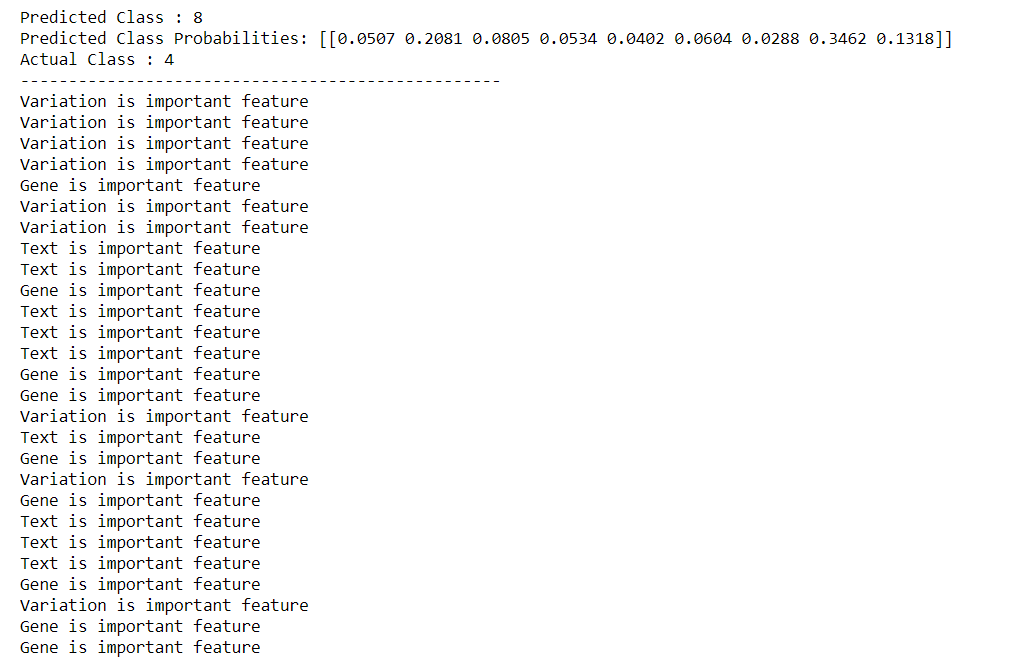




As you can see in the above matrix there are so many confusions arises and misclassified percentage is also too high, so from accuracy point of view this model is not performing well and we have much better models which performing far better than this.

Interpretability of this model:

Query point classification:



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.