<u>Multiclass Classification Using Perceptron</u>

<u>Overview:</u>

The goal of this task is to select, implement and evaluate a machine learning algorithm. The selected algorithm is perceptron and its design and implementational assumption is mentioned in further sections.

<u>Tools/Coding language used:</u> R Studio, R

<u>Algorithm Selected:</u> Perceptron (One vs One approach) with SGD to find optimal weights.

<u>Description of Perceptron:</u>

*It is a type of **linear classifier**, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector [1]. This based on **hard threshold**.*

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

***w:** vector of real valued weights. Dot product of w and x and m is the number of inputs to the perceptron.*

$$\sum_{i=1}^{m} w_i x_i ,$$

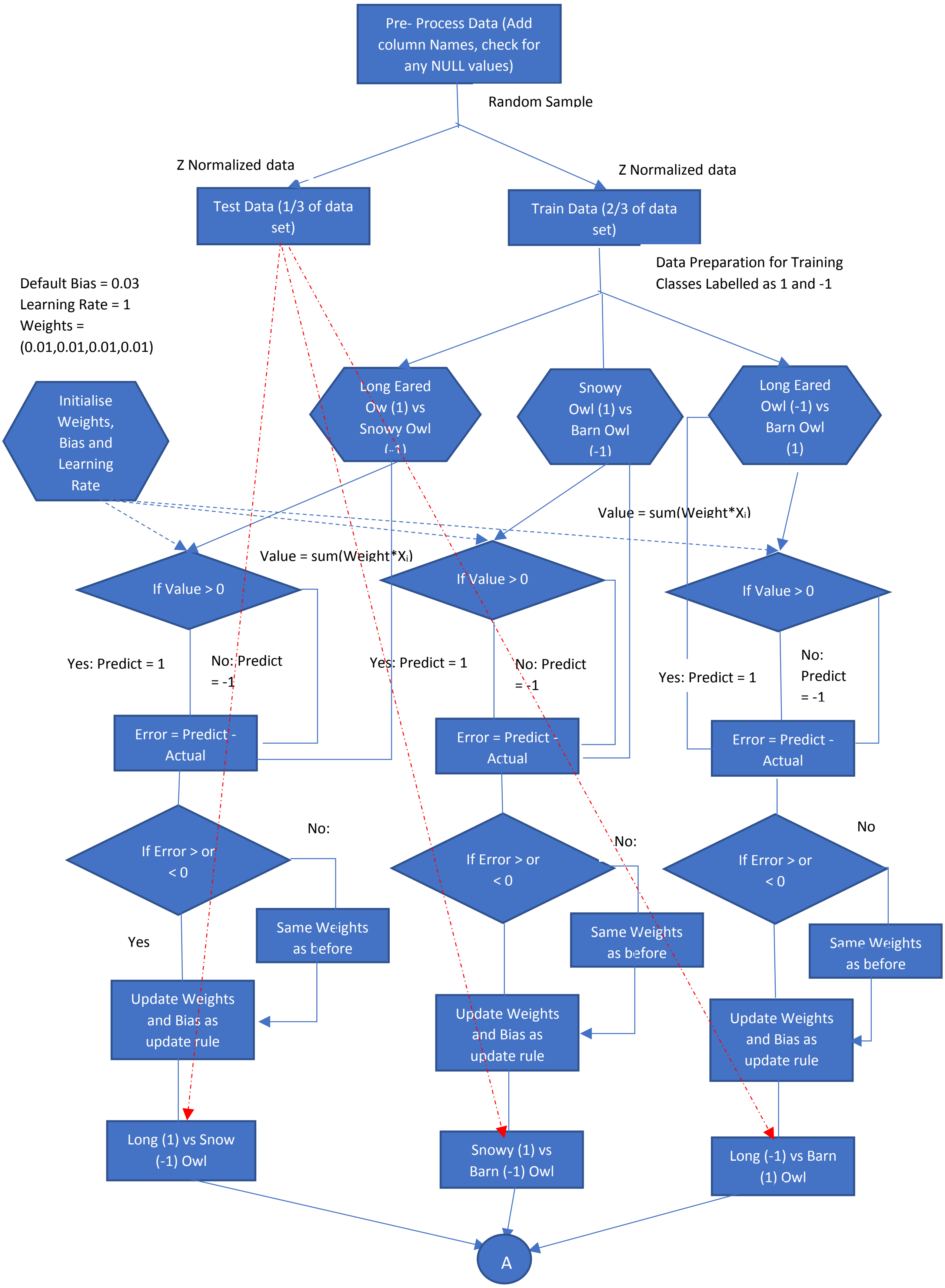
***b:** is the bias. Bias shifts the decision boundary away from the origin and does not depend on input value.*

*Given is the multiclass data sets with classes "LongEaredOwl", "SnowyOwl" and "BarnOwl'. Hence, I used One Vs One (**OVO**) approach. Where 3 classifiers used – One for LongEaredOwl vs SnowyOwl, Second for SnowyOwl vs BarnOwl and Third for LongEaredOwl vs BarnOwl. Output is predicted based on majority votes of three classifier. Advantage of one vs one is less sensitive to imbalanced data sets. However, it is computationally expensive. Assuming accuracy as important factor, I used one vs one approach.*

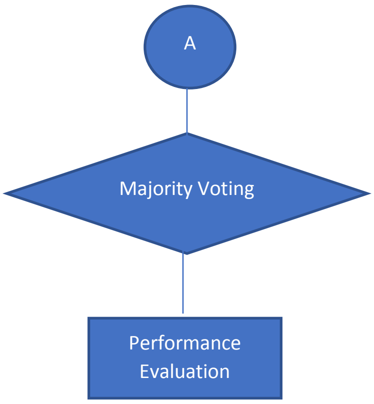
<u>Procedure: (only for one classifier)</u>

- *Initialise the weights (reasonable random weights)*
- *Initialise learning rate (usually between 0 to 1) m*
- *Repeat until converges (or condition satisfies)*
- *For each training instance (x)*
    - *Compute output f (w.x)*
    - *Note Error = output – actual (update bias and weight only if error not equal to 0)*
    - *Bias = bias + m * error*
    - *W(i) = w(i) + error * m * x(i); where, W: weight vector, X: input vector*
- *Similarly, we need to build 3 classifiers and based on max voting from 3 classifier we need to predict the class and evaluate performance of our model.*
- *Detailed implementation is given below.*

Implemented Design for Perceptron:



***Note**: Repeat whole process 10 times, each time random sample train and test data set and taking an average accuracy.*



Weight Update rule used:

```
Total_Errors <<- c(Total_Errors, Error_L)
if(Error_L != 0)
{
  #updating weights and bias if error is not zero
  Bias <<- Bias + m * Error_L
  Weights <<- Weights + (Error_L * m * as.numeric(x))
}
```

Uses Stochastic gradient descent for updating weights with learning rate 0.1.
X: is individual observation.

Voting System:

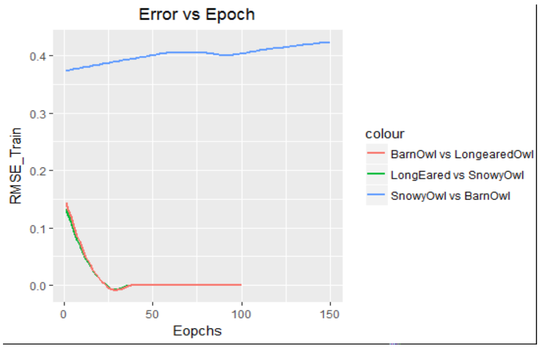
```
#voting system to predicted the multiclass
Pecp_Predicted_Class = function(Predicted1,Predicted2,Predicted3)
{

  Predicted_class = vector(mode = 'numeric', length = nrow(Test_data))

  for(i in 1:length(Test_data$Type))
  {
    if((Predicted1[i] == 1) & (Predicted3[i] == -1)){
      Predicted_class[i] = "LongEaredOwl"
    }
    if((Predicted1[i] == -1) & (Predicted2[i] == 1)){
      Predicted_class[i] = "SnowyOwl"

    }
    if((Predicted2[i] == -1) & (Predicted3[i] == 1)){
      Predicted_class[i] = "BarnOwl"

    }
  }
  return(Predicted_class)
}
```

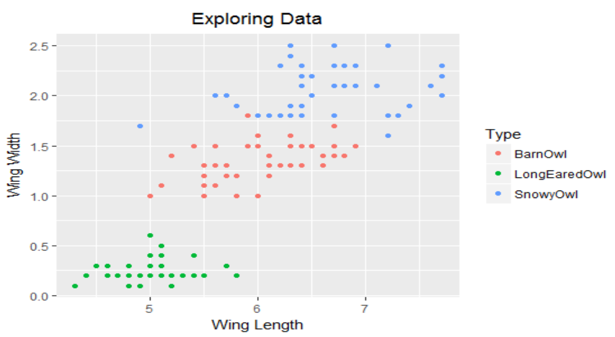
<u>Test Results:</u>

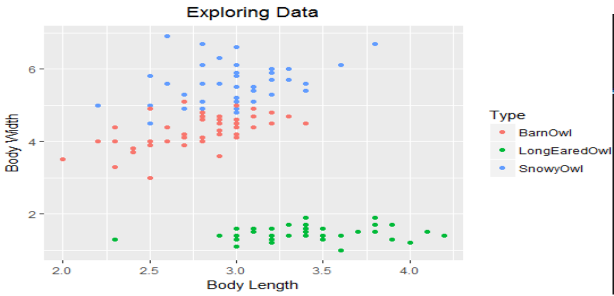
***Note**: The results shown below is 1 sample result. (However, I have run the program 10 times with randomly sampled test and train data to find the average performance).*

Output of program contains:

- ***Actual and Predicted classes***
- ***Confusion Matrix***
- ***Accuracy***
- ***Weights of each classifier used: 3 classifiers for 3 classes (One vs One)***
- ***Epochs (No of iteration to converge/minimise error)***
- ***Error vs Epoch graph***








```
$Confusion_Matrix
              Predicted
Actual         BarnOwl LongEaredOwl SnowyOwl
  BarnOwl           17            0        0
  LongEaredOwl       0           17        0
  SnowyOwl           2            0        9

$Accuracy
[1] 0.9555556
```

```
Predicted_Class
[1]  "LongEaredOwl" "LongEaredOwl" "LongEaredOwl" "LongEaredOwl" "LongEaredOwl"
[6]  "LongEaredOwl" "LongEaredOwl" "LongEaredOwl" "LongEaredOwl" "LongEaredOwl"
[11] "LongEaredOwl" "LongEaredOwl" "LongEaredOwl" "LongEaredOwl" "LongEaredOwl"
[16] "LongEaredOwl" "LongEaredOwl" "SnowyOwl"     "SnowyOwl"     "SnowyOwl"
[21] "SnowyOwl"     "SnowyOwl"     "BarnOwl"      "SnowyOwl"     "BarnOwl"
[26] "SnowyOwl"     "SnowyOwl"     "SnowyOwl"     "BarnOwl"      "BarnOwl"
[31] "BarnOwl"      "BarnOwl"      "BarnOwl"      "BarnOwl"      "BarnOwl"
[36] "BarnOwl"      "BarnOwl"      "BarnOwl"      "BarnOwl"      "BarnOwl"
[41] "BarnOwl"      "BarnOwl"      "BarnOwl"      "BarnOwl"      "BarnOwl"

Actual_Class
[1]  LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl
[7]  LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl
[13] LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl LongEaredOwl SnowyOwl
[19] SnowyOwl     SnowyOwl     SnowyOwl     SnowyOwl     SnowyOwl     SnowyOwl
[25] SnowyOwl     SnowyOwl     SnowyOwl     SnowyOwl     BarnOwl      BarnOwl
[31] BarnOwl      BarnOwl      BarnOwl      BarnOwl      BarnOwl      BarnOwl
[37] BarnOwl      BarnOwl      BarnOwl      BarnOwl      BarnOwl      BarnOwl
[43] BarnOwl      BarnOwl      BarnOwl
evels: BarnOwl LongEaredOwl SnowyOwl
```

No of Iteration	Total Test data	Accuracy	No of Miss-Classification	Cases couldn't predict any of the classes (couldn't distinguish)	Error - (1-Accuracy)
1	45	0.8666	6	0	0.1334
2	45	0.9556	2	0	0.0444
3	45	0.9556	2	0	0.0444
4	45	0.9111	4	0	0.0889
5	45	0.8888	5	0	0.1112
6	45	0.9556	2	0	0.0444
7	45	0.9556	2	0	0.0444
8	45	0.9556	2	0	0.0444
9	45	0.9333	3	0	0.0667
10	45	0.9556	2	0	0.0444

Average Accuracy: *0.9333*

Conclusion:

Most of the misclassification is observed, in distinguishing between BarnOwl and SnowyOwl. This also can be observed in Error vs Epoch graph where the blue line (SnowyOwl vs BarnOwl) not able to converge to zero. This may be due to, SnowyOwl and BarnOwl is not 100% linearly separable (this is also supported by above scatter plot – Exploring Data). If BarnOwl and SnowyOwl is fully linearly separable then we may get 100% accuracy. However, in this case there is 2-5 points which cannot separate by hyperplane hence average accuracy of 93% we *may expect on any future data points. Moreover, One vs One approach is time consuming, since data is set small only 150 observation we can use OVO. In addition, OVO suits well for imbalanced data sets.*

References:

- https://en.wikipedia.org/wiki/Perceptron
- https://www.youtube.com/watch?v=1XkjVl-j8MM

Appendix

Code Implemented:

#Name: Swaroop S Bhat

#Student Id: 17230755

#Class: 178-CT475 (MSc Data Analytics)

*#**

#Installing the below packages is necerssary is it is not installed in the system

library("ggplot2")

library("dplyr")

library("lubridate")

library("readr")

library("stringr")

library("ReporteRs")

#Note: Program runs for approximately 4 minutes due to 1000 epochs (loops to converge error)

*#** NOTE ***

#Steps to run the program

#1. set the R directory to the directory where the data set is stored. setwd("path")

#2. Run the program. (Everything is prameterized, hence no need to call any functions)

```
#Note: For convinenccce purpose.. Splitting of data to training (2/3) and testing (1/3)
#is already done. And calling the training and prediction function is already parametized
#and hence, no need to call the functions explicitly.
#**********************************************************************
*************************


options(warn = -1)
Data_set = suppressMessages(read_csv("owls15.csv", col_names = FALSE))
colnames(Data_set) = c("Body_Length", "Wing_Length", "Body_Width", "Wing_Width",
"Type")
start_time = Sys.time()#To calculate elapsed time of the program
Estimated_Accuracy  = vector(mode = 'numeric', length = 10)#to store accuracy over
repetition


for(k in 1:10)#10 random samples to estimate future accuracy
{

  Perceptron_Alg = function(Data_set)
  {

    Owl_data = Data_set

    z_norm <- function(x){((x - min(x))/(max(x) - min(x)))}#Normalising data
    Owl_data$Type <- as.factor(Owl_data$Type)
    Nrm_data <- as_data_frame(sapply(Owl_data[,-5], z_norm))
    Nrm_data$Type <- Owl_data$Type

    #Exploring Data
    print(ggplot(Owl_data)+
```

```
        geom_point(aes(x = Body_Length, y = Body_Width, colour = Type))+
        ggtitle("Exploring Data")+
        xlab("Body Length")+
        ylab("Body Width")+
        theme(plot.title = element_text(hjust = 0.5)))
  print(ggplot(Owl_data)+
        geom_point(aes(x = Wing_Length, y = Wing_Width, colour = Type))+
        ggtitle("Exploring Data")+
        xlab("Wing Length")+
        ylab("Wing Width")+
        theme(plot.title = element_text(hjust = 0.5)))

  #Testing and Training data
  set.seed(k+50)
  index = sample(1:nrow(Nrm_data), size = (nrow(Nrm_data)*2/3), prob = NULL, replace= FALSE)
  Test_data = Nrm_data[-index, ] #This is test data set. Used for validation
  Train_data = Nrm_data[index, ] #This data set is further splitted according to the classification for one vs one classification


  #Seperating training data set according to the classification for One vs one approach
  C1_data <- Train_data[Train_data$Type == "LongEaredOwl", ] #LongEaredOwl Data
  C2_data <- Train_data[Train_data$Type == "SnowyOwl", ] #SnowyOwl
  C3_data <- Train_data[Train_data$Type == "BarnOwl", ] #BarnOwl


#***************************************************************************************************
  #One Vs One classification. Hence Preparing the training data accordingly

  #************************** LongEared vs Snowvy Owl *******************************************
```

```
  Class_data1 <- rbind(C1_data, C2_data)
  Class_data1$Type <- ifelse((Class_data1$Type == "LongEaredOwl"), 1, -1)

  Train_data1 <- Class_data1[, -5]
  desired_Op_Train1 <- lapply(Class_data1[, 5], function(x){x})[[1]]

  #**************************** SnowyOwl vs BarnOwl
*****************************************
  Class_data2 <- rbind(C2_data, C3_data)
  Class_data2$Type <- ifelse((Class_data2$Type == "SnowyOwl"), 1, -1)

  Train_data2 <- Class_data2[, -5]
  desired_Op_Train2 <- lapply(Class_data2[, 5], function(x){x})[[1]]

  #******************************Barn_Owl Vs
LongEaredOwl**********************************
  Class_data3 <- rbind(C1_data, C3_data)
  Class_data3$Type <- ifelse((Class_data3$Type == "BarnOwl"), 1, -1)

  Train_data3 <- Class_data3[, -5]
  desired_Op_Train3 <- lapply(Class_data3[, 5], function(x){x})[[1]]



#************************************************************************
*****************
  #                          Training weights and bias

#************************************************************************
****************
  #Default weight and bias of perceptron algorithm
  Default_Bias = 0.03
  lr_rate = 1 # Learning rate
```

```
Initial_wt = c(0.01,0.01,0.01,0.01)



#Perceptron Training
Train_Perc = function(Train_data, b, w, lr, desired)
{
  Bias = b
  Weights = w
  m = lr#learning rate
  desired_op = desired
  Predicted_value = vector(mode = 'numeric', length = nrow(Train_data))#to store
predicted class
  Total_RMSE = vector(mode = 'numeric', length = 100)#To store Root Mean Squared Error
for each repeats

  #Learning weights to optimally seperate class
  Learning_Weights <- function(x, y)
  {

    Pred_value <- Predict_value(x)
    Func_x[y] <<- Pred_value
    Error_L <- (desired_op[y] - Pred_value)#Error = Actual - Predicted
    Total_Errors <<- c(Total_Errors, Error_L)
    if(Error_L != 0)
    {
      #Updating weights and bias if error is not zero
      Bias <<- Bias + m * Error_L
      Weights <<- Weights + (Error_L * m * as.numeric(x))
    }

  }
```

```
#Hard threshold
#Prediction based on sum of (weights*X[i]): if sum is greater that 0 predic 1 else predict -1
Predict_value = function(x)
{
  value = sum(unlist(c((Weights * as.numeric(x)), Bias)))
  Pred_value = ifelse((value > 0), 1, -1)
  return(Pred_value)


}

#Epoch which leads to the convergence of error(if linear seperable) or to find effective dicision hyperplane
s = 0
repeat
{

  Total_Errors = vector(mode = 'numeric', length = 100)
  Func_x = vector(mode = 'numeric', length = nrow(Train_data))

  for(i in 1:nrow(Train_data))
  {
    Learning_Weights(Train_data[i,], i)
  }

  s = s+1
  RMSE = sqrt(sum(Total_Errors^2)/length(desired_op))#RMSE
  Total_RMSE[s] <- RMSE
  #Condition to go out of repeat
  if((RMSE < 0.02) | (s==1000)){
    break
```

```
    }
   }
   Predicted_value <- Func_x

   #Final values of paramter selected
   Final_param = list(Bias_va = Bias,
                 Weight = Weights,
                 Predicted_value = Predicted_value,
                 Actual_value = desired_op,
                 RMSE_Epochs = Total_RMSE
   )
   return(Final_param)
  }



#**********************************************************************
***
  #Test data prediction and comparing the accuracy
  Test = function(h, W, b){
   Weights = W
   Bias = b

   #Prediction based on sum of (weights*X[i]): if sum is greater that 0 predic 1 else predict -
1
   Test_predict = function(x)
   {
    for(i in 1:nrow(x))
    {
     value = sum(unlist(c((Weights * as.numeric(x[i,])), Bias)))
     Pred_value = ifelse((value > 0), 1, -1)
     Test_predict_val[i] <<- Pred_value
```

```
    }
   }

  Test_predict_val = vector(mode = 'numeric', length = nrow(h))
  Test_predict(h)

  Test_result = list(Predicted = Test_predict_val)

  return(Test_result)

}


#Voting system to predict the class = majority voting
Pecp_Predicted_Class = function(Predicted1,Predicted2,Predicted3)
{

  Predicted_class = vector(mode = 'numeric', length = nrow(Test_data))

  for(i in 1:length(Test_data$Type))
  {
    if((Predicted1[i] == 1) & (Predicted3[i] == -1)){
      Predicted_class[i] = "LongEaredOwl"
    }
    if((Predicted1[i] == -1) & (Predicted2[i] == 1)){
      Predicted_class[i] = "SnowyOwl"

    }
    if((Predicted2[i] == -1) & (Predicted3[i] == 1)){
      Predicted_class[i] = "BarnOwl"
```

```
      }
     }
     return(Predicted_class)
   }


#*************************************************************************
********
   #Training the algorithm based on 2/3 of data: Splitting of data has been done at the
begining
   t = Train_Perc(Train_data1, Default_Bias, Initial_wt, lr_rate, desired_Op_Train1)
   e = Train_Perc(Train_data2, Default_Bias, Initial_wt, lr_rate, desired_Op_Train2)
   v = Train_Perc(Train_data3, Default_Bias, Initial_wt, lr_rate, desired_Op_Train3)



   #Testing the algorithm based on testing data 1/3 (Unseen data) : Splitting of data has
been done at the begining
   t1 = Test(Test_data[,-5], t$Weight, t$Bias_va)
   t2 = Test(Test_data[,-5], e$Weight, e$Bias_va)
   t3 = Test(Test_data[,-5], v$Weight, v$Bias_va)

   #ggplot to see the error convergence if points are linearly seperable
   Learning = function(a, b, c)
   {
     print(ggplot()+
          geom_smooth(aes(x = c(1:150), y = a[1:150], colour = "LongEared vs SnowyOwl"), se
= F)+
          geom_smooth(aes(x = c(1:150), y = b[1:150], colour = "SnowyOwl vs BarnOwl"),
se=F)+
          geom_smooth(aes(x = c(1:150), y = c[1:150], colour = "BarnOwl vs LongearedOwl"),
se=F)+
          ggtitle("Error vs Epoch")+
          xlab("Eopchs")+
```

```
        ylab("RMSE_Train")+
        theme(plot.title = element_text(hjust = 0.5)))
  }

  (Learning(t$RMSE_Epochs, e$RMSE_Epochs, v$RMSE_Epochs))

  #Confusion matrix to find the missclassification
  confusion_matrix = table(Actual = Test_data$Type, Predicted =
Pecp_Predicted_Class(t1$Predicted,t2$Predicted,t3$Predicted))




  #To display the final parameters (Epochs and weights), Accuracy and confusion matrix on
screen
  op_list = list(
    Predicted_Class = Pecp_Predicted_Class(t1$Predicted,t2$Predicted,t3$Predicted),
    Actual_Class = Test_data$Type,
    Confusion_Matrix = confusion_matrix,
    Accuracy =
((confusion_matrix["BarnOwl","BarnOwl"]+confusion_matrix["LongEaredOwl","LongEaredO
wl"]+confusion_matrix["SnowyOwl","SnowyOwl"])/length(Test_data$Type)),
    Classfier1_Weight = c(unlist(t$Weight), Bias = t$Bias_va),
    Classfier2_Weight = c(unlist(e$Weight), Bias = e$Bias_va),
    Classfier3_Weight = c(unlist(v$Weight), Bias = v$Bias_va),
    Epochs_class1 = length(t$RMSE_Epochs),
    Epochs_class2 = length(e$RMSE_Epochs),
    Epochs_class2 = length(v$RMSE_Epochs)
  )
  print(op_list)
  Estimated_Accuracy[k] <<- unlist(op_list[4])

 }
```

```r
  Perceptron_Alg(Data_set) #calling the function to train and test
}

#Mean Accuracy
(Estimated_Accuracy)
cat("Likely Expected Accuracy (mean) on prediction is:", mean(Estimated_Accuracy))
cat("Time Elapsed: ",(Sys.time() - start_time))
```