Assignment#3-- Perceptron for Multiclass Classification

Overview:

The goal of this assignment is to select, implement and evaluate a machine learning algorithm. Perceptron algorithm is selected to classify multiclass data of Owl types. The tools used for implementing algorithm is R studio, and the language used is R programming language.

Description about Perceptron:

Perceptron or single-layer neural network is the basic neural network. It receives multi-dimensional input which is processed using an activation function. The algorithm is trained using labeled data and learning algorithm then adjusts the weight in the process if there are any wrong predictions. It is a self-learning algorithm which uses back-propagation error method. In the self-learning phase, the difference between the predicted value and actual value is calculated. Based on this difference, the error is estimated. The error is back-propagated to all the units to keep the error at each unit proportional to the contribution of that unit towards total error of the process. This back-propagated error at each unit is used to optimize the weight at each connection.

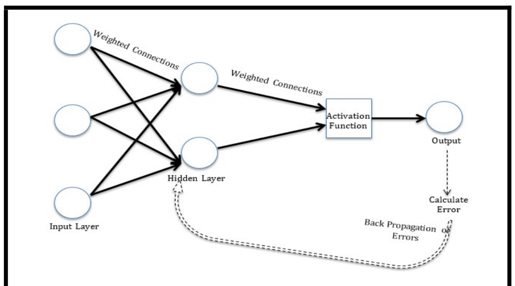


Figure 1 A simple neural network model

Design Decision:

Since one vs one approach is used, the data is divided into 3 groups with each group containing two species, and the type is labelled accordingly.

```
shuffle_index <- createDataPartition(norm_data$Type, p = .66, list = FALSE)
TrainData <- norm_data[shuffle_index,]
TestData <- norm_data[-shuffle_index,]

Owl_type1_train <- TrainData[TrainData$Type == "LongEaredOwl",]
Owl_type2_train <- TrainData[TrainData$Type == "SnowyOwl",]
Owl_type3_train <- TrainData[TrainData$Type == "BarnOwl",]

###########Creating Binary Groups

####Pair_1##LongEared vs Snowvy Owl

Train_Grp_1 <- rbind(Owl_type1_train,Owl_type2_train)
Train_Grp_1$Type <- ifelse((Train_Grp_1$Type == "LongEaredOwl"), 1, -1)
```

Euclidean norm or L2 norm is being calculated in the algorithm. It calculates the distance of the vector from the origin of the vector space. The L2 norm is further used to calculate the maximum norm of the vector which is used in the regularization of the neural network weights.

```
euclidean_dist <- function(x){
  sqrt(sum(x*x))
}
```

```r
euc_dist_max <- max(apply(x,1,euclidean_dist))
```

```r
s <- euclidean_dist(w)
return(list(w= w/s, b = b/s, error = k, iteration = count))
}
```

The processing done at each neuron unit is denoted by:

$$output = sum(weights * inputs) + bias$$

which is denoted in the code as:

```r
dist_plane <- function(z,w,b){
  sum(z*w) + b
}
```

The weight(w) vector are the numerical parameters which shares the magnitude of impact each neuron has on another neuron. The input(z) vector is the vector of attributes on which the output depends which is the matrix multiplication to get the weighted sum.Bias(b) is analogous to the constant (c) which is added in the linear equation (y = m*x + c). It is used to adjust the output of the weighted sum of the input.Activation function is the function to be applied on the output obtained from the neuron of the previous layer. The activation function applied in my algorithm is 'heaviside step function' which will label the output depending if the output is less than or greater than threshold value.

$$\text{output} = \begin{cases} -1 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

```r
activte_fun <- function(x,w,b){
  distances <- apply(x, 1, dist_plane,w,b)
  return(ifelse(distances < 0, -1, +1))
}
```

The feature of perceptron is that it modifies the weight and bias initially provided as per the prediction obtained. If the prediction is wrong, then the weight and bias are updated with learning rate times. The learning rate helps in converging the model and deciding the appropriate weight and bias. The weight and bias keep on modifying until predicted 'y' is as close as possible to the actual 'y' value. The performance of the system depends solely on the distance between the actual 'y' and the predicted 'y'.

Now adjust score based on error:

$f(x)= sign(sum\ of\ weights*inputs)$, the errors are possible

if $y=+1$ and $f(x)=-1$, $w*x$ is too small, make it bigger

if $y=-1$ and $f(x)=+1$, $w*x$ is too large make it smaller

Apply the following rules:

make $w=w-x$ if $f(f)=+1$ and $y=-1$

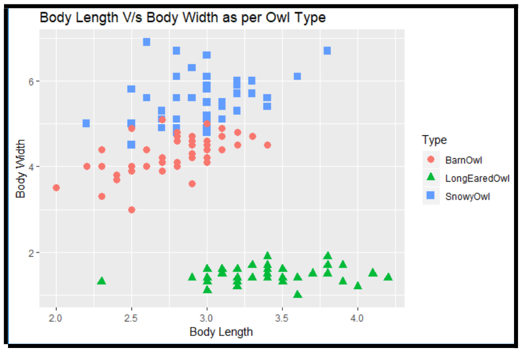
make $w=w+x$ if $f(f)=-1$ and $y=+1$

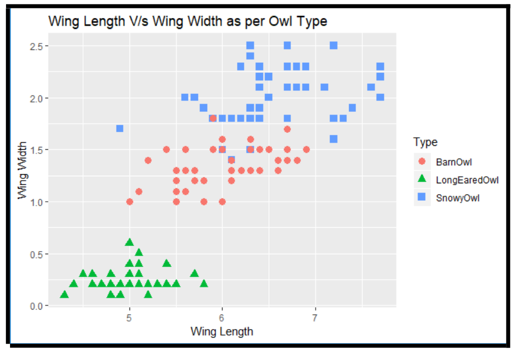
$w=w$ if $f(x)=y$

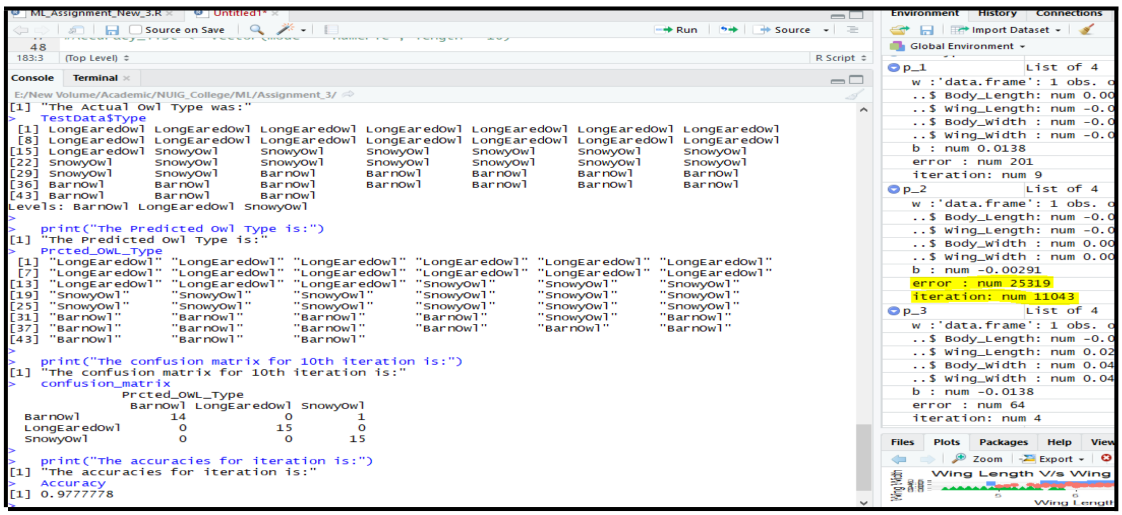
Or simply, $w=w+yx$ if $f(x)!=y$

Result:

The results are for one of the samples obtained by taking a random seed value.

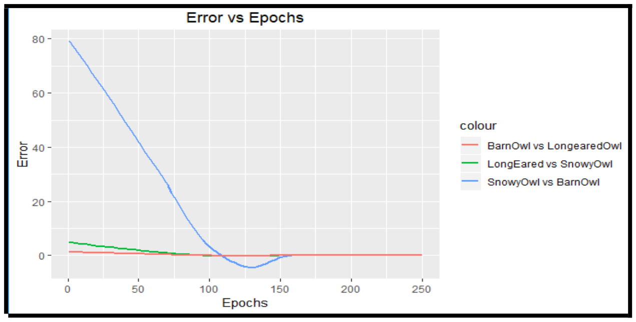






seed value	Accuracy	Missclassification Num	Prediction failure	Number of iterations for weight and bias prediction in group of SnowyOwl and BarnOwl (p2)	Number of Errors in prediction
360	0.9333333	2	1	759	3242
525	0.9555556	2	0	442	3119
688	0.9555556	2	0	411	2208
693	0.9555556	2	0	232	1758
1386	0.9555556	2	0	624	2637
694	0.9111111	4	0	640	3341
999	0.9111111	4	0	743	3734
1050	0.9111111	4	0	772	2884
650	0.9777778	1	0	11043	25319
2121	0.9111111	4	0	162	1985
Mean Accuracy	0.937472665				

```
[1] "The confusion matrix for 10th iteration is:"
> confusion_matrix
              Prcted_OWL_Type
               BarnOwl LongEaredOwl SnowyOwl
  BarnOwl           14            0        1
  LongEaredOwl       0           15        0
  SnowyOwl           0            0       15
```



Conclusion:

We can observe from ggplots that “LongEaredOwl” is linearly separable in both Body Length and Width and Wing Length and Width attributes as compared to "SnowyOwl" and "BarnOwl" which are close enough to mix at certain points. This complexity in data points in "SnowyOwl" and "BarnOwl" created problem in prediction of Owl Type in Test Data. As per the screen-shots above, the prediction of weight and bias in the data group of "SnowyOwl" and "BarnOwl" (p2) took maximum time, with the number of iterations reaching 11,043 and error count reaching 25,319 for the seed value 650 with 1 misclassification. The iterations/Epochs were not fixed to 1,000 to test the efficiency of the code in the test phase. The Epochs were later fixed to 1,000, and the accuracy for seed 650 moved down to 0.9555556 from 0.9777778 which is acceptable. If "SnowyOwl" and "BarnOwl” would be linearly separable, then the prediction would have been 100%. Therefore, the prediction of around 94% was achieved overall. We can conclude that the One vs One Perceptron approach works the best with linearly separable data, and it is computationally very expensive and very time consuming. The convergence is one of the biggest problems of the perceptron. It can be proved from the predictions that the perceptron learning rule converges if the two classes can be separated by linear hyperplane, but problems arise if the classes cannot be separated perfectly by a linear classifier as in the case of Train_Grp_2 of SnowyOwl vs BarnOwl…

Reference:

https://www.analyticsvidhya.com/blog/2017/09/creating-visualizing-neural-network-in-r/
https://datascience.stackexchange.com/questions/410/choosing-a-learning-rate
https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9
https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html
https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975
Book: Neural Network with R by Giuseppe and Balaji

Appendix:

```r
#########Downloading important packages

#install.packages("ggplot2")
library("ggplot2")
#install.packages("caret")
library("caret")

########Upload Raw Dataset and name the columns

#getwd()

setwd("E:/New Volume/Academic/NUIG_College/ML/Assignment_3/")

Data_Set <-  read.csv("owls.csv", header = FALSE)

colnames(Data_Set) <- c("Body_Length","Wing_Length", "Body_Width", "Wing_Width", "Type")


Classification_Algo <- function(Data_Set, no_of_iterations){

  raw_data <- Data_Set
  cycles <- no_of_iterations

  #########Normalizing Data

  normzng_functn <- function(x){
    ((x - min(x))/(max(x) - min(x)))
  }

  raw_data$Type <- as.factor(raw_data$Type)
  norm_data <- as.data.frame(apply(raw_data[,-5], 2, normzng_functn))
  norm_data$Type <- raw_data$Type
```

```r
#summary(norm_data)

##########Initial Data Exploration

ggplot(raw_data, aes(x = Body_Length, y = Body_Width)) +
  geom_point(aes(colour=Type, shape=Type), size = 3) +
  xlab("Body Length") +
  ylab("Body Width") +
  ggtitle("Body Length V/S Body Width as per Owl Type")

ggplot(raw_data, aes(x = Wing_Length, y = Wing_Width)) +
  geom_point(aes(colour=Type, shape=Type), size = 3) +
  xlab("Wing Length") +
  ylab("Wing Width") +
  ggtitle("Wing Length V/S Wing Width as per Owl Type")


###########Preparing Training and Test Data

Accuracy_list <- vector(mode = 'numeric', length = cycles)

for (k in 1:cycles){
  set.seed(649 + k)
  shuffle_index <- createDataPartition(norm_data$Type, p = .66, list = FALSE)
  TrainData <- norm_data[shuffle_index,]
  TestData <- norm_data[-shuffle_index,]

  Owl_type1_train <- TrainData[TrainData$Type == "LongEaredOwl",]
  Owl_type2_train <- TrainData[TrainData$Type == "SnowyOwl",]
  Owl_type3_train <- TrainData[TrainData$Type == "BarnOwl",]

  ###########Creating Binary Groups

  ####Pair_1##LongEared vs Snowy Owl

  Train_Grp_1 <- rbind(Owl_type1_train,Owl_type2_train)
  Train_Grp_1$Type <- ifelse((Train_Grp_1$Type == "LongEaredOwl"), 1, -1)
```

```r
  Train_Grp_1_data <- Train_Grp_1[,(1:4)]
  Train_Grp_1_exptd <-Train_Grp_1[, 5]

  ####Pair_2##SnowyOwl vs BarnOwl

  Train_Grp_2 <- rbind(Owl_type2_train,Owl_type3_train)
  Train_Grp_2$Type <- ifelse((Train_Grp_2$Type == "SnowyOwl"), 1, -1)

  Train_Grp_2_data <- Train_Grp_2[,(1:4)]
  Train_Grp_2_exptd <-Train_Grp_2[, 5]

  ####Pair_3##Barn_Owl vs LongEaredOwl

  Train_Grp_3 <- rbind(Owl_type3_train,Owl_type1_train)
  Train_Grp_3$Type <- ifelse((Train_Grp_3$Type == "BarnOwl"), 1, -1)

  Train_Grp_3_data <- Train_Grp_3[,(1:4)]
  Train_Grp_3_exptd <-Train_Grp_3[, 5]

  ###########Creating Perception Train and Test Algorithm

  euclidean_dist <- function(x){
    sqrt(sum(x*x))
  }

  dist_plane <- function(z,w,b){
    sum(z*w) + b
  }

  activte_fun <- function(x,w,b){
    distances <- apply(x, 1, dist_plane,w,b)
    return(ifelse(distances < 0, -1, +1))
  }
```

```r
perceptron <- function(x,y,learn_rate = 1){
  w <- rep(0, length = ncol(x))  #Initial weight
  b <- 0 #Initialize bias
  count <- 0 #track the run count
  err <- rep(0,1000) #count update of error
  Ecd_dist <- max(apply(x,1,euclidean_dist))
  flag <- TRUE

  while(flag){
    flag <- FALSE
    yc = activte_fun(x,w,b)
    for (i in 1:nrow(x)){
      if (y[i] != yc[i]){
        w <- w + learn_rate * y[i] * x[i,]
        b <- b + learn_rate * y[i] * (Ecd_dist)^2
        err[i] <- err[i] + 1
        flag <- TRUE
      }
    }
    count = count + 1
    if(count > 1000)
      break
  }
  s <- euclidean_dist(w)
  return(list(w= w/s, b = b/s, error = err, iteration = count))
}

perceptron_Test <- function(x,w,b){

  yc = activte_fun(x,w,b)

  return(list(predtcd_value = yc))
}

p_1 = perceptron(Train_Grp_1_data,Train_Grp_1_exptd)
p_2 = perceptron(Train_Grp_2_data,Train_Grp_2_exptd)
p_3 = perceptron(Train_Grp_3_data,Train_Grp_3_exptd)
```

```r
p_test_1 <- perceptron_Test(TestData[,(1:4)],p_1$w,p_1$b)
p_test_2 <- perceptron_Test(TestData[,(1:4)],p_2$w,p_2$b)
p_test_3 <- perceptron_Test(TestData[,(1:4)],p_3$w,p_3$b)

Predctd_owl_type <- function(Test_model_Grp_1,Test_model_Grp_2,Test_model_Grp_3){

  Predcted_Type <- vector(mode = "numeric", length = nrow(TestData))

  for(i in 1:length(TestData$Type)){
    if((p_test_1$predtcd_value[i] == 1) & (p_test_3$predtcd_value[i] == -1)){
      Predcted_Type[i] = "LongEaredOwl"
    }

    if((p_test_2$predtcd_value[i] == 1) & (p_test_1$predtcd_value[i] == -1)){
      Predcted_Type[i] = "SnowyOwl"
    }

    if((p_test_3$predtcd_value[i] == 1) & (p_test_2$predtcd_value[i]== -1)){
      Predcted_Type[i] = "BarnOwl"
    }
  }
  return(Predcted_Type)
}

Prcted_OWL_Type <- Predctd_owl_type(p_test_1,p_test_2,p_test_3)

confusion_matrix <- table(TestData$Type,Prcted_OWL_Type)

Accuracy <- ((confusion_matrix["BarnOwl","BarnOwl"]+
                confusion_matrix["LongEaredOwl","LongEaredOwl"]+
                confusion_matrix["SnowyOwl","SnowyOwl"])/length(TestData$Type))

Accuracy_list[k] <- Accuracy
```

```r
    Output <- list(
      Actual_Owl_Type = TestData$Type,
      Predicted_Owl_Type = Prcted_OWL_Type,
      Confusion_Matrix = confusion_matrix,
      Current_itr_Accuracy = Accuracy,
      List_Of_Accuracies = Accuracy_list,
      Perceptron_1_weights = p_1$w,
      Perceptron_1_bias = p_1$b,
      Perceptron_2_weights = p_2$w,
      Perceptron_2_bias = p_2$b,
      Perceptron_3_weights = p_3$w,
      Perceptron_3_bias = p_3$b,
      Perceptron_1_Epochs = p_1$iteration,
      Perceptron_2_Epochs = p_2$iteration,
      Perceptron_3_Epochs = p_3$iteration
    )

    print(Output)

    print(ggplot()+
            geom_smooth(aes(x = c(1:250), y = p_1$err[1:250], colour = "LongEared vs SnowyOwl"), se = F)+
            geom_smooth(aes(x = c(1:250), y = p_2$err[1:250], colour = "SnowyOwl vs BarnOwl"), se=F)+
            geom_smooth(aes(x = c(1:250), y = p_3$err[1:250], colour = "BarnOwl vs LongearedOwl"), se=F)+
            ggtitle("Error vs Epochs")+
            xlab("Epochs")+
            ylab("Error")+
            theme(plot.title = element_text(hjust = 0.5)))

    scatter3D(TestData$Body_Length, TestData$Body_Width, TestData$Wing_Length, phi = 0, bty = "g",
              pch = 20, cex = 2, ticktype = "detailed")

  }

  cat("Mean Accuracy is:", mean(Accuracy_list))

}

Classification_Algo(Data_Set,1)
```