**Multiclass Classification Using Perceptron**

***Overview:***

*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.*

*Tools/Coding language used: R Studio, R*

*Algorithm Selected: Perceptron (One vs One approach) with SGD to find optimal weights.*

***Description of Perceptron:***

*It is a type of*[***linear classifier***](https://en.wikipedia.org/wiki/Linear_classifier)*, i.e. a classification algorithm that makes its predictions based on a*[*linear predictor function*](https://en.wikipedia.org/wiki/Linear_predictor_function)*combining a set of weights with the*[*feature vector*](https://en.wikipedia.org/wiki/Feature_vector) *[1]. This based on* ***hard threshold****.*

**

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

**

***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.*

*Procedure: (only for one classifier)*

* *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:***

Pre- Process Data (Add column Names, check for any NULL values)

A

No:

No:

No:

Same Weights as before

Same Weights as before

If Error > or < 0

If Error > or < 0

Same Weights as before

Yes:

If Error > or < 0

Snowy Owl (1) vs Barn Owl (-1)

Yes: Predict = 1

No: Predict = -1

No: Predict = -1

Yes: Predict = 1

No: Predict = -1

Yes: Predict = 1

Error = Predict - Actual

Error = Predict - Actual

Error = Predict - Actual

Value = sum(Weight\*Xi)

If Value > 0

If Value > 0

If Value > 0

Value = sum(Weight\*Xi)

Initialise Weights, Bias and Learning Rate

Test Data (1/3 of data set)

Data Preparation for Training Classes Labelled as 1 and -1

Long Eared Owl (-1) vs Barn Owl (1)

Long Eared Ow (1) vs Snowy Owl (-1)

Z Normalized data

Z Normalized data

Random Sample

Train Data (2/3 of data set)

Long (1) vs Snow (-1) Owl

]

Snowy (1) vs Barn (-1) Owl

]

Long (-1) vs Barn (1) Owl

Default Bias = 0.03

Learning Rate = 1

Weights = (0.01,0.01,0.01,0.01)

Update Weights and Bias as update rule

Update Weights and Bias as update rule

Update Weights and Bias as update rule

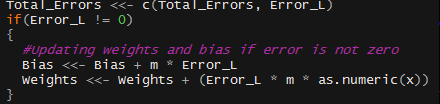
***Note****: Repeat whole process 10 times, each time random sample train and test data set and taking an average accuracy****.***

A

Majority Voting

Performance Evaluation

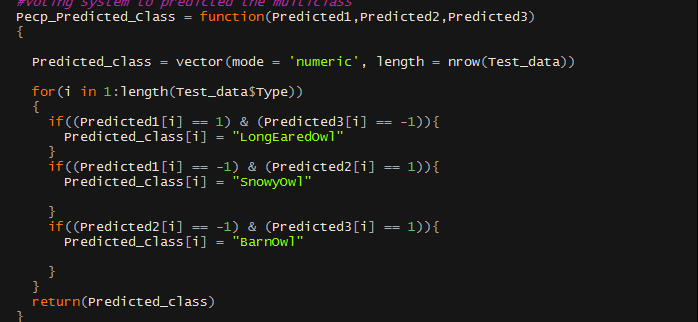
***Weight Update rule used:***



*Uses Stochastic gradient descent for updating weights with learning rate 0.1.*

*X: is individual observation.*

***Voting System:***

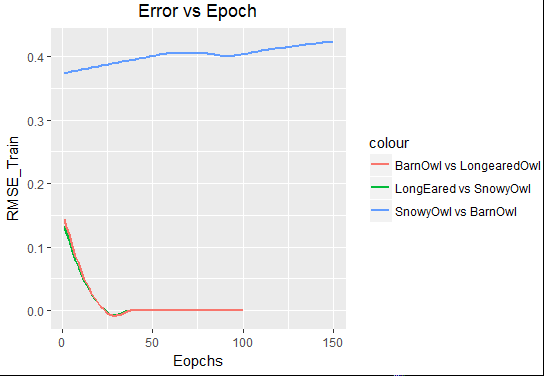
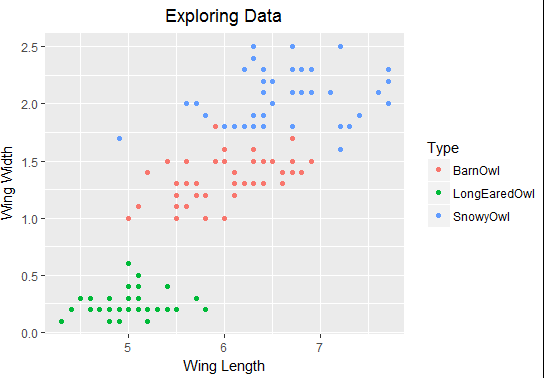


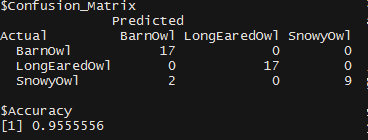
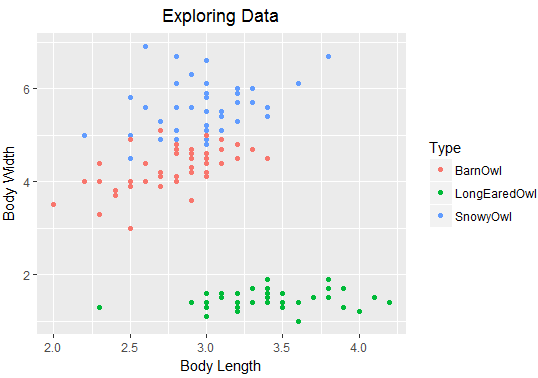
***Test Results:***

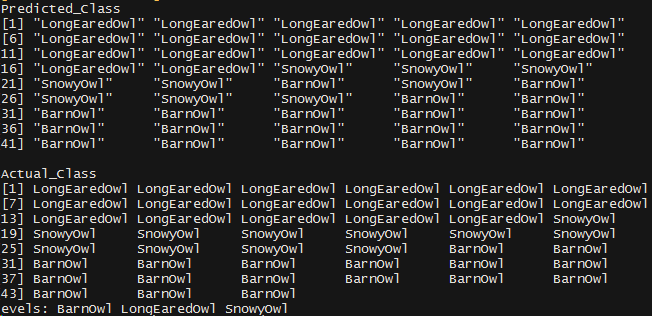
***Note****: The results shown below is 1 sample result. (However, I have run the program 10 times with randomly sampled test and train data to find the average performance).*

*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***





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 convinencce 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","LongEaredOwl"]+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])*

*}*

*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))*