Homework 2

library(pacman)  
library(gam); data("kyphosis")

## Loading required package: splines

## Loading required package: foreach

## Loaded gam 1.20

require(caTools)

## Loading required package: caTools

library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.1-2

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(psych)   
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5 v purrr 0.3.4  
## v tibble 3.1.4 v stringr 1.4.0  
## v tidyr 1.1.3 v forcats 0.5.1  
## v readr 2.0.1

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x ggplot2::%+%() masks psych::%+%()  
## x purrr::accumulate() masks foreach::accumulate()  
## x ggplot2::alpha() masks psych::alpha()  
## x tidyr::expand() masks Matrix::expand()  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## x tidyr::pack() masks Matrix::pack()  
## x tidyr::unpack() masks Matrix::unpack()  
## x purrr::when() masks foreach::when()

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

library(pls)

##   
## Attaching package: 'pls'

## The following object is masked from 'package:caret':  
##   
## R2

## The following object is masked from 'package:stats':  
##   
## loadings

library(e1071)  
library(MASS)

##   
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':  
##   
## select

library(klaR)  
library(nnet)  
library(sigmoid)

##   
## Attaching package: 'sigmoid'

## The following object is masked from 'package:e1071':  
##   
## sigmoid

## The following objects are masked from 'package:psych':  
##   
## logistic, logit

library(nnet)  
library(fastDummies)  
head(kyphosis)

## Kyphosis Age Number Start  
## 1 absent 71 3 5  
## 2 absent 158 3 14  
## 3 present 128 4 5  
## 4 absent 2 5 1  
## 5 absent 1 4 15  
## 6 absent 1 2 16

Q1: Based on the formulas we derived in class, derive the decision boundary between the two classes given the distribution parameters below.

A whiteboard with writing on it

Description automatically generated with medium confidence

LDA.png

#### Q2: Use data set kyphosis from R package “gam”, randomly divide the data into training (75%) and testing (25%) set.

#Splitting Data into Training and Testing Data  
set.seed(11)  
training.samples <- kyphosis$Kyphosis %>%  
 createDataPartition(p = 0.75, list = FALSE)  
train.data <- kyphosis[training.samples, ]  
train.data = data.frame(train.data)  
test.data <- kyphosis[-training.samples, ]  
test.data = data.frame(test.data)  
  
  
#Proving the proper split  
SizeOriginal <- nrow(kyphosis)  
SizeTraining <- nrow(train.data)  
SizeTest <- nrow(test.data)  
PercentageTraining <- (SizeTraining/SizeOriginal)\*100  
PercentageTest <- (SizeTest/SizeOriginal)\*100  
  
sprintf('Percent of Testing Data = %f', PercentageTest)

## [1] "Percent of Testing Data = 24.691358"

sprintf('Percent of Training Data = %f', PercentageTraining)

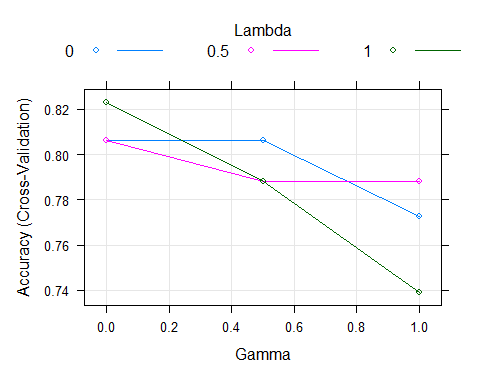
## [1] "Percent of Training Data = 75.308642"

#### 2.1 Fit a regularized discriminant analysis model, find the best tuning parameter combination, and evaluate the model using the testing set and report the confusion matrix

set.seed(11)  
model.rda <- rda(Kyphosis~.,data=train.data)  
grid = trainControl(method = "cv", number = 5)  
fit\_rda\_grid = train(Kyphosis ~ ., data = train.data, method = "rda", trControl = grid)  
  
fit\_rda\_grid

## Regularized Discriminant Analysis   
##   
## 61 samples  
## 3 predictor  
## 2 classes: 'absent', 'present'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 48, 48, 49, 50, 49   
## Resampling results across tuning parameters:  
##   
## gamma lambda Accuracy Kappa   
## 0.0 0.0 0.8061772 0.22056650  
## 0.0 0.5 0.8061772 0.27339669  
## 0.0 1.0 0.8228438 0.34268241  
## 0.5 0.0 0.8061772 0.17413793  
## 0.5 0.5 0.7879953 0.00000000  
## 0.5 1.0 0.7879953 0.00000000  
## 1.0 0.0 0.7728438 0.12136015  
## 1.0 0.5 0.7879953 0.00000000  
## 1.0 1.0 0.7392774 -0.07608696  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were gamma = 0 and lambda = 1.

plot(fit\_rda\_grid)



predictions.rda = predict(model.rda, test.data)  
confusionM.rda<-confusionMatrix(predictions.rda$class,test.data$Kyphosis)  
print(confusionM.rda$table)

## Reference  
## Prediction absent present  
## absent 16 4  
## present 0 0

print(confusionM.rda)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction absent present  
## absent 16 4  
## present 0 0  
##   
## Accuracy : 0.8   
## 95% CI : (0.5634, 0.9427)  
## No Information Rate : 0.8   
## P-Value [Acc > NIR] : 0.6296   
##   
## Kappa : 0   
##   
## Mcnemar's Test P-Value : 0.1336   
##   
## Sensitivity : 1.0   
## Specificity : 0.0   
## Pos Pred Value : 0.8   
## Neg Pred Value : NaN   
## Prevalence : 0.8   
## Detection Rate : 0.8   
## Detection Prevalence : 1.0   
## Balanced Accuracy : 0.5   
##   
## 'Positive' Class : absent   
##

#Best Tuning Parameters are Gamma and Lambda

#### 2.2 Fit a logistic regression model, and interpret the coefficients you estimated

# Fit the model  
model.lReg <- nnet::multinom(Kyphosis ~., data = train.data)

## # weights: 5 (4 variable)  
## initial value 42.281978   
## iter 10 value 25.720280  
## final value 25.720273   
## converged

predicted.lReg <- model.lReg %>% predict(test.data)  
accuracy.lReg <- mean(predicted.lReg == test.data$Kyphosis)  
sprintf('Model Accuracy = %f', accuracy.lReg)

## [1] "Model Accuracy = 0.850000"

confusionM.lReg<-confusionMatrix(predicted.lReg, test.data$Kyphosis)  
print(confusionM.lReg)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction absent present  
## absent 16 3  
## present 0 1  
##   
## Accuracy : 0.85   
## 95% CI : (0.6211, 0.9679)  
## No Information Rate : 0.8   
## P-Value [Acc > NIR] : 0.4114   
##   
## Kappa : 0.3478   
##   
## Mcnemar's Test P-Value : 0.2482   
##   
## Sensitivity : 1.0000   
## Specificity : 0.2500   
## Pos Pred Value : 0.8421   
## Neg Pred Value : 1.0000   
## Prevalence : 0.8000   
## Detection Rate : 0.8000   
## Detection Prevalence : 0.9500   
## Balanced Accuracy : 0.6250   
##   
## 'Positive' Class : absent   
##

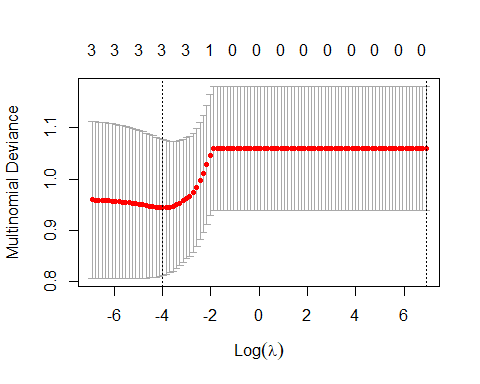
summary(model.lReg)

## Call:  
## nnet::multinom(formula = Kyphosis ~ ., data = train.data)  
##   
## Coefficients:  
## Values Std. Err.  
## (Intercept) -2.802845551 1.651499825  
## Age 0.008487559 0.006541321  
## Number 0.452436071 0.241935524  
## Start -0.116142793 0.077015622  
##   
## Residual Deviance: 51.44055   
## AIC: 59.44055

# Based off the results,   
# Age has little effect to if kyphosis is found in the child,   
# Start(which vertebrae the surgeon started on) has a negative correlation,   
# Number of operations has a positive correlation to finding the kyphosis.

#### 2.3 Fit a regularized logistic regression model, and find the best model from your regularization

x <- train.data[,2:4] %>% as.matrix()  
y <- train.data[,1] %>% as.matrix()  
  
y\_dummy <- dummy\_columns(y)  
colnames(y\_dummy)[colnames(y\_dummy) == "V\_absent"] <- "absent"  
colnames(y\_dummy)[colnames(y\_dummy) == "V\_present"] <- "present"  
  
  
y\_dummy1 <- y\_dummy[,2:3] %>% as.matrix()  
  
# Perform 10-fold cross-validation to select lambda ---------------------------  
lambdas\_to\_try <- 10^seq(-3, 3, length.out = 100)  
  
# Setting alpha = 1 implements lasso regression  
lasso\_cv <- cv.glmnet(x, y\_dummy1, family="multinomial", alpha = 1, lambda = lambdas\_to\_try,  
 standardize = TRUE, nfolds = 10)  
plot(lasso\_cv)



lasso\_model<-glmnet(x, y\_dummy1, family="multinomial", alpha = 1, lambda=lasso\_cv$lambda.min, standardize = TRUE)  
test\_class\_hat<-as.factor(predict(lasso\_model, newx = as.matrix(test.data[,2:4]),type="class"))

#### 3.1 Use mvrnorm function to generate the training data set below (use seed(100)):

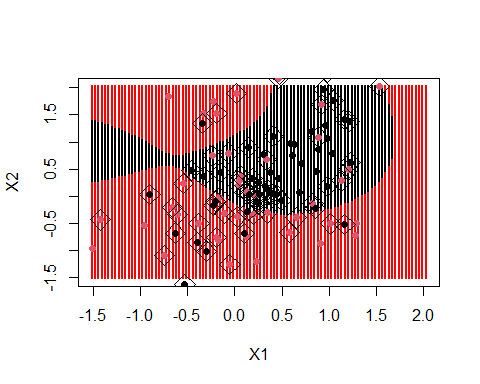
# Training Data  
set.seed(100)  
  
cmatrix = cbind(c(1, 0.8),c(0.8, 1.5))  
mu = c(0,0)  
mu = as.matrix(mu)  
mu = t(mu)  
  
cmatrix2 = cbind(c(0.5, 0.4),c(0.4, 1))  
  
mu2 = c(0.5,0.5)  
mu2 = as.matrix(mu2)  
mu2 = t(mu2)  
nrows = 50  
  
x1 = data.frame(mvrnorm(n = 50, mu = mu , Sigma = cmatrix))  
x2 = data.frame(mvrnorm(n = 50, mu = mu2, Sigma = cmatrix2))  
x\_train = rbind(x1,x2)  
  
Y\_train = rep(c(1,-1), c(50,50))  
  
Y = as.factor(Y\_train)  
dat = cbind(x\_train, Y)  
dat = data.frame(dat)

#### 3.2 Fit a nonlinear SVM model (using radial kernel) based on the training data generated in 3.1, and make a plot to visualize the decision boundary (set cost=10).

svmfit = svm(Y ~ ., data = dat, kernel = "radial", cost = 10, scale = FALSE, type = 'C-classification')  
print(svmfit)

##   
## Call:  
## svm(formula = Y ~ ., data = dat, kernel = "radial", cost = 10, type = "C-classification",   
## scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10   
##   
## Number of Support Vectors: 68

dat.svm<-predict(svmfit,data=dat)  
  
  
make.grid = function(x\_train, n = 100) {  
 grange = apply(x\_train, 2, range)  
 x1 = seq(from = grange[1,1], to = grange[2,1], length = n)  
 x2 = seq(from = grange[2,1], to = grange[1,1], length = n)  
 expand.grid(X1 = x1, X2 = x2)  
}  
xgrid = make.grid(x\_train)  
  
ygrid = predict(svmfit, xgrid)  
plot(xgrid, col = c("black","red")[as.numeric(ygrid)], pch = 20, cex = .2)  
points(x\_train, col = Y , pch = 19)  
points(x\_train[svmfit$index,], pch = 5, cex = 2)

 #### 3.3 Use the same approach in 3.1 to generate the testing data set with 100 samples (use seed(123)).a) Evaluate the classification performance using the classification accuracy based on the testing data set.b) Specify the number of type I errors and type II errors based on the confusion matrix.

# Testing data  
set.seed(123)  
  
  
Sigma = cbind(c(1, 0.8),c(0.8, 1.5))  
mu = c(0,0)  
mu = as.matrix(mu)  
mu = t(mu)  
  
Sigma2 = cbind(c(0.5, 0.4),c(0.4, 1))  
  
mu2 = c(0.5,0.5)  
mu2 = as.matrix(mu2)  
mu2 = t(mu2)  
nrows = 50  
  
x1 = data.frame(mvrnorm(n = 50, mu = mu , Sigma = Sigma))  
x2 = data.frame(mvrnorm(n = 50, mu = mu2, Sigma = Sigma2))  
x\_test = rbind(x1,x2)  
  
Y\_test = rep(c(1,-1), c(50,50))  
  
Y\_test2 = as.factor(Y\_test)  
dat.test = cbind(x\_test, Y\_test2)  
dat.test = data.frame(dat.test)  
  
results = predict(svmfit, dat.test)  
results = as.factor(results)  
  
ConfusionM.svm<-confusionMatrix(results, dat.test$Y)  
print(ConfusionM.svm)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction -1 1  
## -1 26 22  
## 1 24 28  
##   
## Accuracy : 0.54   
## 95% CI : (0.4374, 0.6402)  
## No Information Rate : 0.5   
## P-Value [Acc > NIR] : 0.2421   
##   
## Kappa : 0.08   
##   
## Mcnemar's Test P-Value : 0.8828   
##   
## Sensitivity : 0.5200   
## Specificity : 0.5600   
## Pos Pred Value : 0.5417   
## Neg Pred Value : 0.5385   
## Prevalence : 0.5000   
## Detection Rate : 0.2600   
## Detection Prevalence : 0.4800   
## Balanced Accuracy : 0.5400   
##   
## 'Positive' Class : -1   
##

#### 3.4 (Bonus 2 points) Use a for-loop to test the performance of the SVM model using radial kernel when using cost = 1, 10, 100, 1000, and report the performance based on the testing set by plotting the classification accuracy vs log10(cost).

#### Read section 11.6 of the textbook “Elements of Statistical Learning”. Follow the steps in the textbook and recreate the Figure 11.6, 11.7, and 11.8 for the Sum of sigmoids only. The objective of this question is helping you better understand the effect of parameter selection on the accuracy of the trained neural network.

# Generating the training data  
set.seed(100)  
x1\_train = data.frame(rnorm(100, mean = 0, sd = 1))  
set.seed(101)  
x2\_train = data.frame(rnorm(100, mean = 0, sd = 1))  
  
x\_train = cbind(x1\_train,x2\_train)  
colnames(x\_train) = c('x1', 'x2')  
x\_train2 = matrix(as.numeric(unlist(x\_train)),nrow=nrow(x\_train))  
  
a1 = c(3,3)  
a2 = c(3,-3)  
  
a1 = matrix(a1)  
a2 = matrix(a2)  
  
Y\_train = sigmoid(x\_train2 %\*% a1) + sigmoid(x\_train2 %\*% a2)  
  
  
# Generating the testing data  
set.seed(102)  
x1\_test = data.frame(rnorm(10000, mean = 0, sd = 1))  
set.seed(103)  
x2\_test = data.frame(rnorm(10000, mean = 0, sd = 1))  
  
x\_test = cbind(x1\_test,x2\_test)  
colnames(x\_test) = c('x1', 'x2')  
x\_test2 = matrix(as.numeric(unlist(x\_test)),nrow=nrow(x\_test))  
  
  
Y\_test = sigmoid(x\_test2 %\*% a1) + sigmoid(x\_test2 %\*% a2)

# Developing Neural Network  
  
  
AvgError = c()  
Size = c(1,2,3,4,5,6,7,8,9,10)  
  
  
# Standard Weight Decay  
  
for (i in seq(Size)){  
 ran.nn <- nnet(x\_train, Y\_train, size = i, rang = 0.1, decay = .005, maxit = 200)  
 predic = predict(ran.nn, x\_test2)  
 TestError = (Y\_test - predic)^2  
 AvgError = c(AvgError,TestError)  
}

## # weights: 5  
## initial value 54.509940   
## iter 10 value 31.485646  
## iter 20 value 17.816070  
## final value 17.812993   
## converged  
## # weights: 9  
## initial value 52.005357   
## iter 10 value 31.446865  
## iter 20 value 17.804562  
## iter 30 value 16.367286  
## iter 40 value 16.173539  
## iter 50 value 16.168836  
## final value 16.168819   
## converged  
## # weights: 13  
## initial value 51.680582   
## iter 10 value 31.504967  
## iter 20 value 17.755042  
## iter 30 value 16.640062  
## iter 40 value 16.112864  
## iter 50 value 16.104923  
## iter 60 value 16.101990  
## iter 70 value 16.101956  
## iter 70 value 16.101956  
## iter 70 value 16.101956  
## final value 16.101956   
## converged  
## # weights: 17  
## initial value 53.445620   
## iter 10 value 31.549499  
## iter 20 value 19.877531  
## iter 30 value 16.620749  
## iter 40 value 16.109660  
## iter 50 value 16.095710  
## iter 60 value 16.091725  
## iter 70 value 16.090594  
## iter 80 value 16.088429  
## iter 90 value 16.087972  
## iter 100 value 16.087891  
## iter 110 value 16.087886  
## iter 110 value 16.087885  
## iter 110 value 16.087885  
## final value 16.087885   
## converged  
## # weights: 21  
## initial value 56.318426   
## iter 10 value 31.607473  
## iter 20 value 17.789134  
## iter 30 value 17.529717  
## iter 40 value 16.106014  
## iter 50 value 16.085010  
## iter 60 value 16.082394  
## iter 70 value 16.082164  
## iter 80 value 16.082128  
## iter 90 value 16.082117  
## final value 16.082112   
## converged  
## # weights: 25  
## initial value 50.529298   
## iter 10 value 31.567234  
## iter 20 value 17.954000  
## iter 30 value 17.418806  
## iter 40 value 16.132203  
## iter 50 value 16.098089  
## iter 60 value 16.095189  
## iter 70 value 16.094144  
## iter 80 value 16.093524  
## iter 90 value 16.092831  
## iter 100 value 16.091950  
## iter 110 value 16.089699  
## iter 120 value 16.088523  
## iter 130 value 16.088403  
## iter 140 value 16.088383  
## iter 150 value 16.088334  
## iter 160 value 16.088249  
## iter 170 value 16.087993  
## iter 180 value 16.087873  
## iter 190 value 16.087828  
## iter 200 value 16.087785  
## final value 16.087785   
## stopped after 200 iterations  
## # weights: 29  
## initial value 56.469064   
## iter 10 value 31.683340  
## iter 20 value 23.861415  
## iter 30 value 17.724133  
## iter 40 value 16.166619  
## iter 50 value 16.110420  
## iter 60 value 16.087716  
## iter 70 value 16.056101  
## iter 80 value 16.040768  
## iter 90 value 16.038221  
## iter 100 value 16.037412  
## iter 110 value 16.036970  
## iter 120 value 16.036724  
## iter 130 value 16.036608  
## iter 140 value 16.036522  
## iter 150 value 16.036441  
## iter 160 value 16.036416  
## iter 170 value 16.036401  
## iter 180 value 16.036394  
## final value 16.036393   
## converged  
## # weights: 33  
## initial value 56.590796   
## iter 10 value 31.719995  
## iter 20 value 24.456594  
## iter 30 value 17.607185  
## iter 40 value 16.182188  
## iter 50 value 16.126185  
## iter 60 value 16.120386  
## iter 70 value 16.117074  
## iter 80 value 16.112329  
## iter 90 value 16.103749  
## iter 100 value 16.077739  
## iter 110 value 16.060549  
## iter 120 value 16.052117  
## iter 130 value 16.046351  
## iter 140 value 16.043270  
## iter 150 value 16.039841  
## iter 160 value 16.038253  
## iter 170 value 16.037364  
## iter 180 value 16.037020  
## iter 190 value 16.036951  
## iter 200 value 16.036889  
## final value 16.036889   
## stopped after 200 iterations  
## # weights: 37  
## initial value 57.354939   
## iter 10 value 31.758972  
## iter 20 value 19.066724  
## iter 30 value 17.512933  
## iter 40 value 16.223262  
## iter 50 value 16.130901  
## iter 60 value 16.128156  
## iter 70 value 16.122384  
## iter 80 value 16.107778  
## iter 90 value 16.081420  
## iter 100 value 16.064107  
## iter 110 value 16.055016  
## iter 120 value 16.050982  
## iter 130 value 16.049044  
## iter 140 value 16.043675  
## iter 150 value 16.042281  
## iter 160 value 16.041608  
## iter 170 value 16.041012  
## iter 180 value 16.039789  
## iter 190 value 16.037185  
## iter 200 value 16.036269  
## final value 16.036269   
## stopped after 200 iterations  
## # weights: 41  
## initial value 51.591646   
## iter 10 value 31.696476  
## iter 20 value 26.065030  
## iter 30 value 17.680911  
## iter 40 value 16.413951  
## iter 50 value 16.148072  
## iter 60 value 16.140534  
## iter 70 value 16.130118  
## iter 80 value 16.105469  
## iter 90 value 16.082572  
## iter 100 value 16.067754  
## iter 110 value 16.056964  
## iter 120 value 16.051836  
## iter 130 value 16.048630  
## iter 140 value 16.041588  
## iter 150 value 16.037989  
## iter 160 value 16.036759  
## iter 170 value 16.036612  
## iter 180 value 16.036579  
## final value 16.036571   
## converged

AvgError2 = c()  
Size = c(1,2,3,4,5,6,7,8,9,10)  
  
# No Weight Decay  
  
for (i in seq(Size)){  
 ran.nn <- nnet(x\_train, Y\_train, size = i, rang = 0.1, decay = 0, maxit = 200)  
 predic = predict(ran.nn, x\_test2)  
 TestError = (Y\_test - predic)^2  
 AvgError2 = c(AvgError2,TestError)  
}

## # weights: 5  
## initial value 53.845462   
## final value 31.314791   
## converged  
## # weights: 9  
## initial value 54.617091   
## final value 31.314791   
## converged  
## # weights: 13  
## initial value 58.985732   
## final value 31.314791   
## converged  
## # weights: 17  
## initial value 53.578982   
## final value 31.314791   
## converged  
## # weights: 21  
## initial value 56.111726   
## final value 31.314791   
## converged  
## # weights: 25  
## initial value 54.527393   
## final value 31.314791   
## converged  
## # weights: 29  
## initial value 54.396699   
## final value 31.314791   
## converged  
## # weights: 33  
## initial value 51.916905   
## final value 31.314791   
## converged  
## # weights: 37  
## initial value 48.968696   
## final value 31.314791   
## converged  
## # weights: 41  
## initial value 53.549527   
## final value 31.314791   
## converged

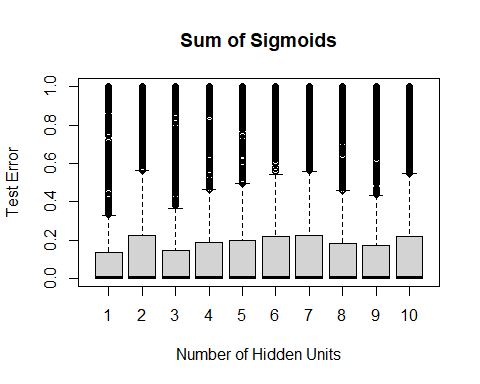
AvgError3 = c()  
Size = c(1,2,3,4,5,6,7,8,9,10)  
  
# Large Weight Decay  
  
for (i in Size){  
 ran.nn <- nnet(x\_train, Y\_train, size = i, rang = 0.1, decay = .1, maxit = 200)  
 predic = predict(ran.nn, x\_test2)  
 TestError = (Y\_test - predic)^2  
 AvgError3 = c(AvgError3,TestError)  
}

## # weights: 5  
## initial value 55.683961   
## iter 10 value 22.094028  
## final value 21.977936   
## converged  
## # weights: 9  
## initial value 58.015376   
## iter 10 value 21.330001  
## iter 20 value 21.198894  
## iter 30 value 21.162559  
## final value 21.161211   
## converged  
## # weights: 13  
## initial value 56.154658   
## iter 10 value 22.841612  
## iter 20 value 20.965897  
## iter 30 value 20.954946  
## iter 40 value 20.928246  
## final value 20.927823   
## converged  
## # weights: 17  
## initial value 56.460262   
## iter 10 value 28.227609  
## iter 20 value 20.889193  
## iter 30 value 20.870339  
## iter 40 value 20.861566  
## iter 50 value 20.861311  
## final value 20.861310   
## converged  
## # weights: 21  
## initial value 56.009776   
## iter 10 value 22.854209  
## iter 20 value 20.871062  
## iter 30 value 20.869216  
## iter 40 value 20.858595  
## iter 50 value 20.858360  
## final value 20.858352   
## converged  
## # weights: 25  
## initial value 54.616355   
## iter 10 value 22.062540  
## iter 20 value 20.887458  
## iter 30 value 20.887314  
## final value 20.887291   
## converged  
## # weights: 29  
## initial value 55.926462   
## iter 10 value 28.214186  
## iter 20 value 20.874802  
## iter 30 value 20.483845  
## iter 40 value 20.422788  
## iter 50 value 20.419362  
## iter 60 value 20.419043  
## iter 70 value 20.419006  
## iter 80 value 20.418998  
## final value 20.418996   
## converged  
## # weights: 33  
## initial value 52.628270   
## iter 10 value 22.073983  
## iter 20 value 20.968316  
## iter 30 value 20.966475  
## iter 40 value 20.965575  
## iter 50 value 20.965355  
## iter 60 value 20.965331  
## iter 60 value 20.965331  
## iter 60 value 20.965331  
## final value 20.965331   
## converged  
## # weights: 37  
## initial value 56.487233   
## iter 10 value 30.619576  
## iter 20 value 21.017713  
## iter 30 value 21.016278  
## iter 40 value 21.011923  
## iter 50 value 20.804991  
## iter 60 value 20.490845  
## iter 70 value 20.485849  
## iter 80 value 20.456066  
## iter 90 value 20.411320  
## iter 100 value 20.408850  
## iter 110 value 20.407698  
## final value 20.407629   
## converged  
## # weights: 41  
## initial value 54.159894   
## iter 10 value 30.276730  
## iter 20 value 21.071312  
## iter 30 value 21.021683  
## iter 40 value 20.465060  
## iter 50 value 20.401725  
## iter 60 value 20.393408  
## iter 70 value 20.391526  
## final value 20.391511   
## converged

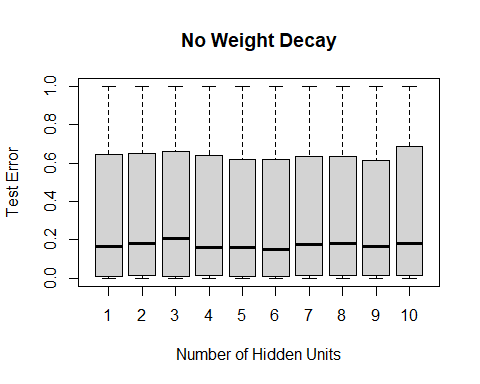
AvgError4 = c()  
Weights = c(.00,.01,.02,.03,.04,.05,.06,.07,.08,.09,.10,.11,.12,.13,.14,.15)  
  
# Changing Weights  
  
for (i in Weights){  
 ran.nn <- nnet(x\_train, Y\_train, size = 10, rang = 0.1, decay = i, maxit = 200)  
 predic = predict(ran.nn, x\_test2)  
 TestError = (Y\_test - predic)^2  
 AvgError4 = c(AvgError4,TestError)  
}

## # weights: 41  
## initial value 51.298925   
## final value 31.314791   
## converged  
## # weights: 41  
## initial value 51.930212   
## iter 10 value 32.109423  
## iter 20 value 18.437373  
## iter 30 value 17.878109  
## iter 40 value 16.640589  
## iter 50 value 16.616303  
## iter 60 value 16.547563  
## iter 70 value 16.500867  
## iter 80 value 16.469441  
## iter 90 value 16.456466  
## iter 100 value 16.451083  
## iter 110 value 16.447488  
## iter 120 value 16.445113  
## iter 130 value 16.444451  
## iter 140 value 16.443309  
## iter 150 value 16.440287  
## iter 160 value 16.438040  
## iter 170 value 16.437130  
## iter 180 value 16.437053  
## iter 190 value 16.437020  
## final value 16.437013   
## converged  
## # weights: 41  
## initial value 59.479822   
## iter 10 value 31.412097  
## iter 20 value 18.556990  
## iter 30 value 17.447754  
## iter 40 value 17.286057  
## iter 50 value 17.150792  
## iter 60 value 17.124781  
## iter 70 value 17.115050  
## iter 80 value 17.110719  
## iter 90 value 17.110537  
## iter 100 value 17.110475  
## iter 110 value 17.110461  
## final value 17.110457   
## converged  
## # weights: 41  
## initial value 59.228373   
## iter 10 value 30.918532  
## iter 20 value 18.784356  
## iter 30 value 17.975044  
## iter 40 value 17.793047  
## iter 50 value 17.709413  
## iter 60 value 17.698757  
## iter 70 value 17.696262  
## iter 80 value 17.695575  
## iter 90 value 17.693411  
## iter 100 value 17.692946  
## iter 110 value 17.692874  
## iter 120 value 17.692861  
## final value 17.692856   
## converged  
## # weights: 41  
## initial value 52.031884   
## iter 10 value 28.829994  
## iter 20 value 19.174106  
## iter 30 value 18.705870  
## iter 40 value 18.481004  
## iter 50 value 18.263090  
## iter 60 value 18.216124  
## iter 70 value 18.210629  
## iter 80 value 18.209093  
## iter 90 value 18.208862  
## iter 100 value 18.208798  
## iter 110 value 18.208792  
## final value 18.208790   
## converged  
## # weights: 41  
## initial value 58.200773   
## iter 10 value 30.705347  
## iter 20 value 19.506468  
## iter 30 value 18.829242  
## iter 40 value 18.714442  
## iter 50 value 18.678250  
## iter 60 value 18.673069  
## iter 70 value 18.671235  
## iter 80 value 18.670805  
## iter 90 value 18.670740  
## final value 18.670738   
## converged  
## # weights: 41  
## initial value 53.310712   
## iter 10 value 31.100024  
## iter 20 value 20.605774  
## iter 30 value 19.303462  
## iter 40 value 19.110908  
## iter 50 value 19.093935  
## iter 60 value 19.093129  
## iter 70 value 19.092996  
## iter 80 value 19.090630  
## iter 90 value 19.089847  
## iter 100 value 19.089508  
## iter 110 value 19.089379  
## iter 120 value 19.089326  
## iter 130 value 19.089319  
## iter 140 value 19.089315  
## final value 19.089311   
## converged  
## # weights: 41  
## initial value 54.689681   
## iter 10 value 29.662445  
## iter 20 value 20.203682  
## iter 30 value 19.896981  
## iter 40 value 19.523230  
## iter 50 value 19.474852  
## iter 60 value 19.470363  
## iter 70 value 19.470169  
## iter 80 value 19.470087  
## iter 90 value 19.470060  
## final value 19.470059   
## converged  
## # weights: 41  
## initial value 58.109008   
## iter 10 value 28.392050  
## iter 20 value 20.482258  
## iter 30 value 19.951956  
## iter 40 value 19.830922  
## iter 50 value 19.818275  
## iter 60 value 19.817025  
## iter 70 value 19.816568  
## iter 80 value 19.816357  
## final value 19.816355   
## converged  
## # weights: 41  
## initial value 57.860424   
## iter 10 value 28.302282  
## iter 20 value 20.520155  
## iter 30 value 20.264316  
## iter 40 value 20.143725  
## iter 50 value 20.128036  
## iter 60 value 20.125328  
## iter 70 value 20.124327  
## iter 80 value 20.124050  
## iter 90 value 20.123994  
## iter 100 value 20.123961  
## iter 110 value 20.123926  
## iter 120 value 20.123879  
## iter 130 value 20.123365  
## iter 140 value 20.122786  
## iter 150 value 20.122572  
## final value 20.122570   
## converged  
## # weights: 41  
## initial value 55.202224   
## iter 10 value 30.412381  
## iter 20 value 21.067838  
## iter 30 value 20.840104  
## iter 40 value 20.501118  
## iter 50 value 20.455578  
## iter 60 value 20.403070  
## iter 70 value 20.399356  
## iter 80 value 20.395222  
## iter 90 value 20.394685  
## iter 100 value 20.393234  
## iter 110 value 20.392899  
## iter 120 value 20.392883  
## iter 130 value 20.392869  
## iter 140 value 20.392856  
## final value 20.392855   
## converged  
## # weights: 41  
## initial value 56.467194   
## iter 10 value 30.691138  
## iter 20 value 21.301161  
## iter 30 value 20.745675  
## iter 40 value 20.638514  
## iter 50 value 20.629589  
## iter 60 value 20.628845  
## final value 20.628823   
## converged  
## # weights: 41  
## initial value 51.002946   
## iter 10 value 30.103847  
## iter 20 value 21.607866  
## iter 30 value 21.328097  
## iter 40 value 20.858589  
## iter 50 value 20.852971  
## iter 60 value 20.852811  
## final value 20.852811   
## converged  
## # weights: 41  
## initial value 54.725368   
## iter 10 value 28.557309  
## iter 20 value 21.869006  
## iter 30 value 21.757791  
## iter 40 value 21.124883  
## iter 50 value 21.069910  
## iter 60 value 21.069517  
## final value 21.069512   
## converged  
## # weights: 41  
## initial value 53.623676   
## iter 10 value 30.413325  
## iter 20 value 22.118523  
## iter 30 value 21.907641  
## iter 40 value 21.419460  
## iter 50 value 21.417561  
## iter 60 value 21.417467  
## final value 21.417466   
## converged  
## # weights: 41  
## initial value 59.488878   
## iter 10 value 29.991913  
## iter 20 value 22.360891  
## iter 30 value 22.282816  
## iter 40 value 21.684877  
## iter 50 value 21.634728  
## iter 60 value 21.631715  
## iter 70 value 21.631624  
## final value 21.631623   
## converged

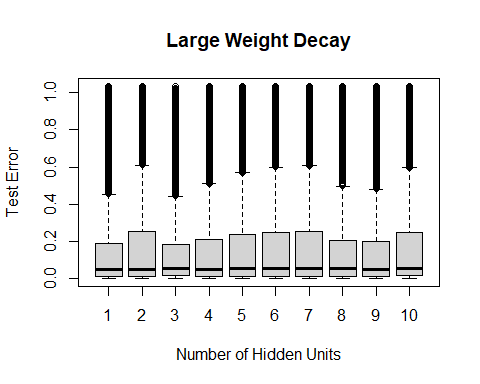
# Making the Box plots  
  
Size = data.frame(Size)  
Weights = data.frame(Weights)  
Error1 = data.frame(AvgError)  
Error2 = data.frame(AvgError2)  
Error3 = data.frame(AvgError3)  
Error4 = data.frame(AvgError4)  
  
Error1 = cbind.data.frame(Size, Error1)  
Error2 = cbind.data.frame(Size, Error2)  
Error3 = cbind.data.frame(Size, Error3)  
Error4 = cbind.data.frame(Weights, Error4)  
  
  
  
boxplot(AvgError~Size, data = Error1, main = "Sum of Sigmoids", xlab = "Number of Hidden Units", ylab = "Test Error")



boxplot(AvgError2~Size, data = Error2, main = "No Weight Decay", xlab = "Number of Hidden Units", ylab = "Test Error")



boxplot(AvgError3~Size, data = Error3, main = "Large Weight Decay", xlab = "Number of Hidden Units", ylab = "Test Error")



boxplot(AvgError4~Weights, data = Error4, main = "Sum of Sigmoids, 10 units", xlab = "Weight Decay Parameter", ylab = "Test Error")

