Homework - 3

Group 04

# Problem 1: Gradient Descent Algorithm for Multiple Linear Regression

The file concrete.csv includes 1,030 types of concrete with numerical features indicating characteristics of the concrete. The variable “strength” is treated as the response variable.

• Standardize all variables (including the response variable “strength”). Split the data set into a training set (60%) and a validation set (40%).

• Implement the gradient descent algorithm in R with the ordinary least square cost function.

• Fit the multiple linear regression model using the gradient descent algorithm and the training set. Try out different learning rates: 𝛼 = 0.01,0.1,0.3,0.5 and compare the speed of convergence by plotting the cost function. Determine the number of iterations needed for each 𝛼 value.

• Apply the fitted regression model to the validation set and evaluate the model performance (ME, RMSE, MAE, MPE, MPAE). Calculate the correlation between the predicted strength and the actual strength. Create a lift chart to show model performance.

# Import Required Packages  
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(data.table)

##   
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

library(reshape2)

##   
## Attaching package: 'reshape2'

## The following objects are masked from 'package:data.table':  
##   
## dcast, melt

library(MLmetrics)

##   
## Attaching package: 'MLmetrics'

## The following object is masked from 'package:base':  
##   
## Recall

library(ggplot2)  
  
# Read the csv file  
df <- data.table(read.csv("concrete.csv"))  
  
# Scale the dataframe  
df <- as.data.frame(scale(df))  
  
# Split into train and validation datasets  
training\_rows <- sample(seq\_len(nrow(df)), size = floor(0.6 \* nrow(df)))  
  
train\_data <- df[training\_rows, ]  
validation\_data <- df[-training\_rows, ]

Implementing Gradient Descent algorithm with the Ordinary Least Square cost function.

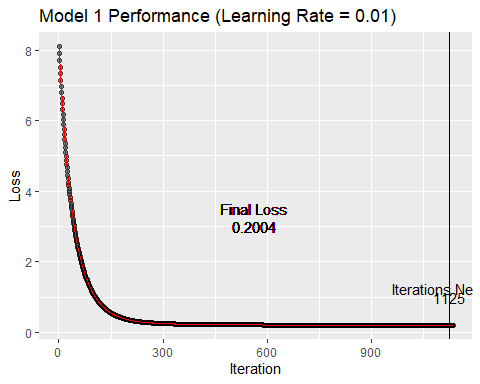
# Define the gradient descent function  
gradient\_desc <- function(x, y, lr, iters) {  
 # First we create a list to keep the track  
 # of the cost function for each iteration  
 losses <- list()  
  
 # Convert y to a matrix  
 y <- as.matrix(y)  
  
 # create a column of 1  
 ones <- rep(1, dim(x)[[1]])  
 # append it to the input (this is our X0)  
 X <- as.matrix(cbind(ones, x))  
 # Calculate number of samples  
 n <- length(y)  
  
 # Initialize model parameters/coefficients  
 theta <- as.matrix(rnorm(n = dim(X)[2], 0, 1))  
  
 # Calculate model predictions  
 y\_hat <- X %\*% theta  
  
 # calculate the loss using OLS cost function  
 loss <- sum((y\_hat - y)^2) / (2 \* n)  
  
 # Calculate the gradients of the cost function  
 grads <- t(X) %\*% (y\_hat - y)  
  
 # Update theta  
 theta <- theta - lr \* (1 / n) \* grads  
  
  
 # That was the first iteration of the gradient descent algorithm  
 # Let's add the cost function to the list  
 losses[[1]] <- loss  
  
  
 counter <- 0  
 # Number of iterations required to get the lowest loss  
 sufficient\_iterations <- 0  
 for (i in 1:iters) {  
 # Calculate model predictions  
 y\_hat <- X %\*% theta  
  
 # Calculate the loss using OLS cost function  
 loss <- sum((y\_hat - y)^2) / (2 \* n)  
  
 # Calculate the gradients  
 grads <- t(X) %\*% (y\_hat - y)  
  
 # Update theta  
 theta <- theta - lr \* (1 / n) \* grads  
  
 # Add cost to the list  
 losses[[i + 1]] <- loss  
  
 if (round(losses[[i]], 4) <= round(loss, 4)) {  
 if (counter > 6) {  
 break  
 } else {  
 counter <- counter + 1  
 sufficient\_iterations <- sufficient\_iterations + 1  
 }  
 } else {  
 counter <- 0  
 sufficient\_iterations <- sufficient\_iterations + 1  
 }  
 }  
  
  
 sufficient\_iterations <- sufficient\_iterations - counter  
 # return the theta (aka model weights)  
 return(list(  
 "coeffs" = theta,  
 "losses" = losses,  
 "iterations\_required" = sufficient\_iterations,  
 "final\_loss" = loss  
 ))  
}  
  
# Predict function  
predict <- function(x, theta) {  
 ones <- rep(1, dim(x)[[1]])  
 # append it to the input (this is our X0)  
 X <- as.matrix(cbind(ones, x))  
  
 return(X %\*% t(theta))  
}

Now we create and train 4 models each with a different learning rate

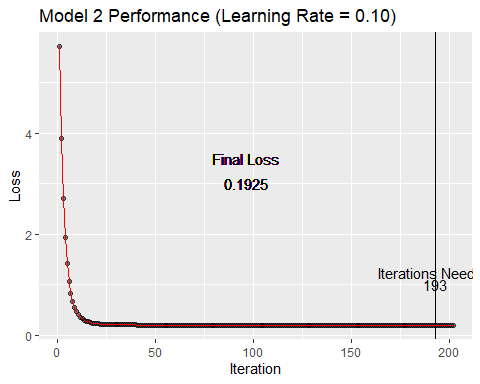
# Model 1, lr = 0.01  
model1 <- gradient\_desc(train\_data[, 1:8], train\_data$strength, lr = 0.01, iters = 10000)  
  
model1\_weights <- t(model1$coeffs)  
model1\_losses <- melt(data.frame(model1$losses))  
model1\_losses$index <- 1:dim(model1\_losses)[[1]]  
  
  
# Model 2, lr = 0.10  
model2 <- gradient\_desc(train\_data[, 1:8], train\_data$strength, lr = 0.10, iters = 10000)  
  
model2\_weights <- t(model2$coeffs)  
model2\_losses <- melt(data.frame(model2$losses))  
model2\_losses$index <- 1:dim(model2\_losses)[[1]]  
  
# Model 3, lr = 0.30  
model3 <- gradient\_desc(train\_data[, 1:8], train\_data$strength, lr = 0.30, iters = 10000)  
  
model3\_weights <- t(model3$coeffs)  
model3\_losses <- melt(data.frame(model3$losses))  
model3\_losses$index <- 1:dim(model3\_losses)[[1]]  
  
# Model 4, lr = 0.50  
model4 <- gradient\_desc(train\_data[, 1:8], train\_data$strength, lr = 0.50, iters = 10000)  
  
model4\_weights <- t(model4$coeffs)  
model4\_losses <- melt(data.frame(model4$losses))  
model4\_losses$index <- 1:dim(model4\_losses)[[1]]

Let’s plot the loss vs number of iterations for each model to evaluate their performance.

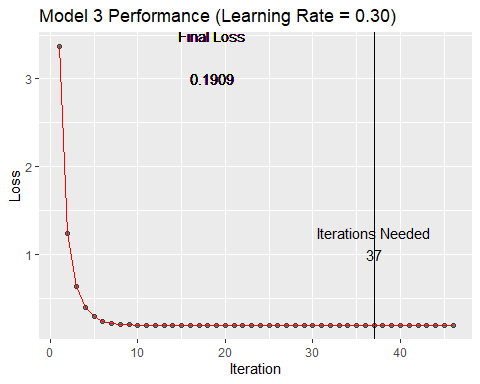
# Model 1  
ggplot(model1\_losses, aes(x = index, y = value)) +  
 geom\_point(alpha = 0.5) +  
 geom\_vline(xintercept = model1$iterations\_required) +  
 geom\_text(x = model1$iterations\_required / 2, y = 3.5, label = "Final Loss") +  
 geom\_text(x = model1$iterations\_required / 2, y = 3, label = as.character(round(model1$final\_loss, 4))) +  
 geom\_text(  
 x = model1$iterations\_required,  
 y = 1,  
 label = as.character(model1$iterations\_required),  
 check\_overlap = TRUE  
 ) +  
 geom\_text(  
 x = model1$iterations\_required,  
 y = 1.25,  
 label = "Iterations Needed",  
 check\_overlap = TRUE  
 ) +  
 geom\_line(color = "red") +  
 labs(x = "Iteration", y = "Loss") +  
 ggtitle("Model 1 Performance (Learning Rate = 0.01)")



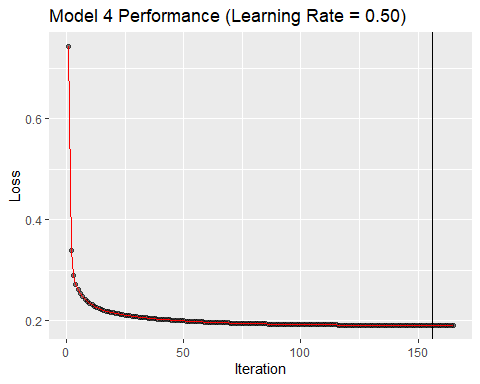
# Model 2  
ggplot(model2\_losses, aes(x = index, y = value)) +  
 geom\_point(alpha = 0.5) +  
 geom\_vline(xintercept = model2$iterations\_required) +  
 geom\_text(x = model2$iterations\_required / 2, y = 3.5, label = "Final Loss") +  
 geom\_text(x = model2$iterations\_required / 2, y = 3, label = as.character(round(model2$final\_loss, 4))) +  
 geom\_text(  
 x = model2$iterations\_required,  
 y = 1,  
 label = as.character(model2$iterations\_required),  
 check\_overlap = TRUE  
 ) +  
 geom\_text(  
 x = model2$iterations\_required,  
 y = 1.25,  
 label = "Iterations Needed",  
 check\_overlap = TRUE  
 ) +  
 geom\_line(color = "red") +  
 labs(x = "Iteration", y = "Loss") +  
 ggtitle("Model 2 Performance (Learning Rate = 0.10)")



# Model 3  
ggplot(model3\_losses, aes(x = index, y = value)) +  
 geom\_point(alpha = 0.5) +  
 geom\_vline(xintercept = model3$iterations\_required) +  
 geom\_text(x = model3$iterations\_required / 2, y = 3.5, label = "Final Loss") +  
 geom\_text(x = model3$iterations\_required / 2, y = 3, label = as.character(round(model3$final\_loss, 4))) +  
 geom\_text(  
 x = model3$iterations\_required,  
 y = 1,  
 label = as.character(model3$iterations\_required),  
 check\_overlap = TRUE  
 ) +  
 geom\_text(  
 x = model3$iterations\_required,  
 y = 1.25,  
 label = "Iterations Needed",  
 check\_overlap = TRUE  
 ) +  
 geom\_line(color = "red") +  
 labs(x = "Iteration", y = "Loss") +  
 ggtitle("Model 3 Performance (Learning Rate = 0.30)")



# Model 4  
ggplot(model4\_losses, aes(x = index, y = value)) +  
 geom\_point(alpha = 0.5) +  
 geom\_vline(xintercept = model4$iterations\_required) +  
 geom\_text(x = model4$iterations\_required / 2, y = 3.5, label = "Final Loss") +  
 geom\_text(x = model4$iterations\_required / 2, y = 3, label = as.character(round(model4$final\_loss, 4))) +  
 geom\_text(  
 x = model4$iterations\_required,  
 y = 1,  
 label = as.character(model4$iterations\_required),  
 check\_overlap = TRUE  
 ) +  
 geom\_text(  
 x = model4$iterations\_required,  
 y = 1.25,  
 label = "Iterations Needed",  
 check\_overlap = TRUE  
 ) +  
 geom\_line(color = "red") +  
 labs(x = "Iteration", y = "Loss") +  
 ggtitle("Model 4 Performance (Learning Rate = 0.50)")



cat("Number of iterations required for each model are :\n")

## Number of iterations required for each model are :

cat("Model 1:", as.character(model1$iterations\_required), "\n")

## Model 1: 1125

cat("Model 2:", as.character(model2$iterations\_required), "\n")

## Model 2: 193

cat("Model 3:", as.character(model3$iterations\_required), "\n")

## Model 3: 37

cat("Model 4:", as.character(model4$iterations\_required), "\n")

## Model 4: 156

As observed, the model converges faster as the learning rate increases.

Testing the model on the validation data and calulcating errors -

# We define the Mean Error function  
ME <- function(y\_hat, y) {  
 sum(y - y\_hat) / length(y)  
}  
  
# We define the Mean Percentage Error Function  
MPE <- function(y\_hat, y) {  
 (sum((y - y\_hat) / y)) / length(y)  
}

Now let’s look at the model statstics -

model1\_predictions <- predict(validation\_data[, 1:8], model1\_weights)  
  
cat("----Model 1 Summary ----\n")

## ----Model 1 Summary ----

cat("MAE:", MAE(model1\_predictions, validation\_data[, 9]), "\n")

## MAE: 0.5103062

cat("RMSE:", RMSE(model1\_predictions, validation\_data[, 9]), "\n")

## RMSE: 0.6552274

cat("ME:", ME(model1\_predictions, validation\_data[, 9]), "\n")

## ME: 0.01826382

cat("MPE:", MPE(model1\_predictions, validation\_data[, 9]), "\n")

## MPE: 0.6028765

cat("MPAE", MAPE(model1\_predictions, validation\_data[, 9]), "\n")

## MPAE 1.437657

model2\_predictions <- predict(validation\_data[, 1:8], model2\_weights)  
  
cat("----Model 2 Summary ---- \n")

## ----Model 2 Summary ----

cat("MAE:", MAE(model2\_predictions, validation\_data[, 9]), "\n")

## MAE: 0.494978

cat("RMSE:", RMSE(model2\_predictions, validation\_data[, 9]), "\n")

## RMSE: 0.6226464

cat("ME:", ME(model2\_predictions, validation\_data[, 9]), "\n")

## ME: 0.02408618

cat("MPE:", MPE(model2\_predictions, validation\_data[, 9]), "\n")

## MPE: 0.2804273

cat("MPAE", MAPE(model2\_predictions, validation\_data[, 9]), "\n")

## MPAE 1.379004

model3\_predictions <- predict(validation\_data[, 1:8], model3\_weights)  
  
cat("----Model 3 Summary ----\n")

## ----Model 3 Summary ----

cat("MAE:", MAE(model3\_predictions, validation\_data[, 9]), "\n")

## MAE: 0.4933958

cat("RMSE:", RMSE(model3\_predictions, validation\_data[, 9]), "\n")

## RMSE: 0.6249209

cat("ME:", ME(model3\_predictions, validation\_data[, 9]), "\n")

## ME: 0.02252379

cat("MPE:", MPE(model3\_predictions, validation\_data[, 9]), "\n")

## MPE: 0.3620293

cat("MPAE", MAPE(model3\_predictions, validation\_data[, 9]), "\n")

## MPAE 1.321165

model4\_predictions <- predict(validation\_data[, 1:8], model4\_weights)  
  
cat("----Model 4 Summary ----\n")

## ----Model 4 Summary ----

cat("MAE:", MAE(model4\_predictions, validation\_data[, 9]), "\n")

## MAE: 0.4932587

cat("RMSE:", RMSE(model4\_predictions, validation\_data[, 9]), "\n")

## RMSE: 0.6236018

cat("ME:", ME(model4\_predictions, validation\_data[, 9]), "\n")

## ME: 0.02270841

cat("MPE:", MPE(model4\_predictions, validation\_data[, 9]), "\n")

## MPE: 0.3420679

cat("MPAE", MAPE(model4\_predictions, validation\_data[, 9]), "\n")

## MPAE 1.336092

We can see that all the models have approximately the same accuracy regardless the learning rate.

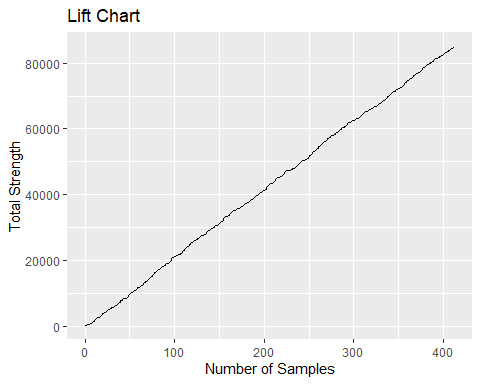
Calculating the correlation between predicted strength and actual strength

cat("The correlation is :", cor(model1\_predictions, validation\_data[, 9]), "\n")

## The correlation is : 0.7680595

Plotting a lift chart

# Create a temp data frame to calculate the sumulative strength  
temp <- data.frame("strength" = order(validation\_data[, 9]))  
temp$cumstrength <- cumsum(temp$strength)  
temp$samples <- 1:dim(temp)[[1]]  
  
# Plot the lift chart  
ggplot(temp, aes(x = samples, y = cumstrength)) +  
 geom\_line() +  
 labs(x = "Number of Samples", y = "Total Strength") +  
 ggtitle("Lift Chart")



# Delete all environment variables  
rm(list = ls())