# Chapter 18 Interactive Notebook for Students

Ram Gopal, Dan Philps, and Tillman Weyde

2022

## Contents

**Load packages** 2

**Get the data and pre-process** 2

- Read data . . . 2
- Partition Data . . . 3

**Linear Regression Model** 3

**Compute functions for Residual Mean,MSE, RMSE and R_2** 3

**Performance of the Linear Regression Model** 4

**Polynomial Regression Model** 4

- Generate the dataset . . . 5
- Create training, validation and test sets . . . 5
- Run the Polynomial Regression . . . 6
- Evaluate the results . . . 6

**Ridge Regression** 7

- Basic Model . . . 7
- Hyper-parameter tuning . . . 7
- Run the Optimized Ridge Regression . . . 7
- Evaluate Performance of the optimized Ridge Regression . . . 8

**Lasso Regression** 9

- Basic Model . . . 9
- Hyper-parameter tuning . . . 9
- Run the Optimized Lasso Regression . . . 9
- Performance of the Optimized Lasso Regression . . . 10
- Determine non-zero coefficients . . . 11

**Neural Network** **11**
Large network . . . 11
Smaller network . . . 12

**Regression Tree** **12**
Basic Tree . . . 12
Tree with a maximum depth of 2 . . . 13

**Harder problem** **14**
Data preparation . . . 14
Build and Evaluate Models . . . 14

# Load packages

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(rattle)
```

```
## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

# Get the data and pre-process

## Read data

```
dataset <- read.csv('diabetes.csv', header = TRUE)
dim(dataset)
```

```
## [1] 442  11
```

### Partition Data

- Note the use of sample() function to create training, validation, and test datasets. Another interesting function is sapply(). This is part of a list of so-called apply() functions in R. They essentially allow you to run a function on multiple different inputs. In the case below, we want to get the number of rows (which is done through a function) for three different dataframes. Instead of writing three different lines of code, sapply() makes the coding simpler and more elegant.

```
set.seed(123456)
N = nrow(dataset)
cut1 = floor(0.6*N)
cut2 = floor(0.8*N)
index = sample(1:N)
train_index = index[1:cut1]
val_index = index[(cut1+1):cut2]
test_index = index[(cut2+1):N]
df_train = dataset[train_index,]
df_val = dataset[val_index,]
df_test = dataset[test_index,]
sapply(list(df_train,df_val,df_test),nrow)
```

```
## [1] 265  88  89
```

## Linear Regression Model

- For this and the subsequent chapter, we will make use of the caret package that provides a uniform approach to work with a variety of predictive models. Also note the use of :: notation. This is used to indicate the package that contains the function that follows. This is helpful as often different packages may use the same function names.

```
df_train[-11] = scale(df_train[-11])
lr = caret::train(target ~ ., method='lm',data = df_train)
train_pred = predict(lr,newdata = df_train)
val_pred = predict(lr,newdata = scale(df_val[-11]))
test_pred = predict(lr,newdata = scale(df_test[-11]))
```

## Compute functions for Residual Mean,MSE, RMSE and R__2

```
rm <- function(actual,pred) {
  return(mean(abs(actual-pred)))
}
mse <- function(actual,pred) {
  return(mean((pred-actual)^2))
}
rmse <- function(actual,pred) {
  return(mse(pred,actual)^0.5)
}
R_2 <- function(actual,pred){
```

```
  mean_v = rep(mean(actual),length(actual))
  SST = sum((actual-mean_v)^2)
  SSE = sum((actual-pred)^2)
  return(1-(SSE/SST))
}
```

## Performance of the Linear Regression Model

- In the code below, we want to assess the performance of the model on training, validation, and test datasets. To make it easier to read and compare, we will put the results in a dataframe. Also note the use of lapply() function - in this case we want to change the data type of all the variables, except the first column, to numeric. Again, this function makes it easier to code.

```
res = data.frame()
w = rm(df_train$target,train_pred)
x = mse(df_train$target,train_pred)
y = rmse(df_train$target,train_pred)
z = R_2(df_train$target,train_pred)
res = rbind(res,c("Train",w,x,y,z))
w = rm(df_val$target,val_pred)
x = mse(df_val$target,val_pred)
y = rmse(df_val$target,val_pred)
z = R_2(df_val$target,val_pred)
res = rbind(res,c("Validation",w,x,y,z))
w = rm(df_test$target,test_pred)
x = mse(df_test$target,test_pred)
y = rmse(df_test$target,test_pred)
z = R_2(df_test$target,test_pred)
res = rbind(res,c("Test",w,x,y,z))
colnames(res) = c("Data","Residual Mean","MSE","RMSE","R_2")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | Residual Mean | MSE | RMSE | R_2 |
|---|---|---|---|---|
| Train | 41.73 | 2655 | 51.52 | 0.5749 |
| Validation | 47.85 | 3548 | 59.57 | 0.3137 |
| Test | 43.83 | 3023 | 54.98 | 0.4676 |

## Polynomial Regression Model

- Column names of the original data set.

```
colnames(df_train)
```

```
##  [1] "age"    "sex"    "bmi"    "bp"     "s1"     "s2"     "s3"     "s4"
##  [9] "s5"     "s6"     "target"
```

### Generate the dataset

- The strategy is to write a formula for squared and interaction terms, and use the formula to generate a new dataset that contains the polynomial terms.

```
# Degree 2 polynomial feature generation function

pf2_transform <- function(df, target_name='target') {
  formula_pf2 <- as.formula(paste(target_name, '~ .^2 +',
                                  paste('poly(',
                                        colnames(df)[-c(1)],
                                       ',2, raw=TRUE)[, 2]',
                                        collapse = ' + ')
                                  )
                            )
   output <- model.matrix(formula_pf2, data = df)
  # Rewrite column names for readability
   colnames_pf2 <- c("1",
                    colnames(df)[-1],              # exclude target
                    paste0(colnames(df)[-1],"^2"), # include squares
                    colnames(output)[-(1:(length(df)*2-1))]) # include interactions
    colnames(output) <- colnames_pf2
  # Convert to dataframe
  output_df <- data.frame(output)
  # Exclude intercept column
  output_df[,1] <- NULL
  return(output_df)
}
```

### Create training, validation and test sets

- Create the training data set

```
train_sc_pf2 <- pf2_transform(df_train,target_name = "target")
train_sc_pf2$target= df_train$target
train_sc_pf2 = train_sc_pf2[-20]
print(colnames(train_sc_pf2))
```

```
##  [1] "sex"     "bmi"     "bp"      "s1"      "s2"      "s3"      "s4"
##  [8] "s5"      "s6"      "target"  "sex.2"   "bmi.2"   "bp.2"    "s1.2"
## [15] "s2.2"    "s3.2"    "s4.2"    "s5.2"    "s6.2"    "age.sex" "age.bmi"
## [22] "age.bp"  "age.s1"  "age.s2"  "age.s3"  "age.s4"  "age.s5"  "age.s6"
## [29] "sex.bmi" "sex.bp"  "sex.s1"  "sex.s2"  "sex.s3"  "sex.s4"  "sex.s5"
## [36] "sex.s6"  "bmi.bp"  "bmi.s1"  "bmi.s2"  "bmi.s3"  "bmi.s4"  "bmi.s5"
## [43] "bmi.s6"  "bp.s1"   "bp.s2"   "bp.s3"   "bp.s4"   "bp.s5"   "bp.s6"
## [50] "s1.s2"   "s1.s3"   "s1.s4"   "s1.s5"   "s1.s6"   "s2.s3"   "s2.s4"
## [57] "s2.s5"   "s2.s6"   "s3.s4"   "s3.s5"   "s3.s6"   "s4.s5"   "s4.s6"
## [64] "s5.s6"
```

```
dim(df_train[-10])
```

```
## [1] 265  10
```

```
dim(train_sc_pf2)
```

```
## [1] 265  64
```

- Prepare the validation and test sets

```
df_val[-11]= scale(df_val[-11])
val_sc_pf2 <- pf2_transform(df_val,target_name = "target")
val_sc_pf2$target = df_val$target
val_sc_pf2 = val_sc_pf2[-20]


df_test[-11]= scale(df_test[-11])
test_sc_pf2 <- pf2_transform(df_test,target_name = "target")
test_sc_pf2$target = df_test$target
test_sc_pf2 = test_sc_pf2[-20]
```

### Run the Polynomial Regression

```
lr = caret::train(target ~ ., method='lm',data = train_sc_pf2)
train_pred = predict(lr,newdata = train_sc_pf2)
val_pred = predict(lr,newdata = val_sc_pf2)
test_pred = predict(lr,newdata = test_sc_pf2)
```

### Evaluate the results

```
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---:|
| Train | 45.13 |
| Validation | 288.66 |
| Test | 231.93 |

# Ridge Regression

## Basic Model

```
ridge <- caret::train(y = train_sc_pf2$target,x = train_sc_pf2[-10],
                 method = 'glmnet',
                 tuneGrid = expand.grid(alpha = 0, lambda = 1)
               )
train_pred = predict(ridge,newdata = train_sc_pf2)
val_pred = predict(ridge,newdata = val_sc_pf2)
test_pred = predict(ridge,newdata = test_sc_pf2)

res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---:|
| Train | 46.67 |
| Validation | 59.98 |
| Test | 58.11 |

## Hyper-parameter tuning

```
parameters <- c(seq(0.1, 2, by =0.1) ,  seq(2, 5, 0.5) , seq(5, 25, 1))

ridge<-caret::train(y = train_sc_pf2$target,
                 x = train_sc_pf2[-10],
                 method = 'glmnet',
                 tuneGrid = expand.grid(alpha = 0, lambda = parameters) ,
                 metric =  "Rsquared"
               )
paste(" Ridge Best lambda = ",ridge$finalModel$lambdaOpt)
```

```
## [1] " Ridge Best lambda =  25"
```

## Run the Optimized Ridge Regression

```r
ridge_best<-caret::train(y = train_sc_pf2$target,
      x = train_sc_pf2[-10],
      method = 'glmnet',
      tuneGrid = expand.grid(alpha = 0, lambda = ridge$finalModel$lambdaOpt))
train_pred = predict(ridge_best,newdata = train_sc_pf2)
val_pred = predict(ridge_best,newdata = val_sc_pf2)
test_pred = predict(ridge_best,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---:|
| Train | 48.93 |
| Validation | 57.66 |
| Test | 56.02 |

### Evaluate Performance of the optimized Ridge Regression

```r
res = data.frame()
y = R2(train_pred,df_train$target)
res = rbind(res,c("Train",y))
y = R2(val_pred,df_val$target)
res = rbind(res,c("Validation",y))
y = R2(test_pred,df_test$target)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","R_2")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | R_2 |
|---|---:|
| Train | 0.6295 |
| Validation | 0.3788 |
| Test | 0.4761 |

# Lasso Regression

## Basic Model

```
lasso<-caret::train(y = train_sc_pf2$target,
      x = train_sc_pf2[-10],
      method = 'glmnet',
      tuneGrid = expand.grid(alpha = 1, lambda = 0))
train_pred = predict(lasso,newdata = train_sc_pf2)
val_pred = predict(lasso,newdata = val_sc_pf2)
test_pred = predict(lasso,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---|
| Train | 45.51 |
| Validation | 63.96 |
| Test | 59.27 |

## Hyper-parameter tuning

- Find the best hyper parameter lambda

```
parameters <- c(seq(0.1, 2, by =0.1) ,  seq(2, 5, 0.5) , seq(5, 25, 1))
lasso<-caret::train(y = train_sc_pf2$target,
                x = train_sc_pf2[-10],
                method = 'glmnet',
                tuneGrid = expand.grid(alpha = 1, lambda = parameters) ,
                metric =  "Rsquared"
              )
paste(" Lasso Best lambda = ",lasso$finalModel$lambdaOpt)
```

```
## [1] " Lasso Best lambda =  5"
```

## Run the Optimized Lasso Regression

```r
lasso_best<-caret::train(y = train_sc_pf2$target,
      x = train_sc_pf2[-10],
      method = 'glmnet',
      tuneGrid = expand.grid(alpha = 1, lambda = lasso$finalModel$lambdaOpt))
train_pred = predict(lasso_best,newdata = train_sc_pf2)
val_pred = predict(lasso_best,newdata = val_sc_pf2)
test_pred = predict(lasso_best,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---|
| Train | 52.57 |
| Validation | 58.00 |
| Test | 53.24 |

### Performance of the Optimized Lasso Regression

```r
res = data.frame()
y = R2(train_pred,df_train$target)
res = rbind(res,c("Train",y))
y = R2(val_pred,df_val$target)
res = rbind(res,c("Validation",y))
y = R2(test_pred,df_test$target)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","R_2")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | R_2 |
|---|---|
| Train | 0.5673 |
| Validation | 0.3646 |
| Test | 0.5117 |

### Determine non-zero coefficients

```
df = data.frame(
  lasso = as.data.frame.matrix(coef(lasso$finalModel, lasso$finalModel$lambdaOpt))
)
df =subset(df, s1>0.1)
df$var = row.names(df)
knitr::kable(df[order(-df$s1),c(2,1)][2])
```

| | s1 |
|---|---:|
| (Intercept) | 154.3001 |
| bp | 25.3282 |
| s6 | 20.6368 |
| s1 | 13.4184 |
| bmi.s6 | 3.9376 |
| sex.s3 | 2.2168 |
| age.s6 | 0.7353 |
| bmi.bp | 0.4412 |
| age.sex | 0.1728 |
| s6.2 | 0.1407 |

## Neural Network

### Large network

```
mlp <- caret::train(target ~ .,data = train_sc_pf2,method='mlp',size=1000)

train_pred = predict(mlp,newdata = train_sc_pf2)
val_pred = predict(mlp,newdata = val_sc_pf2)
test_pred = predict(mlp,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---:|
| Train | 104.5 |
| Validation | 103.7 |

| Data | RMSE |
|---|---|
| Test | 107.5 |

### Smaller network

```
mlp <- caret::train(target ~ .,data = train_sc_pf2,method='mlp',size=50)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
train_pred = predict(mlp,newdata = train_sc_pf2)
val_pred = predict(mlp,newdata = val_sc_pf2)
test_pred = predict(mlp,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---|
| Train | 83.34 |
| Validation | 74.52 |
| Test | 77.62 |

## Regression Tree

### Basic Tree

```
dtr <- caret::train(target ~ .,data = train_sc_pf2,method='rpart')

train_pred = predict(dtr,newdata = train_sc_pf2)
val_pred = predict(dtr,newdata = val_sc_pf2)
test_pred = predict(dtr,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
```

```r
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---|
| Train | 60.28 |
| Validation | 66.73 |
| Test | 60.40 |

### Tree with a maximum depth of 2

```r
dtr <- caret::train(target ~ .,data = train_sc_pf2,method='rpart',
                    control = list(max_depth=2))
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```r
train_pred = predict(dtr,newdata = train_sc_pf2)
val_pred = predict(dtr,newdata = val_sc_pf2)
test_pred = predict(dtr,newdata = test_sc_pf2)
res = data.frame()
y = rmse(df_train$target,train_pred)
res = rbind(res,c("Train",y))
y = rmse(df_val$target,val_pred)
res = rbind(res,c("Validation",y))
y = rmse(df_test$target,test_pred)
res = rbind(res,c("Test",y))
colnames(res) = c("Data","RMSE")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Data | RMSE |
|---|---|
| Train | 60.28 |
| Validation | 66.73 |
| Test | 60.40 |

## Harder problem

### Data preparation

- Read data

```
df <- read.csv("ENB2012_data.csv")
```

- Create training, validation and test datasets.

```
dataset = df[-c(11,12)]
N = nrow(dataset)
cut1 = floor(0.6*N)
cut2 = floor(0.8*N)
index = sample(1:N)
train_index = index[1:cut1]
val_index = index[(cut1+1):cut2]
test_index = index[(cut2+1):N]
df_train = dataset[train_index,]
df_val = dataset[val_index,]
df_test = dataset[test_index,]
sapply(list(df_train,df_val,df_test),nrow)
```

```
## [1] 460 154 154
```

```
df_train[-10] = scale(df_train[-10])
```

### Build and Evaluate Models

```
res = data.frame()
# Linear Regression
lr = caret::train(Y2 ~ ., method='lm',data = df_train)
train_pred = predict(lr,newdata = df_train)
val_pred = predict(lr,newdata = scale(df_val[-10]))
test_pred = predict(lr,newdata = scale(df_test[-10]))
y1 = rmse(df_train$Y2,train_pred)
y2 = rmse(df_val$Y2,val_pred)
y3 = rmse(df_test$Y2,test_pred)
res = rbind(res,c("Linear Regression",y1,y2,y3))
# Ridge Regression
ridge <- caret::train(y = df_train$Y2,x = df_train[-10],
                  method = 'glmnet',
                  tuneGrid = expand.grid(alpha = 0, lambda = 1)
                )
train_pred = predict(ridge,newdata = df_train)
val_pred = predict(ridge,newdata = scale(df_val[-10]))
test_pred = predict(ridge,newdata = scale(df_test[-10]))
y1 = rmse(df_train$Y2,train_pred)
y2 = rmse(df_val$Y2,val_pred)
```

```r
y3 = rmse(df_test$Y2,test_pred)
res = rbind(res,c("Ridge Regression",y1,y2,y3))
# Lasso Regression
lasso <- caret::train(y = df_train$Y2,x = df_train[-10],
                 method = 'glmnet',
                 tuneGrid = expand.grid(alpha = 1, lambda = 0)
               )
train_pred = predict(lasso,newdata = df_train)
val_pred = predict(lasso,newdata = scale(df_val[-10]))
test_pred = predict(lasso,newdata = scale(df_test[-10]))
y1 = rmse(df_train$Y2,train_pred)
y2 = rmse(df_val$Y2,val_pred)
y3 = rmse(df_test$Y2,test_pred)
res = rbind(res,c("Lasso Regression",y1,y2,y3))
# Neural Net
nn <- caret::train(y = df_train$Y2,x = df_train[-10],
                 method = 'mlp')
train_pred = predict(nn,newdata = df_train)
val_pred = predict(nn,newdata = scale(df_val[-10]))
test_pred = predict(nn,newdata = scale(df_test[-10]))
y1 = rmse(df_train$Y2,train_pred)
y2 = rmse(df_val$Y2,val_pred)
y3 = rmse(df_test$Y2,test_pred)
res = rbind(res,c("Neural Net",y1,y2,y3))
# Regression Tree
dt <- caret::train(y = df_train$Y2,x = df_train[-10],
                 method = 'rpart')
train_pred = predict(dt,newdata = df_train)
val_pred = predict(dt,newdata = scale(df_val[-10]))
test_pred = predict(dt,newdata = scale(df_test[-10]))
y1 = rmse(df_train$Y2,train_pred)
y2 = rmse(df_val$Y2,val_pred)
y3 = rmse(df_test$Y2,test_pred)
res = rbind(res,c("Regression Tree",y1,y2,y3))

colnames(res) = c("Model","Train","Validation","Test")
# Following converts appropriate columns to numeric type
res[-1] = lapply(res[-1], FUN = function(y){as.numeric(y)})
# Following rounds numeric values to 4 digits
res[,sapply(res, is.numeric)] <-round(res[,sapply(res, is.numeric)],4)
knitr::kable(res)
```

| Model | Train | Validation | Test |
|---|---|---|---|
| Linear Regression | 1.901 | 2.014 | 2.026 |
| Ridge Regression | 2.223 | 2.453 | 2.481 |
| Lasso Regression | 1.903 | 2.015 | 2.022 |
| Neural Net | 2.506 | 2.529 | 2.574 |
| Regression Tree | 2.989 | 3.219 | 3.325 |