Lista 3 - Mineração

Victor Alves Dogo Martins, RA: 744878 Ana Beatriz Alves Monteiro, RA: 727838
Larissa Torres, RA: 631914

14-08-2022

Itens 1 e 2

Para a transformação das variáveis categóricas (`US`, `Urban` e `ShelveLoc`) em dummies, utilizamos a função `model.matrix()`, que as transforma automacitamente e são apresentadas da seguinte maneira:

- **US:** transformada na variável X_{USYes}:

$$X_{USYes} = \begin{cases} 1, \text{caso a loja estiver localizada nos EUA;} \\ 0, \text{caso contrário.} \end{cases}$$

- **Urban:** transformada na variável $X_{UrbanYes}$:

$$X_{UrbanYes} = \begin{cases} 1, \text{caso a loja estiver localizada na zona urbana;} \\ 0, \text{caso contrário.} \end{cases}$$

- **ShelveLoc:** transformada nas variáveis $X_{ShelveLocGood}$ $X_{ShelveLocMedium}$:

$$X_{ShelveLocGood} = \begin{cases} 1, \text{caso o produto tiver localização boa na prateleira;} \\ 0, \text{caso contrário.} \end{cases}$$

$$X_{ShelveLocMedium} = \begin{cases} 1, \text{caso o produto tiver localização média (mas não boa) na prateleira;} \\ 0, \text{caso contrário.} \end{cases}$$

Por outro lado, como feito na Lista 2, a divisão do banco de dados entre treino e teste foi feita com o auxílio da função `initial_split()` do pacote `{rsample}`:

```
## Lendo pacotes

set.seed(1)

library(progress)
library(tidyverse)
library(rsample)
library(knitr)
```

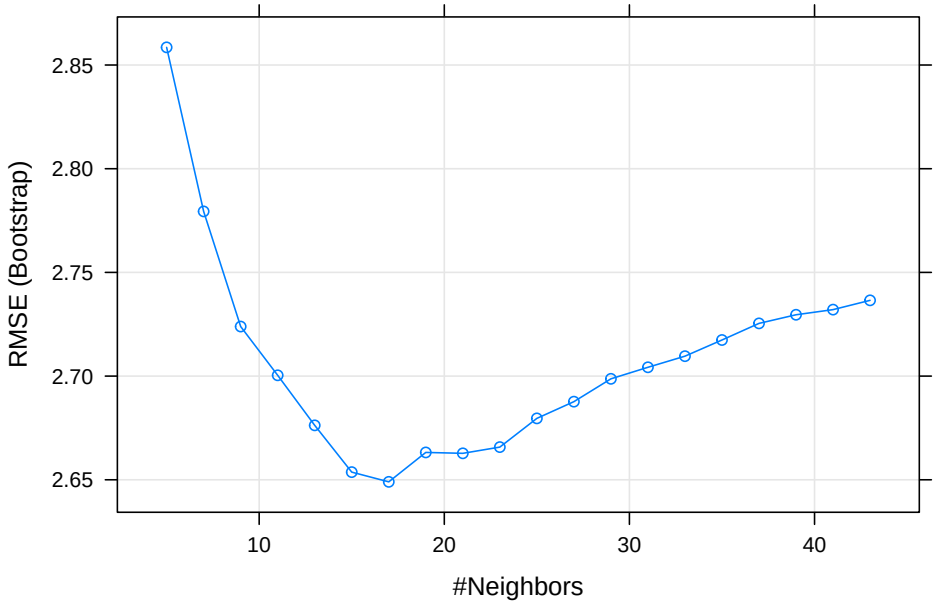
```r
library(kableExtra)
library(caret)
library(FNN)
library(rpart)
library(rpart.plot)
library(randomForest)
library(locfit)

## Definindo funcao de risco (utilizada na lista 2)

funcao_risco <- function(y_pred, y_obs){

  w <- (y_pred-y_obs)^2
  sigma <- var(w)
  risco <- mean(w)
  liminf <- risco - (2*sqrt((1/length(w))*sigma))
  limsup <- risco + (2*sqrt((1/length(w))*sigma))

  return(data.frame(risco, liminf, limsup))
}

## Lendo Dados

df <- ISLR::Carseats |>
  mutate(US=as.factor(US),
         Urban=as.factor(Urban),
         ShelveLoc=as.factor(ShelveLoc))

## Divisão entre treino e teste

split <- initial_split(df, prop=0.6)

tre <- training(split)
tes <- testing(split)

x_tre <- model.matrix(Sales~., tre)
y_tre <- tre[,1]

x_tes <- model.matrix(Sales~., tes)
y_tes <- tes[,1]
```

Item 3

KNN

```r
### ITEM 3

## KNN

# Realizando calculo do melhor K
```

```
ajuste_knn <- train(
  x=x_tre,
  y=y_tre,
  method = 'knn',
  tuneLength = 20
)

# Plotando grafico de K vs Risco

plot(ajuste_knn)
```




```
paste0('O melhor K é: ', ajuste_knn$bestTune)
```

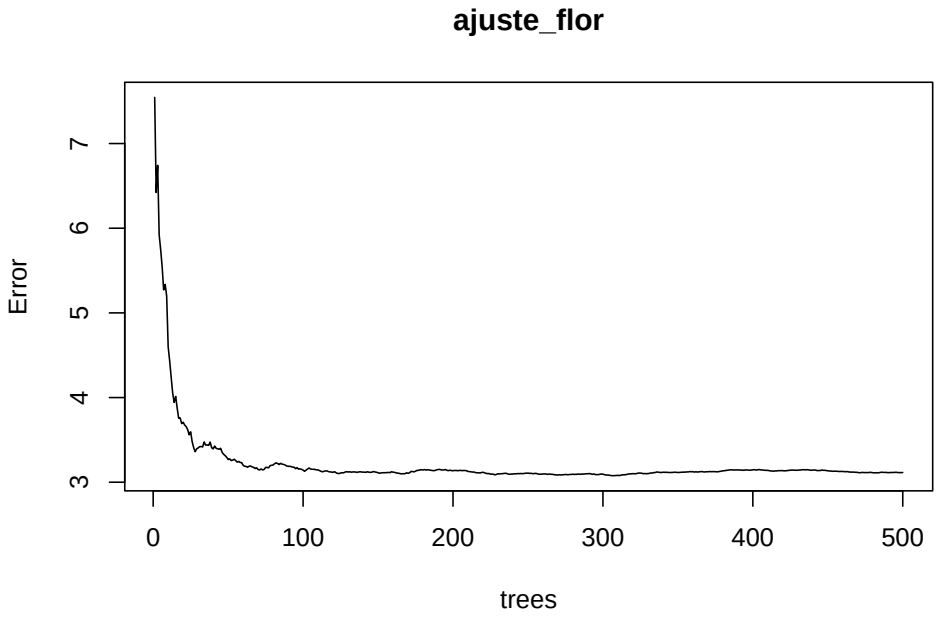
```
## [1] "O melhor K é: 17"
```

```
# Realizando Ajuste

ajuste_knn <- knn.reg(train=x_tre, test=x_tes, y=y_tre, k=17)
```

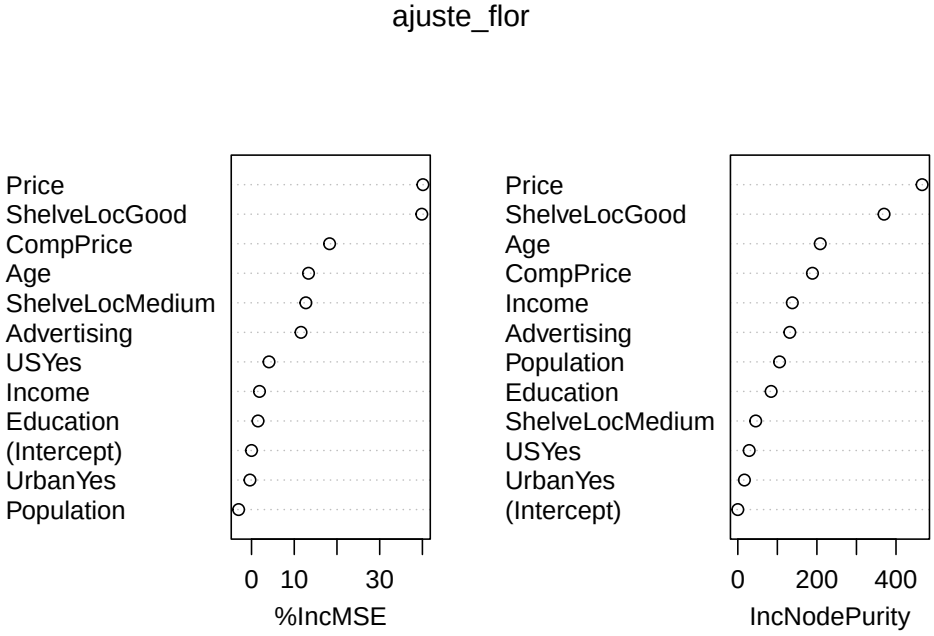
Floresta Aleatória

```
## Floresta Aleatória

# Realizando Ajuste

ajuste_flor <- randomForest(x=x_tre, y=y_tre, mtry = round(ncol(df)/3),
                            importance=TRUE)
```

```r
# Ajuste x Numero de Arvores

plot(ajuste_flor)
```

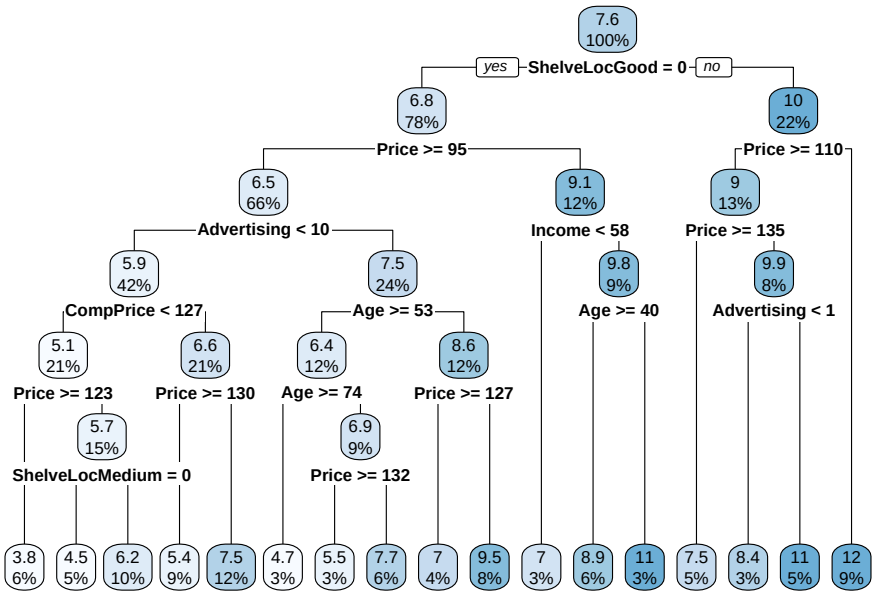



```r
# Importancia

varImpPlot(ajuste_flor)
```

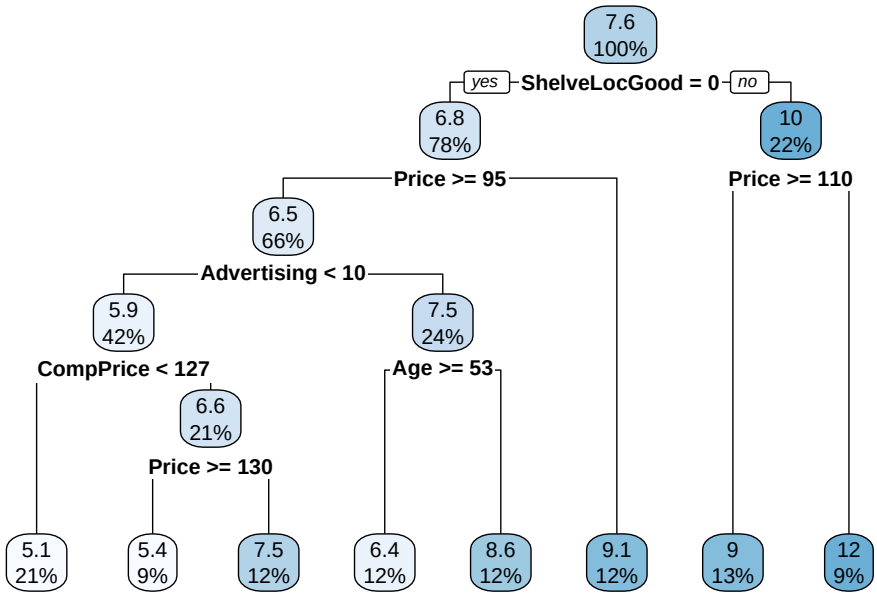


Árvore de Regressão

```
## Árvore de Regressão

tre_arv <- data.frame(y_tre, x_tre[,-1])

# Ajustando para melhor cp automatico

ajuste_arv <- rpart(y_tre~., data=tre_arv, method='anova')
melhor_cp <- ajuste_arv$cptable[which.min(ajuste_arv$cptable[,'xerror']),'CP']
ajuste_arv <- prune(ajuste_arv, cp=melhor_cp)

rpart.plot(ajuste_arv)
```




```
# Ajustando para melhor cp na mão

ajuste_arv <- rpart(y_tre~., data=tre_arv, method='anova')
ajuste_arv <- prune(ajuste_arv, cp=0.027)

rpart.plot(ajuste_arv)
```



Nadaraya-Watson

```
## Nadaraya-Watson

# Função de Ajuste do NW

nw_func <- function(x_tes, x_tre, y, h){

  # Normalizando covariaveis de treino e teste

  x_tes <- scale(x_tes)
  x_tre <- scale(x_tre)

  # Definindo Kernel Gaussiano

  k <- function(h,d){(1/(h*sqrt(2*pi)))*exp(-0.5* (d/h)^2)}

  # Calculando matriz de distancia euclidiana (linha=i-esima obs. de teste,
                                               #coluna=j-esima obs. de treino)

  dis <- fields::rdist(x_tes[,-1], x_tre[,-1])

  # Definindo vetor vazio para y predito

  y_pred <- numeric(ncol(x_tes))

  # Para cada i-esima linha do conjunto de teste...

  for (i in 1:nrow(dis)) {
```

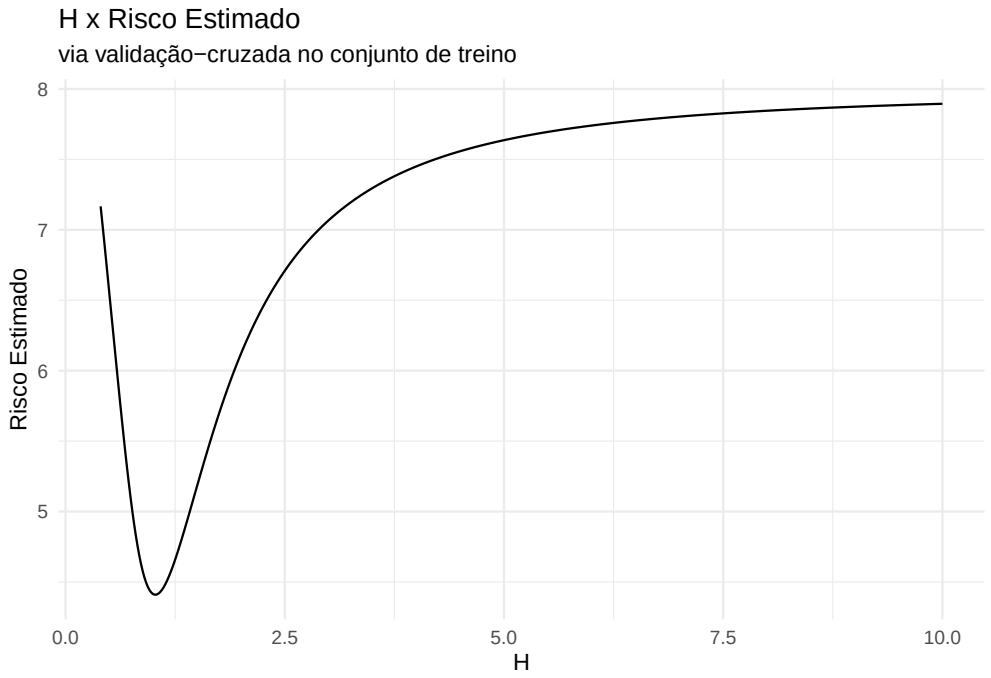
```
    # Criando vetor vazio para resultado dos kernels

    kx <- numeric(ncol(dis))

    # Para cada j-esima linha do conjunto de treino...

    for (j in 1:ncol(dis)) {

      # Calculando j-esimo kernel com base em h e distancia entre i-esima obs do teste e
      # j-esima obs do treino

      kx[j] <- k(h, dis[i,j])

    }

    # Calculando vetor de pesos

    wx <- kx/sum(kx)

    # Calculando i-esimo y predito

    y_pred[i] <- sum(wx*y)

  }

  # Retornando y preditos

  return(y_pred)

}
```

```
# Funcao de melhor h via validacao cruzada no treino

melhor_h_nw <- function(x_tre, y_tre, seed=1){

  set.seed(seed)

  # Embaralhando dados

  df <- data.frame(y_tre,x_tre)
  df <- df[sample(1:nrow(df)),]

  # Definindo kfolds com k=5

  size <- round(nrow(df)/5)

  # Lista com cada fold

  kfoldlist <- list(
    df[1:size,],
    df[(size+1):(2*size),],
    df[(2*size+1):(3*size),],
    df[(3*size+1):(4*size),],
```

```
  df[(4*size+1):(5*size),]
)

# Definindo vetor de h para ser testado

h <- seq(0.4,10,0.02)

# Dataframe com resultados

result <- data.frame(
  h=h,
  risco=numeric(length(h))
)

# Barra de progresso

pb <- progress_bar$new(
  total=length(h),
  format = "[:bar] :percent eta: :eta  elapsed: :elapsed")

for (jj in 1:length(h)) {

  hresult <- numeric(5)

  for (ii in 1:5) {

    # Definindo conjunto de treino e teste para i-esima iteracao

    df_tre <- do.call(rbind.data.frame, kfoldlist[-ii])

    df_tes <- kfoldlist[[ii]]

    # Ajustando

    ypred <- nw_func(x_tes=df_tes[,-1],
                     x_tre=df_tre[,-1],
                     y=df_tre[,1],
                     h=h[jj])

    # Calculando risco

    hresult[ii] <- funcao_risco(y_pred=ypred, y_obs=df_tes[,1])$risco[1]

  }

  # Risco medio do j-esimo h

  result$risco[jj] <- mean(hresult)

  pb$tick()
  Sys.sleep(1 / length(h))

}
```

```
  # Retornando data.frame com resultados

  return(result)

}
```

```
# Calculando risco para cada H

resultados_h <- melhor_h_nw(x_tre=x_tre,y_tre=y_tre)

resultados_h |>
  ggplot()+
  aes(x=h,y=risco)+
  geom_line()+
  labs(x='H', y='Risco Estimado',
       title='H x Risco Estimado',
       subtitle = 'via validação-cruzada no conjunto de treino')+
  theme_minimal()
```




```
# Encontrando melhor H

paste0('O melhor H é: ', resultados_h[resultados_h$risco==min(resultados_h$risco),][1])
```

```
## [1] "O melhor H é: 1.02"
```

```
# Realizando ajuste Nadaraya-Watson com melhor H

ajuste_nw <- nw_func(x_tes, x_tre, y_tre,
                     h=1.02)
```

Item 4

```
### ITEM 4

funcao_risco <- function(y_pred, y_obs){

  w <- (y_pred-y_obs)^2
  sigma <- var(w)
  risco <- mean(w)
  liminf <- risco - (2*sqrt((1/length(w))*sigma))
  limsup <- risco + (2*sqrt((1/length(w))*sigma))

  return(data.frame(risco, liminf, limsup))
}

# Apresentando riscos e ICs com 95% de confiança para cada ajuste

tibble(
  `Variável`=c("Risco Estimado", "Limite Inferior", "Limite Superior"),
  `Nadaraya-Watson`=as.numeric(funcao_risco(ajuste_nw, y_tes)),
  `KNN`=as.numeric(funcao_risco(ajuste_knn$pred, y_tes)),
  `Floresta Aleatória`=as.numeric(funcao_risco(predict(ajuste_flor, x_tes), y_tes)),
  `Árvore de Regressão`=as.numeric(funcao_risco(predict(ajuste_arv,
                                                        data.frame(x_tes)), y_tes))
) |>
  kable('latex', align='cccc',
        caption = 'Risco e Intervalos de Confiança para Ajustes') |>
  kable_styling(position="center",
                latex_options="HOLD_position")
```

Table 1: Risco e Intervalos de Confiança para Ajustes

Variável	Nadaraya-Watson	KNN	Floresta Aleatória	Árvore de Regressão
Risco Estimado	3.928357	7.348200	3.050677	4.863114
Limite Inferior	3.030916	5.756560	2.221421	3.764944
Limite Superior	4.825798	8.939839	3.879933	5.961284

Dentre os métodos ajustados nesta lista, o que apresentou melhor desempenho foi o ajuste via Florestas Aleatórias, com risco estimado de 3.05, com 95% de confiança que o risco esteja entre 2.22 e 3.87 (assim, apresentando a menor amplitude intervalar). Após ele, o ajuste via Nadaraya-Watson apresentou desempenho relativamente bom, com risco estimado de 3.92 e 95% de confiança de que ele está entre 3.03 e 4.82.

Se comparados aos métodos da lista anterior, apresentam desempenho inferior (MQ e Lasso demonstraram risco estimado de aproximadamente 1.2). Isso se dá pelo fato de estarmos tratando, nesta lista, de métodos menos específicos para a regressão (comumente utilizados para classificação). Por outro lado, ajustes via Mínimos Quadrados e com penalização via Lasso são mais adequados para estes fins e que, no geral, retornam melhores predições no contexto de regressão.

Item 5

Item 6