Cross-Validation, Bootstrap & Bagging

Baseado em: “*An introduction to statistical learning with applications in R*”

Objetivo:

Os métodos de cross-validation visam reduzir a diferença entre a taxa de erro do conjunto de teste de um determinado modelo estatístico comparado com o conjunto de treinamento, especialmente em casos onde não há muitas observações com quem trabalhar.

A aplicação desses métodos é usada para verificar a eficiência/taxa de erro/acerto de um modelo estatístico já pronto e não está envolvida na criação dos modelos em si.

Considerações:

- Dataset usado: Diabetes prediction
- Diferença nos modelos estatísticos usados no livro e os usados neste estudo
- Bibliotecas do R Studio usadas:

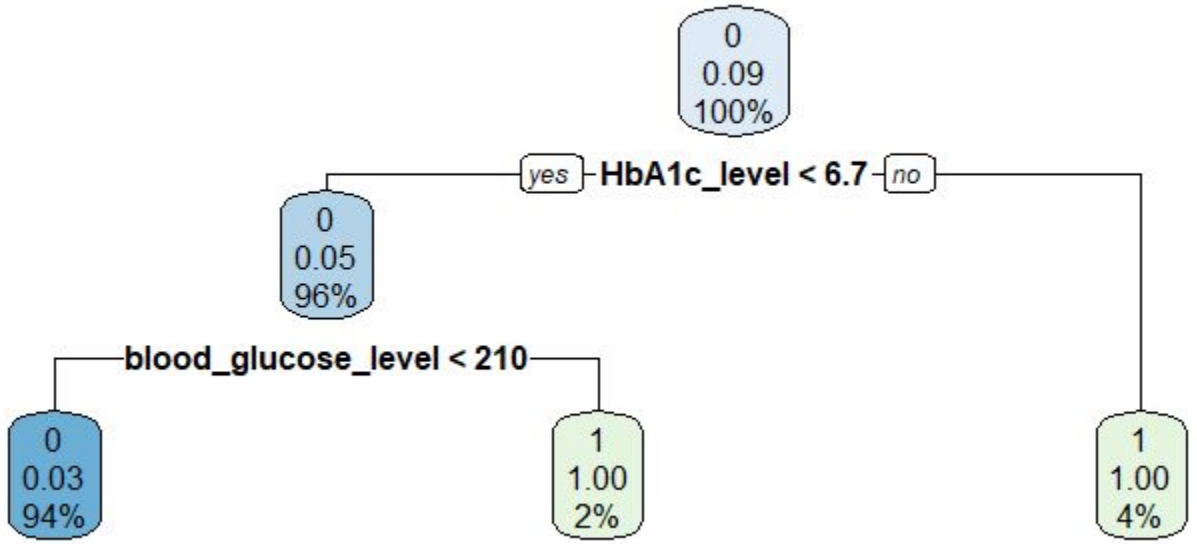
```
library(tidyverse)
library(rpart)
library(rpart.plot)
```

- Taxa de erro/acerto real

```
modelo <- rpart(diabetes ~ .,data = dados,
                method="class",control=rpart.control(cp=0))
sum(predict(modelo,dados,"class")==dados[,9])
```

R studio

Considerações: Modelo de Previsão



R studio

Método: *Validation Set*

Consiste na partição do conjunto de dados em dois subconjuntos, treino e teste, selecionando aleatoriamente os elementos que os compõem.

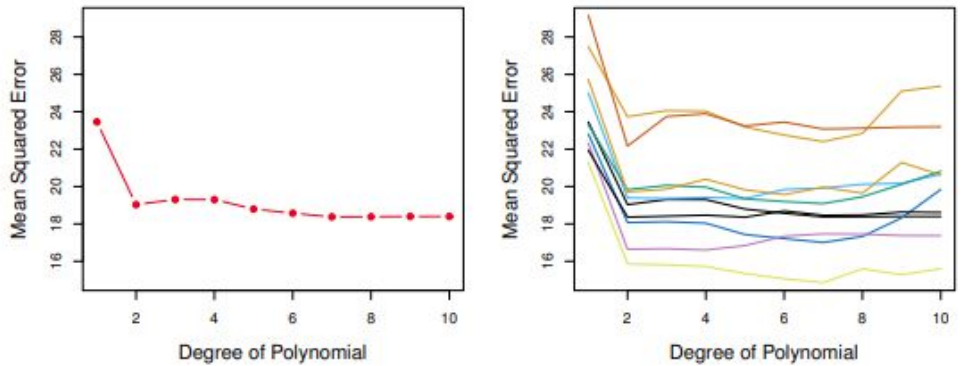
ISRLv2 - sessão 5.1.1

```
ran <- sample(1:nrow(dados), 0.75 * nrow(dados))
treino <- dados[ran,]
teste <- dados[-ran,]
```

R studio

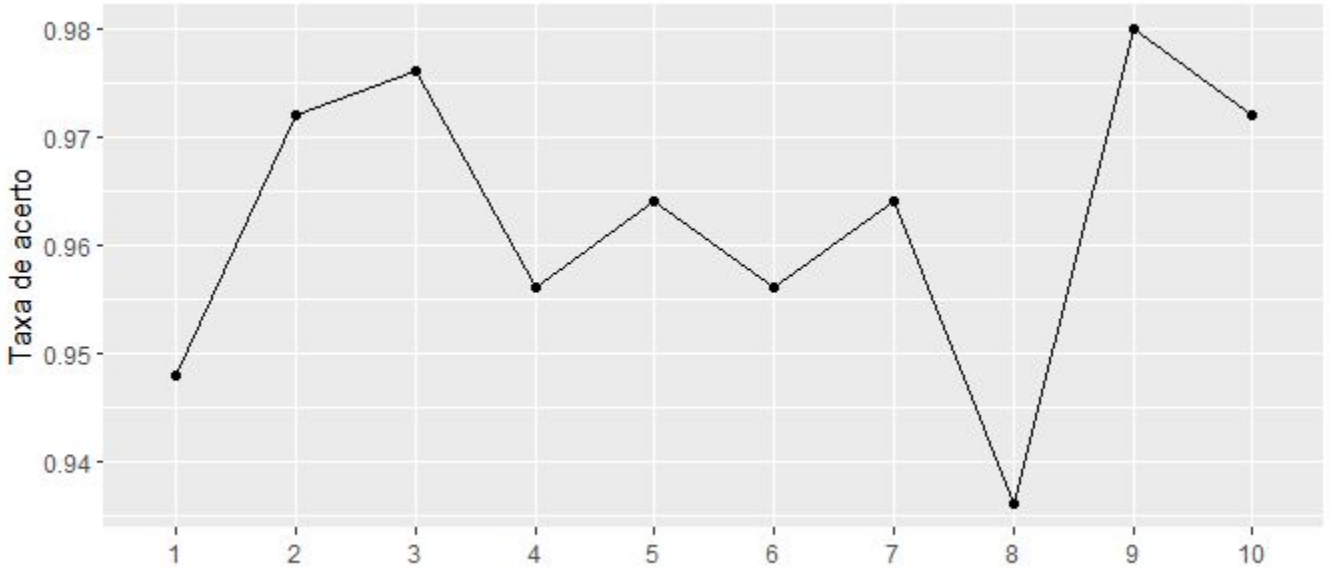
Método: *Validation Set*

Devido a natureza dos modelos estatísticos, especialmente quando há poucas observações de treino, o método de Validation Set é extremamente suscetível a variações no conjunto de treinamento e pode acabar gerando resultados inconsistentes.



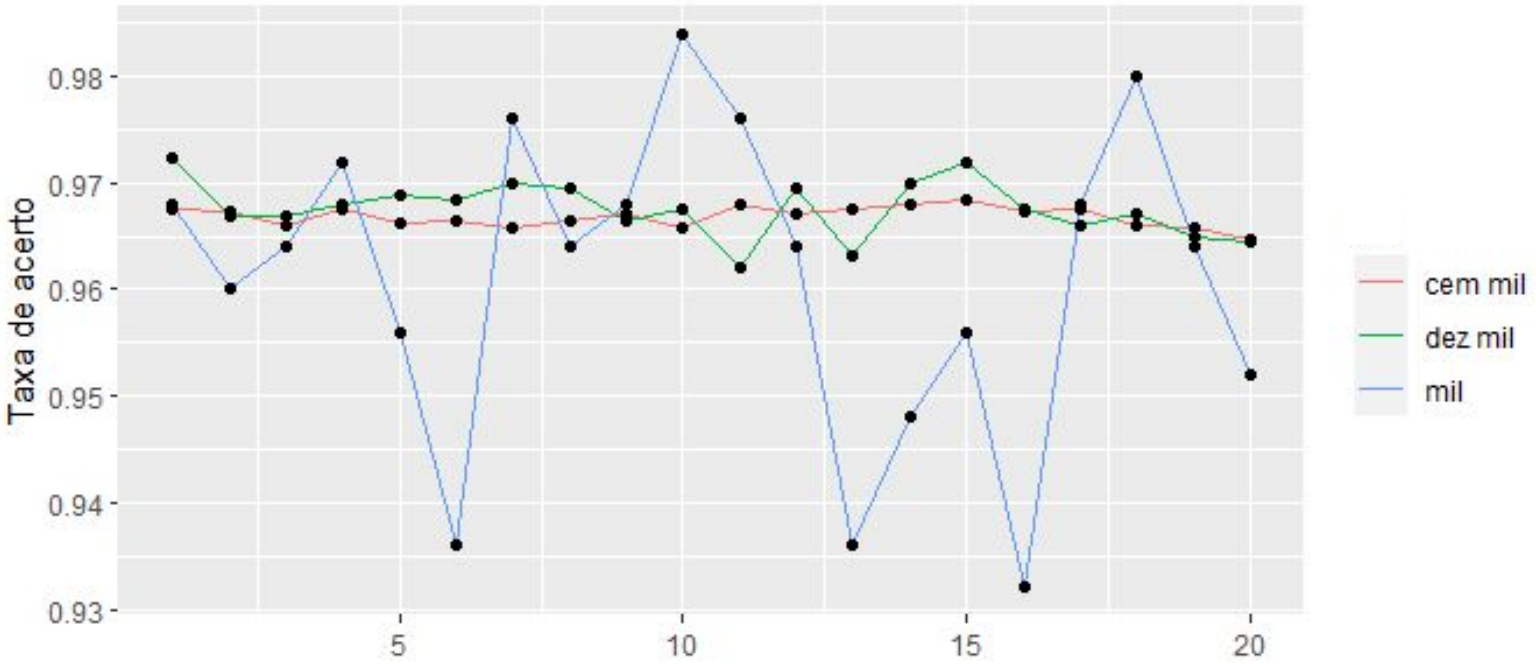
ISRLv2 - sessão 5.1.1

Método: *Validation Set*



R studio

Método: *Validation Set*



R studio

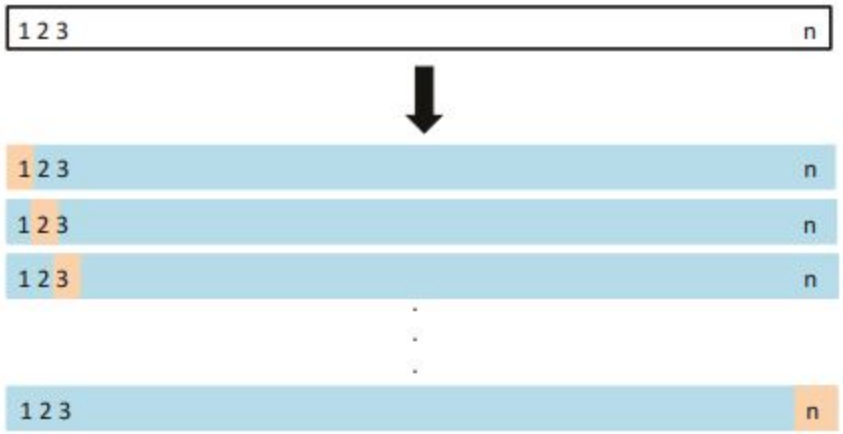
Método: *Validation Set*

```r
dados <- dados[c(1:1000),]
vet <- c()

for(i in 1:10){
  ran <- sample(1:nrow(dados), 0.75 * nrow(dados))
  treino <- dados[ran,]
  teste <- dados[-ran,]
  modelo <- rpart(diabetes ~ .,data = treino,method="class",
                  control=rpart.control(cp=0))
  vet[i] <- sum(predict(modelo,teste,"class")==teste[,9])/(nrow(teste))
}
```

R studio

Método: *Leave-One-Out Cross-Validation*

Reitera na ideia de particionar o conjunto em treino e teste, porém o subconjunto de teste consiste de uma única observação, e o subconjunto de treino são as demais observações. O modelo é treinado e a previsão é realizada para todas as possíveis partições do conjunto original.



Método: *Leave-One-Out Cross-Validation*

A taxa de sucesso do modelo é dada pela média das *n* taxas de sucesso encontradas.

```
dados <- dados1[c(1:1000),]
acc <- 0

for(i in 1:nrow(dados)){
  treino <- dados[-i,]
  teste <- dados[i,]
  modelo <- rpart(diabetes ~ .,data = treino,method="class",
                  control=rpart.control(cp=0))
  acc <- acc + sum(predict(modelo,teste,"class")==teste[,9])
}

acc/nrow(dados)
```

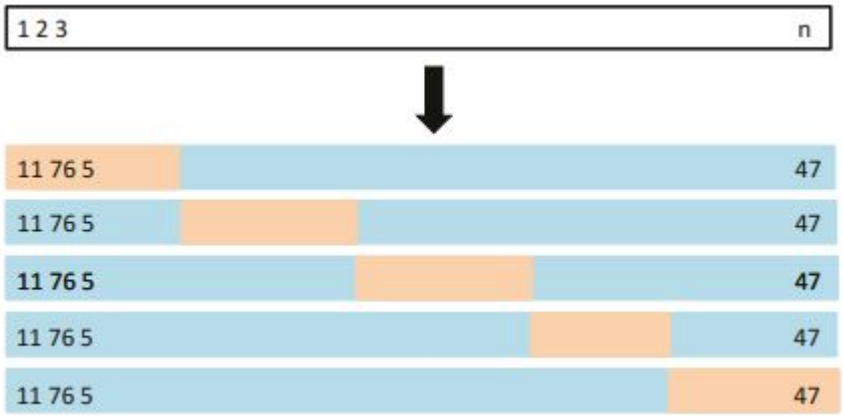
```
> acc/nrow(dados)
[1] 0.964
```

R Studio

Método: *K-fold Cross-Validation*

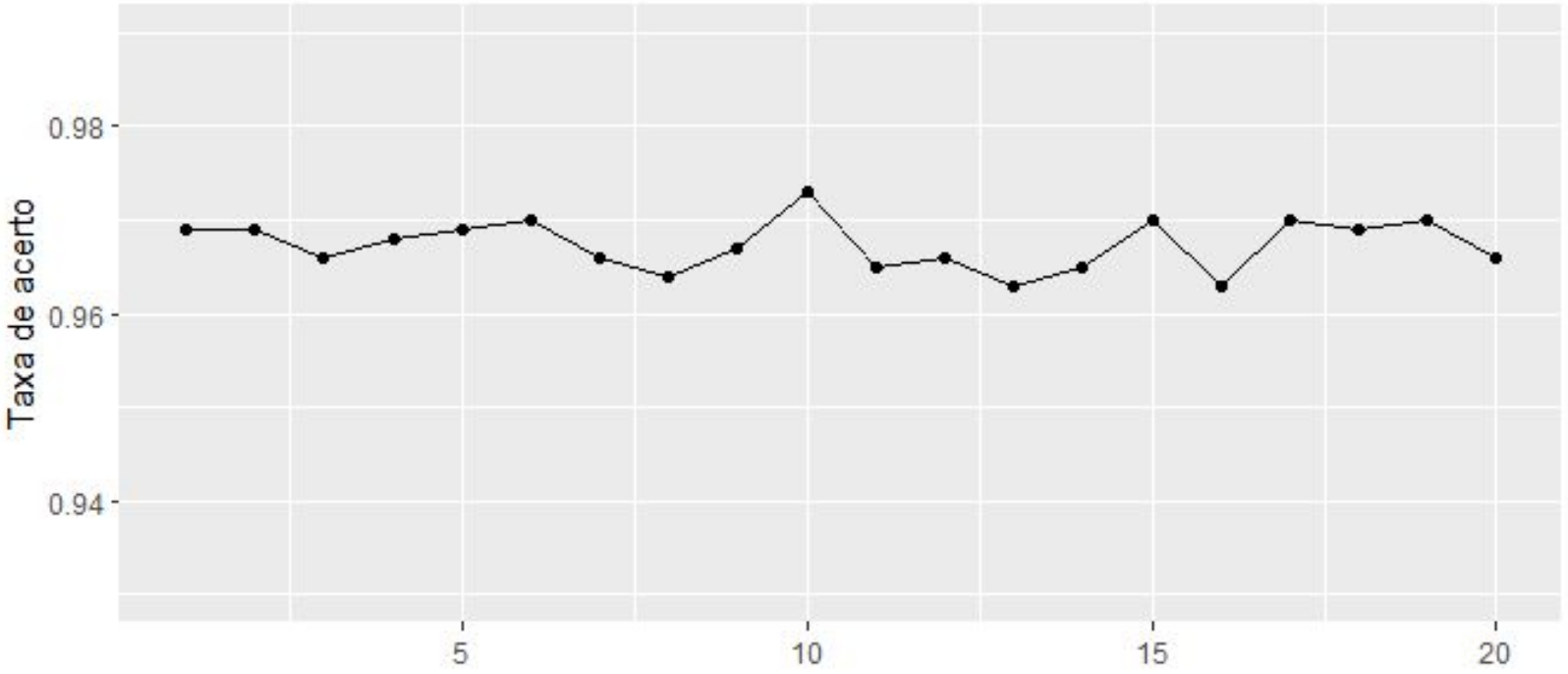
Reitera nos métodos anteriores, especialmente no método de *LOOCV*, porém o conjunto de dados é particionado em *k* subconjuntos com elementos aleatórios.

Um subconjunto é escolhido como conjunto de teste e os demais compõe o conjunto de treino, o modelo é aplicado, os resultados são registrados e o processo se repete para os demais *k-1* subconjuntos. A taxa de acerto final é a média das *k* taxas de acerto.



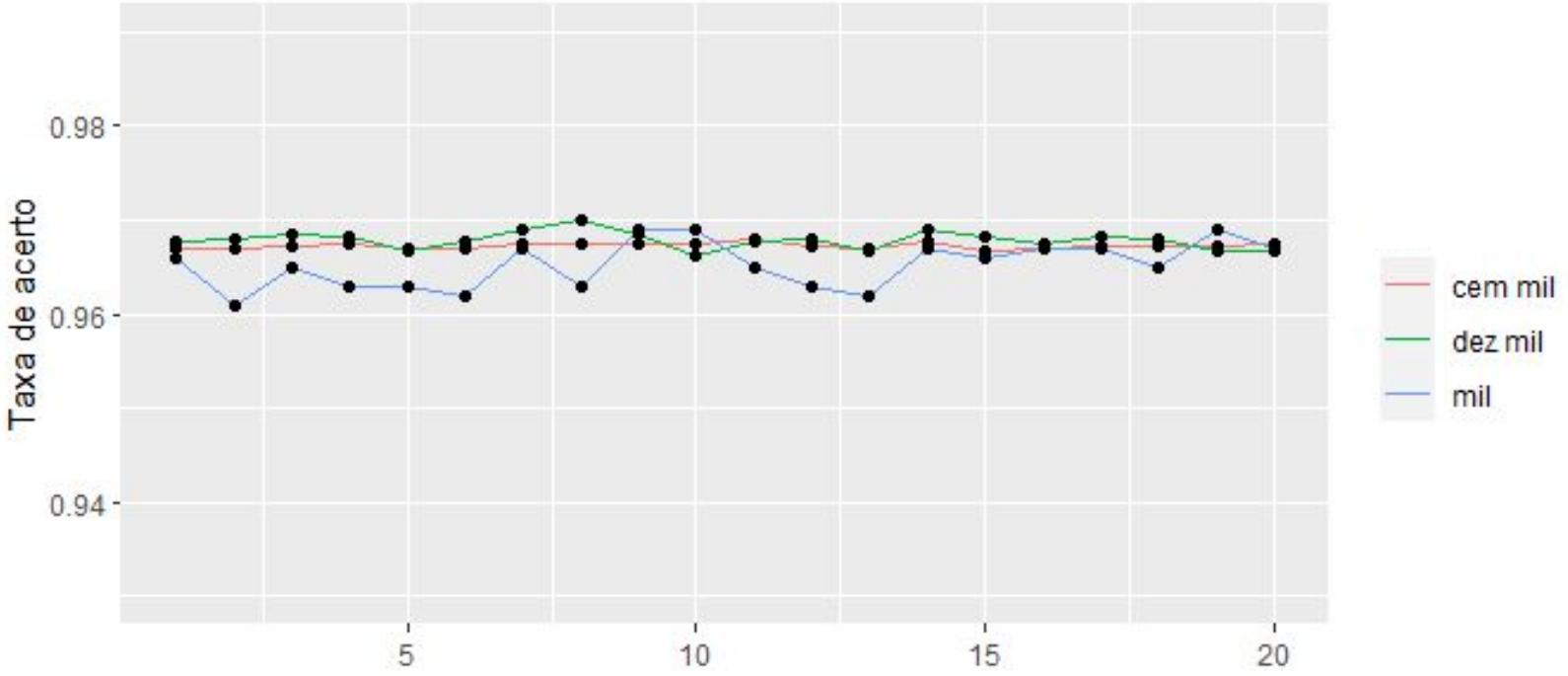
ISRLv2 - sessão 5.1.3

Método: *K-fold Cross-Validation*



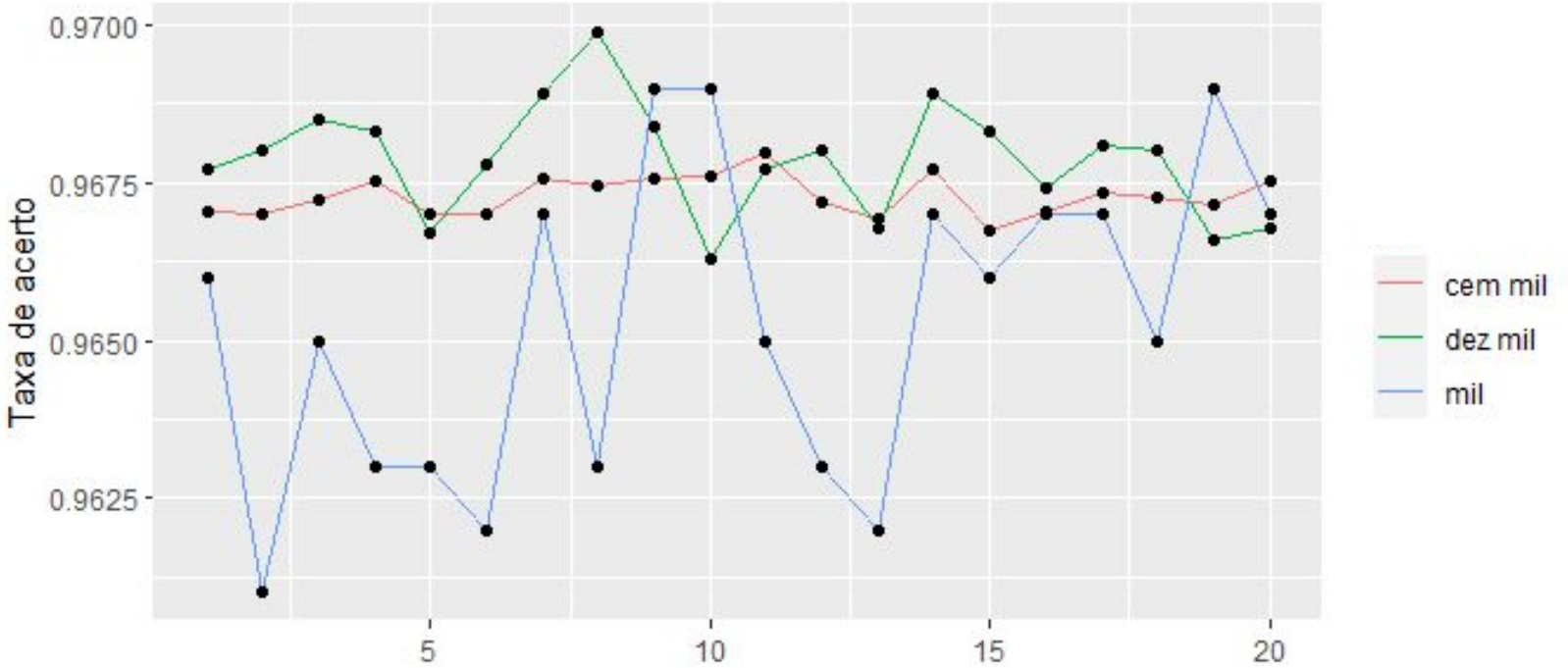
R Studio

Método: *K-fold Cross-Validation*



R Studio

Método: *K-fold Cross-Validation*



R Studio

Método: *K-fold Cross-Validation*

```r
dados <- dados1[c(1:1000),]

dados <- dados[sample(1:nrow(dados)),]

k <- 10
ct1 <- 1
vet <- c()

for(i in 1:k){
  treino <- dados[-(ct1:(ct1+(nrow(dados)/k))-1),]
  teste <- dados[ct1:(ct1+(nrow(dados)/k)-1),]
  ct1 <- ct1 + nrow(dados)/k
  modelo <- rpart(diabetes ~ .,data = treino,
                  method="class",control=rpart.control(cp=0))
  vet[i] <- sum(predict(modelo,teste,"class")==teste[,9])/(nrow(teste))
  remove(treino)
  remove(teste)
}

mean(vet)
```

R Studio

Método: *K-fold Cross-Validation: Bias-Variance Trade-Off*

Validation Set: extremamente tendencioso ((bias)) baseado na partição do subconjunto de treino e por consequência inconsistente para conjuntos menores.

LOOCV: não há tendência ((bias)) uma vez que não há aleatoriedade nos conjuntos porém há uma correlação muito grande entre as *n* taxas de acerto encontradas já que, para todo *i*, *n(i)* e *n(i+1)* foram treinados em conjuntos com 998 observações iguais. Isso faz com que a média das taxas de acerto tenha variância ((variance)) alta.

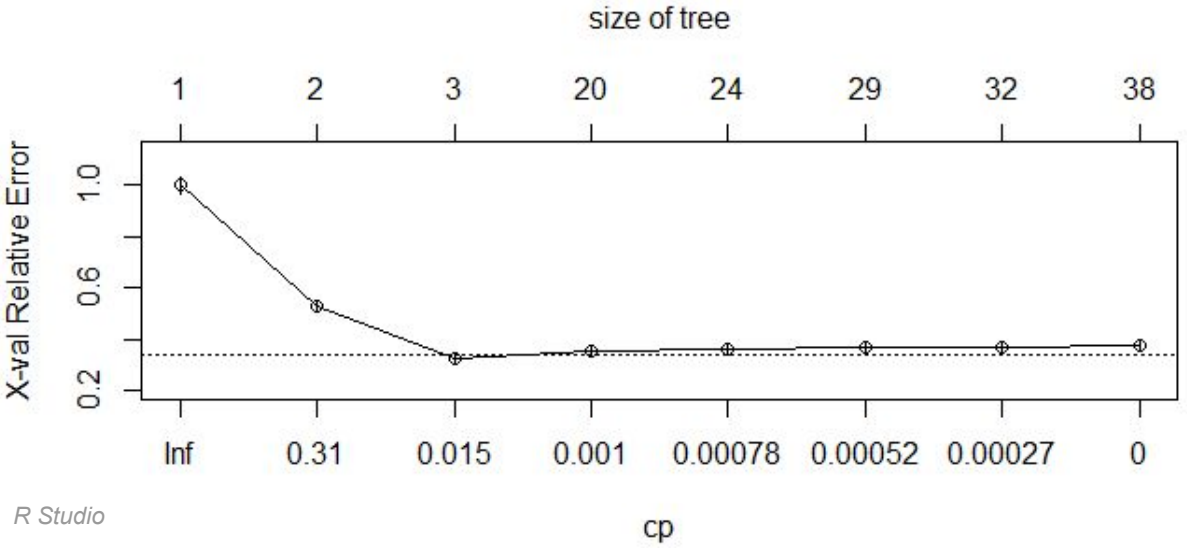
Método: *K-fold Cross-Validation: Bias-Variance Trade-Off*

K-fold Cross-Validation: promove um equilíbrio entre variância, uma vez que os subconjuntos são menos correlacionados, e tendenciosidade, uma vez que embora haja aleatorização, as *k* repetições garantem um resultado mais consistente.

k=5 e *k*=10 são considerados os valores que oferecem o melhor equilíbrio entre esses fatores.

Aplicação prática:

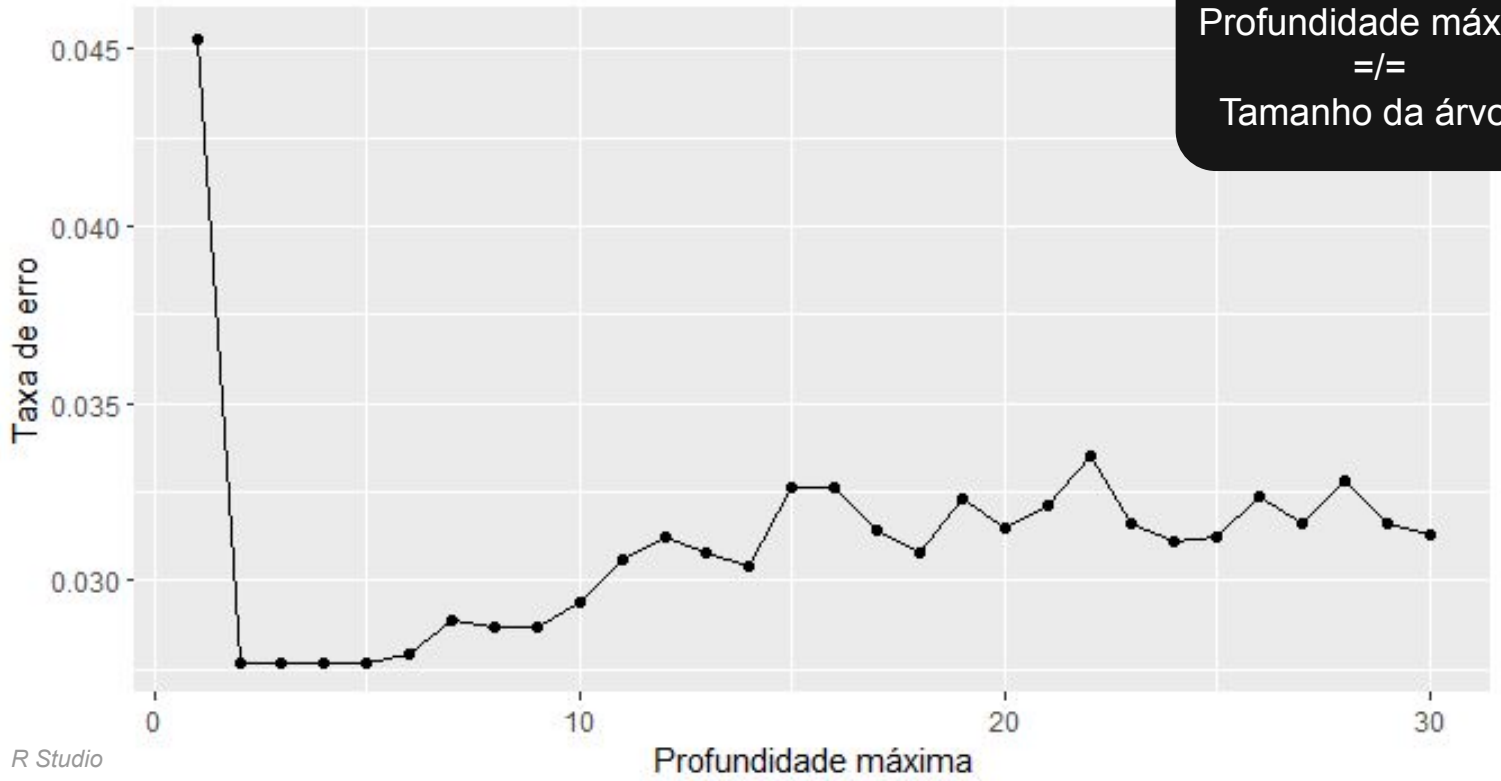
Na prática, usamos Cross-Validation para testar diferentes parâmetros de um modelo de previsão. Neste exemplo aplicaremos *K-fold CV* em todas as árvores com *profundidade máxima* de 1 a 38 e compararemos com o parâmetro ideal definido pelo *parâmetro de complexidade* do modelo.



R Studio

Aplicação prática:

Addendum:
Profundidade máxima
=/=
Tamanho da árvore



R Studio

Aplicação prática:

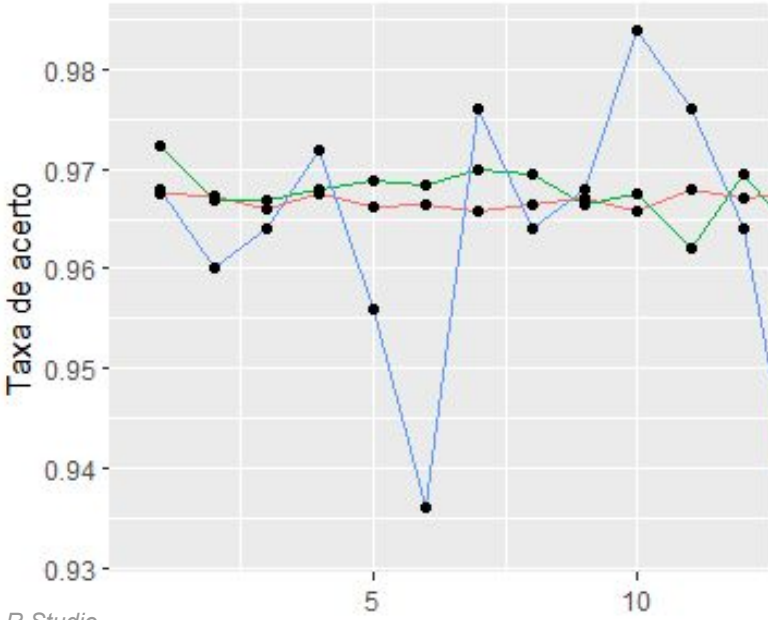
```
vet2<-c()
dados <- dados1[c(1:ct0),]

for(j in 1:30){

  dados <- dados[sample(1:nrow(dados)),]

  k <- 10
  ct1 <- 1
  vet <- c()

  for(i in 1:k){
    treino <- dados[-(ct1:(ct1+(nrow(dados)/k))-1),]
    teste <- dados[ct1:(ct1+(nrow(dados)/k)-1),]
    ct1 <- ct1 + nrow(dados)/k
    modelo <- rpart(diabetes ~ .,data = treino,
                    method="class",control=rpart.control(cp=0,maxdepth = j))
    vet[i] <- sum(predict(modelo,teste,"class")==teste[,9])/(nrow(teste))
    remove(treino)
    remove(teste)
  }

  vet2[j] <- mean(vet)
}
```

R Studio

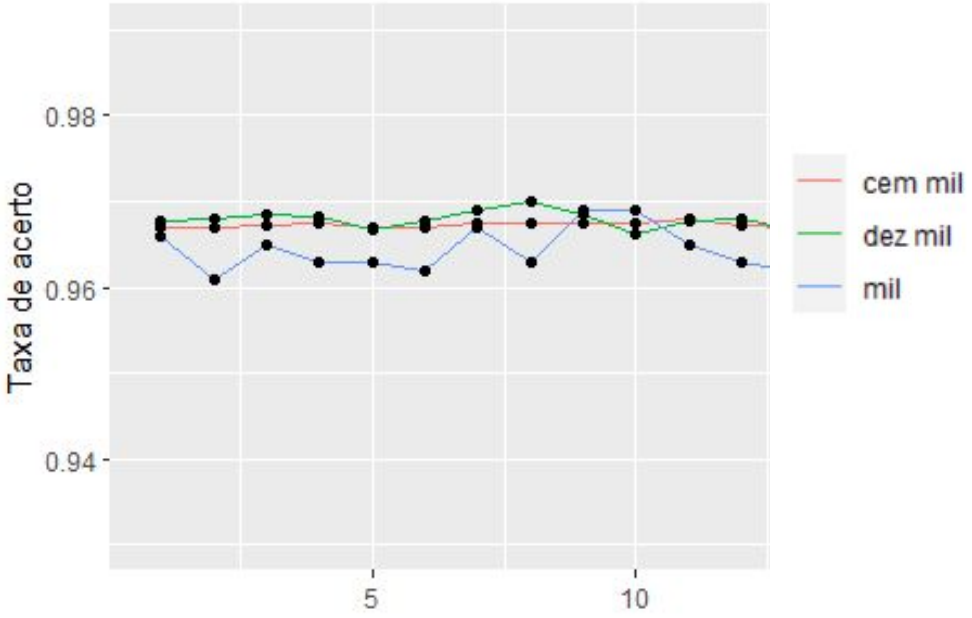
Conclusão:

- Para conjuntos de dados suficientemente grandes, todos os métodos tendem a ser eficientes

Validation Set



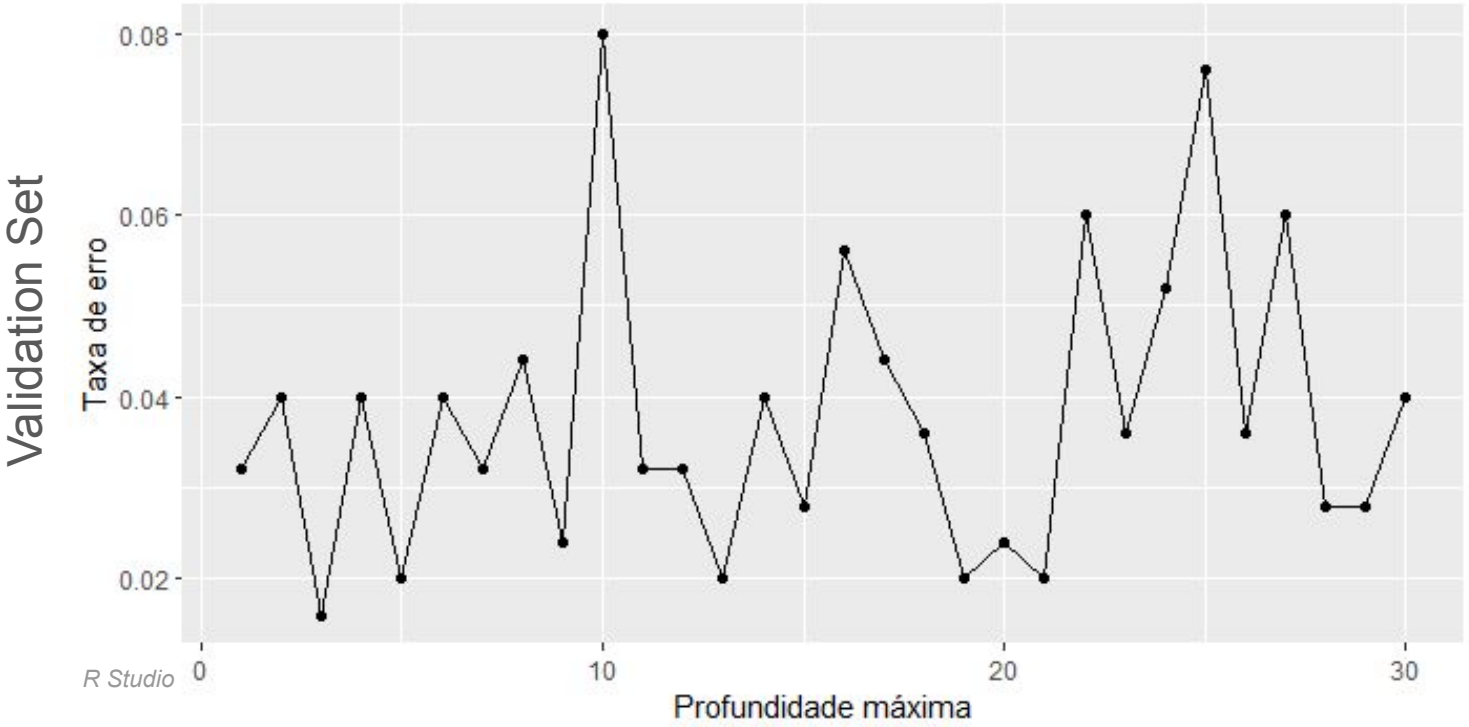
K-fold Cross Validation



R Studio

Conclusão:

- Ainda assim, K-fold é indubitavelmente o método ideal dentre os estudados



Bootstrap & Bagging:

Diferente dos métodos anteriores, *bootstrap* é o processo de partição usado na construção do modelo de previsão. Consiste de dividir o conjunto de dados em *B* subconjuntos compostos de observações aleatórias *com repetição* e realizar a construção de árvores de previsão a partir destes subconjuntos. Esse método alivia os efeitos da presença de observações extremas nos dados, além de permitir um uso mais eficiente de conjuntos com poucas observações.

Bagging consiste de realizar o *bootstrap*, construir uma árvore para cada subconjunto, fazer a previsão com todas as árvores e realizar um voto majoritário entre os resultados para determinar o resultado real.

Bootstrap & Bagging:

```r
dados <- dados1[1:1000,]
dadosTeste <- dados1[1001:2000,]
B <- 1000
vet <- rep(0,1000)
vet2 <- c()

for(i in 1:B){
  dadosB <- dados[sample(1:nrow(dados),replace = TRUE),]
  modelo <- rpart(diabetes ~ .,data = dadosB,
                  method="class",control=rpart.control(cp=0))
  vet <- vet + (as.numeric(predict(modelo,dadosTeste,"class"))-1)
}

for(i in 1:length(vet)){
  if(vet[i]>B/2)
    vet2[i]<-1
  else
    vet2[i]<-0
}

sum(vet2==dadosTeste[,9])/nrow(dadosTeste)
```

R Studio

Bootstrap & Bagging:

```
> sum(vet2==dadosTeste[,9])/nrow(dadosTeste)
[1] 0.97
>
> modelo2 <- rpart(diabetes ~ .,data = dados,
+                  method="class",control=rpart.control(cp=0))
> sum(predict(modelo2,dadosTeste,"class")==dadosTeste[,9])/(nrow(dadosTeste))
[1] 0.969
```

R Studio

Bootstrap & Bagging: estimação de erro *Out-Of-Bag*

O processo de estimação de erro para o modelo de *bagging* consiste de: para cada observação *i* realizar sua previsão com todos as árvores que foram construídas sem ela, realizar a votação e comparar com sua classificação real.

Foi estimado que para uma observação qualquer será possível realizar B/3 previsões pois cada subconjunto terá em torno de ⅔ do conjunto de dados

Bootstrap & Bagging: importância de variáveis

Para aumentar a eficiência do modelo é possível realizar um estudo das variáveis do conjunto de dados e descartar aquelas que não auxiliam ou até mesmo atrapalham na geração das árvores.

Exemplo de variáveis descartáveis: duas variáveis com correlação muito alta, uma delas pode ser descartada pois ambas informam a mesma coisa.

FIM.