Aprendizado de Máquina com o Pacote `tidymodels`

XVII Escola de Modelos de Regressão

Marcus Nunes

29 e 30 de Novembro de 2021

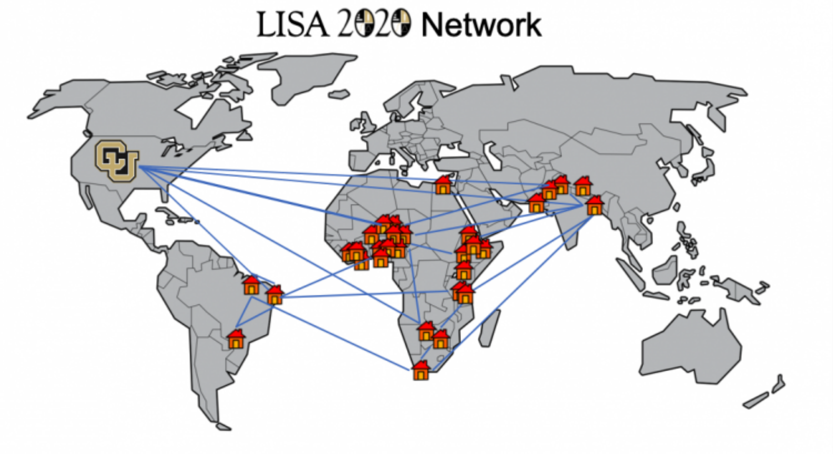
Departamento de Estatística - UFRN

Apresentação

Apresentação

- Marcus Nunes, Professor Adjunto no Departamento de Estatística da UFRN
- PhD em Estatística pela Pennsylvania State University (2013)
- Interessado em Educação Estatística, Aprendizado de Máquina e Projetos de Colaboração Estatística
- Site pessoal: `marcusnunes.me`
- Curso de big data: `introbigdata.org`
- Email: `marcus@marcusnunes.me`

Apresentação



link

Pré-Requisitos

Pré-Requisitos

- Familiariade com a Linguagem R
- Análise de Componentes Principais
- `tidyverse` (pacotes como `ggplot2` e `dplyr` são suficientes)
- Modelos de regressão
- Material do curso disponível em `https://github.com/mnunes/emr-2021`

Validação Cruzada

Conjuntos de Treino e Teste

- Os métodos que veremos neste minicurso são computacionalmente complexos
- Eles envolvem passos extras que normalmente não são utilizados em métodos paramétricos de ajuste de modelos
- Os dois conceitos principais que nos permitirão avaliar se nossos modelos foram bem ajustados são
  - Divisão dos dados em conjuntos de treino e teste
  - Validação cruzada

Conjuntos de Treino e Teste

- Utilizaremos este conceito em algoritmos para classificação e predição de dados
- Estes algoritmos são ferramentas importantes para descobrir padrões
- Eles permitem que generalizemos comportamentos presentes nos dados

Conjuntos de Treino e Teste

- Classificação ou Regressão?
- A resposta é categórica ou numérica?
- Resposta categorizada: classificação
- Resposta numérica: regressão

Conjuntos de Treino e Teste

- Aprendizagem Supervisionada: existe um conjunto de treino no qual o algoritmo se baseia para encontrar as relações entre os dados, com os valores da variável resposta bem definidos
- Aprendizagem Não-Supervisionada: não existe um conjunto de treino no qual o algoritmo se baseia para encontrar as relações entre os dados, sem os valores da variável resposta bem definidos

Conjuntos de Treino e Teste

- Ao aplicarmos um algoritmo de aprendizagem supervisionada, necessitamos ser capazes de avaliar o quão bom (ou ruim) é o nosso método
- A maneira mais comum de fazer isto é através de divisão dos dados originais em dois conjuntos:
    - Conjunto de treino
    - Conjunto de teste

Conjuntos de Treino e Teste

- O conjunto de treino é aquele no qual aplicamos o algoritmo, informando a resposta correta para o algoritmo
- Em geral, utilizamos de 50% a 80% dos dados originais no conjunto de treino
- O algoritmo, então, se ajusta de modo a prever com a maior exatidão possível os outputs que nos interessam

Conjuntos de Treino e Teste

- O conjunto de teste é aquele que utilizamos para prever resultados e verificar como o algoritmo se comportaria em dados reais
- Em geral, utilizamos de 50% a 20% dos dados originais no conjunto de teste (o percentual de dados deste conjunto depende do percentual utilizado no conjunto de treino, pois um é complementar do outro)
- Assim, podemos avaliar o quão bem o algoritmo está conseguindo prever novos resultados que não estavam no conjunto original
- É como se simulássemos a coleta de novos dados

Métricas de Avaliação

- Para avaliar a eficiência do algoritmo de classificação, utilizamos medidas como sensitividade e especificidade
- Sensitividade é a razão entre o número de positivos encontrados pelo modelo pelo total de positivos nos dados
- Especificidade é a razão entre o número de negativos encontrados pelo modelo pelo total de negativos nos dados

Métricas de Avaliação

		Referência	
		Positivo	Negativo
Predição	Positivo	a	b
	Negativo	c	d

- Acurácia: $p_0 = \frac{a+d}{a+b+c+d}$
- Sensitividade: $\frac{a}{a+c}$
- Especificidade: $\frac{d}{b+d}$
- Curva ROC: Sensitividade (Verdadeiros Positivos) vs. 1-Especificidade (Falsos Positivos)

Métricas de Avaliação

		Referência	
		Positivo	Negativo
Predição	Positivo	a	b
	Negativo	c	d

- Kappa:

$$p_{\text{Sim}} = \frac{a+b}{a+b+c+b} \times \frac{a+c}{a+b+c+b}$$

$$p_{\text{Não}} = \frac{c+d}{a+b+c+b} \times \frac{b+d}{a+b+c+b}$$

$$p_e = p_{\text{Sim}} + p_{\text{Não}}$$

$$\kappa = \frac{p_0 - p_e}{1 - p_e}$$

Métricas de Avaliação

- Para avaliar a eficiência do algoritmo de regressão, utilizamos medidas como erro quadrático médio, erro absoluto médio e coeficiente de determinação
- Suponha que temos uma amostra de tamanho n e observações y_i, $i = 1, \cdots, n$
- Suponha que possuímos uma forma de estimar os valores de y_i e chamamos estas estimações de $\widehat{y}_i$

Métricas de Avaliação

- A fórmula do erro quadrático médio é dada por

$$\text{EQM} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$$

Normalmente, a estatística que se usa é a raiz do erro quadrático médio, dada por

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}$$

Métricas de Avaliação

- O erro absoluto médio é dado por

$$\text{EAM} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \widehat{y}_i\right|$$

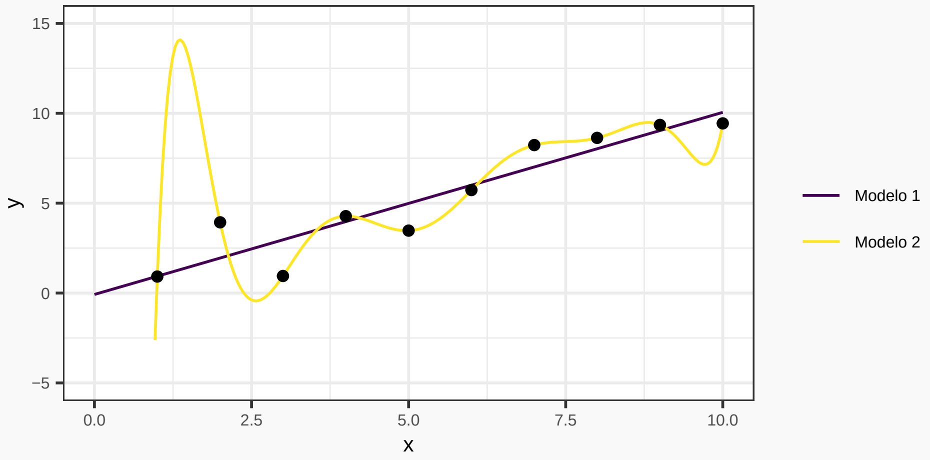
- Podemos calcular o coeficiente de determinação como

$$R^2 = \frac{\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2}{\sum_{i=1}^{n}\left(y_i - \overline{y}\right)^2}$$

Métricas de Avaliação

- O ideal é que as medidas de ajuste do modelo sejam similares nos conjuntos de treino e teste
- Fazemos isso para verificar que não houve sobreajuste (*overfitting*) nos dados
- Se o modelo está sobreajustado, isto significa que ele se ajusta muito bem aos dados originais, mas não é um modelo que é generalizável

Métricas de Avaliação



Validação Cruzada

- Dividir os dados apenas uma vez em treino e teste pode, ainda assim, gerar vícios
- Então por que não realizar esta divisão mais de uma vez?
- Desta forma, a ocorrência de eventos anômalos fica diluída

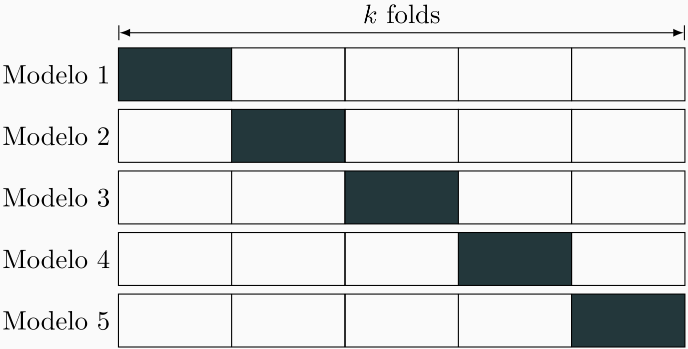
Validação Cruzada

- Validação Cruzada é um método de reamostragem (*resampling*)
- Baseia-se na ideia de tomar diversas amostras aleatórias da mesma população
- Estas amostras aleatórias são todas tomadas a partir de uma amostra que já obtivemos

Validação Cruzada

- O procedimento geral para realizar a validação cruzada é o seguinte:
- Crie k partições aleatórias do conjunto de dados com o mesmo tamanho aproximado
- Para $j = 1, \cdots, k$, faça
    1. Treine o modelo em todos os blocos, exceto o bloco j
    2. Teste o modelo no bloco j
    3. Estime o erro em cada bloco j
- Calcule a média dos erros

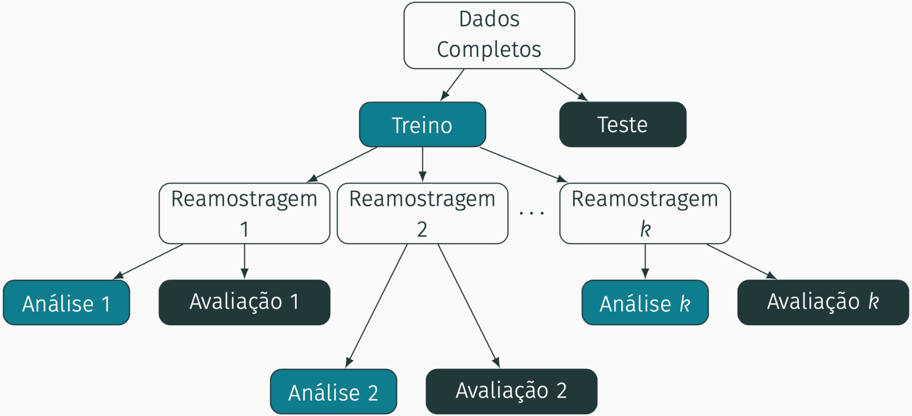
Validação Cruzada



Processo Completo

- A divisão dos dados em treino e teste ocorre em toda análise que formos realizar
- Encontramos o melhor modelo no conjunto de treino através da validação cruzada
- Verificamos o resultado obtido no conjunto de teste
- Ou seja, os dois métodos se complementam

Processo Completo



tidymodels

`tidymodels`

- O pacote **`tidymodels`** irá ajudar a organizar nosso fluxo de trabalho
- De modo geral, cada ferramenta que veremos daqui em diante está implementada em um pacote diferente
- Com o **`tidymodels`** podemos usar uma sintaxe similar para todas essas ferramentas, focando menos em aprender como cada uma é utilizada, e nos dedicando mais a interpretar os resultados obtidos

`tidymodels`

- O `tidymodels` não é um pacote em si
- Assim como o `tidyverse`, o `tidymodels` é uma coleção de pacotes com aplicações específicas
- Os dois pacotes partem dos mesmos princípios básicos para criarem fluxos de trabalho consistentes

Os princípios *tidy* são os seguintes:

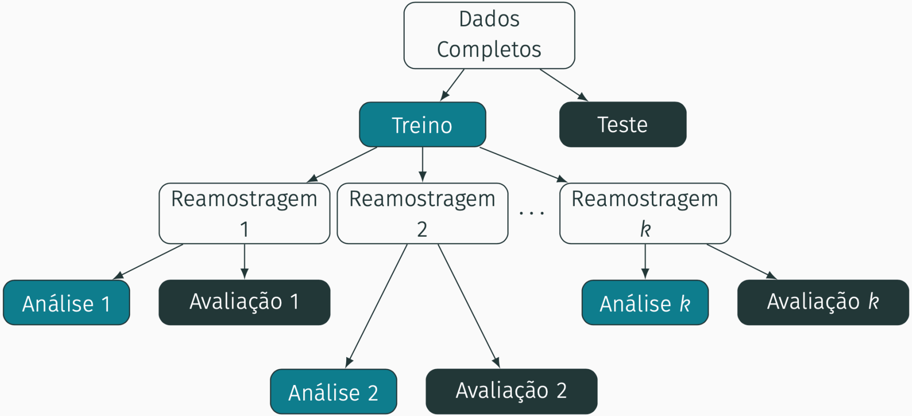
1. Reutilizar estruturas de dados existentes
2. Criar funções simples que utilizam *pipe* (%>%)
3. Adotar programação funcional
4. Feito para humanos

`tidymodels`

Os principais pacotes disponíveis dentro do `tidymodels` são:

- `rsample`: tipos diferentes de reamostragem
- `recipes`: transformações para pré-processamento de dados
- `parsnip`: uma interface comum para modelagem
- `yardstick`: medidas de desempenho do modelo

tidymodels



Dados Completos
Treino
Teste
Reamostragem 1
Reamostragem 2
...
Reamostragem k
Análise 1
Avaliação 1
Análise 2
Avaliação 2
Análise k
Avaliação k

Aplicação Tradicional

- A seguir veremos como ajustar um modelo de regressão linear simples a um conjunto de dados
- É importante entendermos os passos do ajuste de um modelo que já conhecemos para depois expandirmos o método para outros tipos de modelagem

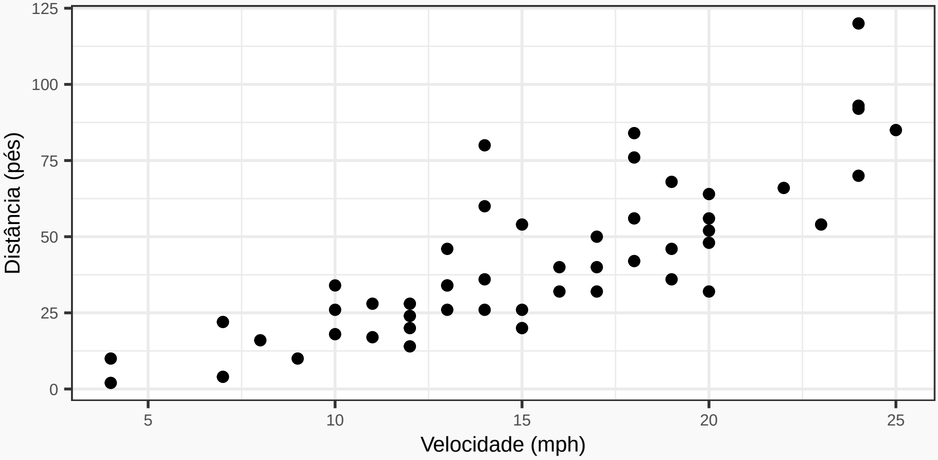
Aplicação Tradicional

- Nossa regressão será feita no conjunto de dados `cars`
- Ele possui apenas duas colunas
    - `speed`: velocidade do carro (milhas por hora)
    - `dist`: distância que o carro levou para parar completamente (pés)

Aplicação Tradicional

```r
library(tidymodels)
theme_set(theme_bw())

ggplot(cars, aes(x = speed, y = dist)) +
  geom_point() +
  labs(x = "Velocidade (mph)", y = "Distância (pés)")
```

Aplicação Tradicional



Aplicação Tradicional

```r
# determinacao do software

cars_lm <-
  linear_reg() %>%
  set_engine("lm")

# ajuste do modelo

cars_lm_fit <-
  cars_lm %>%
  fit(dist ~ speed,
      data = cars)
```

Aplicação Tradicional

```r
# resultados

cars_lm_fit
```

```
## parsnip model object
##
## Fit time:  4ms
##
## Call:
## stats::lm(formula = dist ~ speed, data = data)
##
## Coefficients:
## (Intercept)        speed
```

Aplicação Tradicional

```r
# resultados

tidy(cars_lm_fit)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -17.6      6.76      -2.60 1.23e- 2
## 2 speed           3.93     0.416      9.46 1.49e-12
```

Aplicação Treino/Teste

```r
# semente aleatoria

set.seed(555)

# 75% dos dados como treino

cars_split <- initial_split(cars, prop = .75)
cars_split
```

```
## <Analysis/Assess/Total>
## <37/13/50>
```

Aplicação Treino/Teste

```r
# criar os conjuntos de dados de treino e teste

cars_treino <- training(cars_split)
nrow(cars_treino)/nrow(cars)
```

```
## [1] 0.74
```

```r
cars_teste  <- testing(cars_split)
nrow(cars_teste)/nrow(cars)
```

```
## [1] 0.26
```

Aplicação Treino/Teste

```
# receita

cars_rec <-
  recipe(dist ~ speed,
         data = cars_treino)

cars_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
```

```r
# modelo

cars_lm <-
  linear_reg() %>%
  set_engine("lm")

cars_lm
```

```
## Linear Regression Model Specification (regression)
##
## Computational engine: lm
```

Aplicação Treino/Teste

```r
# criar workflow

cars_wflow <-
  workflow() %>%
  add_recipe(cars_rec) %>%
  add_model(cars_lm)

# ajuste do modelo

cars_lm_fit_treino <- fit(cars_wflow, cars_treino)
```

Aplicação Treino/Teste

```r
tidy(cars_lm_fit_treino)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -17.6      7.70      -2.28 2.86e- 2
## 2 speed           3.99     0.465      8.58 3.96e-10
```

```r
tidy(cars_lm_fit)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
```

Aplicação Validação Cruzada

```r
# semente aleatoria

set.seed(321)

# divisao dos dados

cars_treino_cv <- vfold_cv(cars_treino, v = 5)
```

Aplicação Validação Cruzada

```
cars_treino_cv
```

```
## #  5-fold cross-validation
## # A tibble: 5 x 2
##   splits          id
##   <list>          <chr>
## 1 <split [29/8]> Fold1
## 2 <split [29/8]> Fold2
## 3 <split [30/7]> Fold3
## 4 <split [30/7]> Fold4
## 5 <split [30/7]> Fold5
```

Aplicação Validação Cruzada

```r
# modelo ajustado com validacao cruzada

cars_lm_fit_cv <- fit_resamples(cars_wflow, cars_treino_cv)
```

Aplicação Validação Cruzada

```
cars_lm_fit_cv
```

```
## # Resampling results
## # 5-fold cross-validation
## # A tibble: 5 x 4
##   splits         id    .metrics         .notes
##   <list>         <chr> <list>           <list>
## 1 <split [29/8]> Fold1 <tibble [2 x 4]> <tibble [0 x 1]>
## 2 <split [29/8]> Fold2 <tibble [2 x 4]> <tibble [0 x 1]>
## 3 <split [30/7]> Fold3 <tibble [2 x 4]> <tibble [0 x 1]>
## 4 <split [30/7]> Fold4 <tibble [2 x 4]> <tibble [0 x 1]>
## 5 <split [30/7]> Fold5 <tibble [2 x 4]> <tibble [0 x 1]>
```

Aplicação Validação Cruzada

```r
# resultados

collect_metrics(cars_lm_fit_cv)
```

```
## # A tibble: 2 x 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   15.0       5   2.13  Preprocessor1_Model1
## 2 rsq     standard    0.644     5   0.129 Preprocessor1_Model1
```

```r
sqrt(mean((predict(cars_lm_fit$fit) - cars$dist)^2))
```

```
## [1] 15.06886
```

Aplicação Validação Cruzada

```r
# resultados no conjunto de teste

resultado <-
  cars_teste %>%
  bind_cols(predict(cars_lm_fit_treino, cars_teste) %>%
            rename(predicao_lm = .pred))
```

Aplicação Validação Cruzada

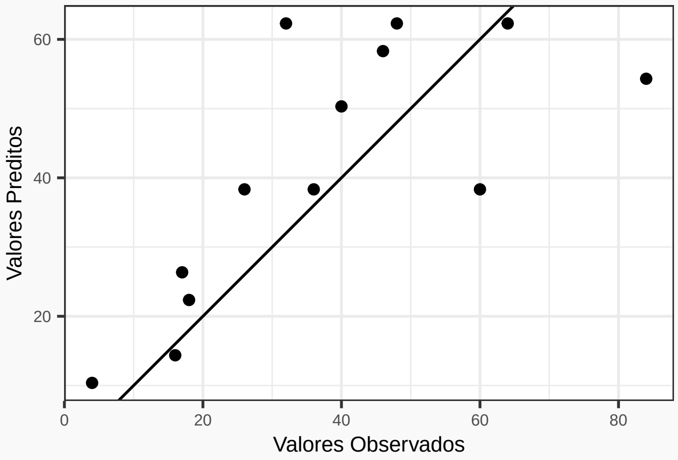
```
# resultado final

metrics(resultado,
        truth = dist,
        estimate = predicao_lm)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      15.3
## 2 rsq     standard       0.538
## 3 mae     standard      12.1
```

Aplicação Validação Cruzada

```r
# grafico final

ggplot(resultado, aes(x = dist, y = predicao_lm)) +
  geom_point() +
  labs(x = "Valores Observados", y = "Valores Preditos") +
  geom_abline(intercept = 0, slope = 1) +
  coord_fixed()
```

Aplicação Validação Cruzada



Exercícios

Exercícios

O conjunto de dados `mpg` faz parte do pacote `ggplot2`. Ele possui informações a respeito de carros vendidos no mercado norte-americano. As variáveis são:

- `manufacturer`: fabricante
- `model`: modelo do carro
- `displ`: tamanho do motor em litros
- `year`: ano de fabricação
- `cyl`: número de cilindros
- `trans`: tipo de transmissão
- `drv`: tipo de tração
- `cty`: consumo na cidade em milhas por galão
- `hwy`: consumo na estrada em milhas por galão
- `fl`: tipo de combustível
- `class`: tipo de automóvel

Exercícios

1. Crie um novo objeto chamado `mpg2` com apenas as variáveis numéricas presentes no conjunto de dados
2. Encontre a variável com a maior correlação negativa com a variável `hwy`. Visualize essa relação em um gráfico de dispersão. Intuitivamente, a correlação entre estas variáveis faz sentido? Explique.
3. Crie conjuntos de treinamento e teste com o objeto `mpg2`. Reserve 80% das observações para o conjunto de treinamento.
4. Utilize a validação cruzada com 5 grupos para ajustar um modelo de regressão linear simples. Utilize `hwy` como variável resposta e a variável encontrada na pergunta 2 como preditora.

Exercícios

5. Verifique se o resultado do ajuste ficou aceitável, comparando os RMSE do modelo nos conjuntos de treinamento e teste.
6. Ajuste uma regressão linear múltipla neste conjunto de dados. Mantenha `hwy` como variável resposta e adicione as outras variáveis numéricas como variáveis preditoras. Padronize as variáveis preditoras no conjunto de teste com as funções `step_center`, `step_scale`, `prep` e `juice` do pacote `tidymodels`.
7. Como ficou o novo ajuste? Decida baseando-se em medidas relevantes nos conjuntos de treinamento e teste (transforme os dados do conjunto teste com a função `bake`) e em gráficos de diagnóstico.

Random Forest

Random Forest

- É um algoritmo derivado das árvores de classificação e regressão
- Foi criado por Tin Kam Ho em 1995 e aperfeiçoado por Leo Breiman em 2001
- Surgiu em um artigo discutindo duas culturas para análise de dados: uma derivada da estatística, outra derivada da computação
- Se tornou muito popular popular nos últimos anos, servindo como base para algoritmos mais avançados

Random Forest

- É muito utilizado tanto em aplicações de classificação quanto regressão
- Pode lidar com problemas do tipo “small n large p’ ’ - problemas em que temos um tamanho amostral n muito pequeno se comparado ao número de parâmetros p do modelo
- Não é utilizada apenas para predição, podendo ser aplicada em problemas de seleção de variáveis

Random Forest

- É uma combinação de várias árvores de regressão e classificação
- Parte do princípio que previsões feitas a partir da combinação de modelos são melhores do que de um modelo apenas
- Os erros dos estimadores são combinados e diminuídos, gerando assim um resultado com menor variância
- Além disso, por ser baseado em árvores, transformações monótonas nas variáveis preditoras não influenciam no desempenho dos algoritmos

Bagging

- É uma sigla para Bootstrap agggregating
- Combina o resultados das classificações de conjuntos de treinamento gerados aleatoriamente
- Melhora a estabilidade e a acurácia dos algoritmos, além de reduzir a variância e evitar o sobreajuste

Bagging

- Bootstrap é uma técnica de reamostragem com reposição utilizada para estimar algum parâmetro de uma população
- Assuma que dispomos de uma amostra $X_1, X_2, \cdots, X_n$ e queremos alguma informação sobre o parâmetro θ da variável aleatória X
- São tomadas B reamostras $X_1^*, X_2^*, \cdots, X_n^*$, com reposição, de tamanho n
- Calculamos a estatística de interesse $\widehat{\theta}^*$ para cada reamostra
- Assim, conseguimos construir a distribuição empírica do estimador $\widehat{\theta}$

Bagging

- Gere B subamostras com reposição a partir do conjunto de treinamento
- Treine um modelo CART em cada nova amostra
- Classificação: a classe é definida pela maioria dos votos
- Regressão: média dos valores preditos
- A estabilidade e a acurácia são melhoradas, além de reduzir a variância e evitar o sobreajuste

Algoritmo

- Random forest é uma coleção de várias árvores de decisão decorrelacionadas
- Algoritmos de decorrelação são técnicas usadas para reduzir autocorrelação
- Random forest (floresta aleatória) possui este nome porque é definido através do uso de várias árvores de classificação e regressão

Algoritmo

- Suponha que temos uma matriz S composta de n amostras de treinamento, com 3 variáveis preditoras (X, Y e Z)

$$S = \begin{bmatrix} X_1 & Y_1 & Z_1 & C_1 \\ X_2 & Y_2 & Z_2 & C_2 \\ \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & C_n \end{bmatrix}$$

- A ideia é criar B subamostras aleatórias $S_1, S_2, S_3, \cdots, S_B$ da matriz S, todas de tamanho n, com reposição

$$S_1 = \begin{bmatrix} X_5 & Y_5 & Z_5 & C_5 \\ X_8 & Y_8 & Z_8 & C_8 \\ \vdots & \vdots & \vdots & \vdots \\ X_{33} & Y_{33} & Z_{33} & C_{33} \end{bmatrix}$$

$$S_2 = \begin{bmatrix} X_3 & Y_3 & Z_3 & C_3 \\ X_{20} & Y_{20} & Z_{20} & C_{20} \\ \vdots & \vdots & \vdots & \vdots \\ X_6 & Y_6 & Z_6 & C_6 \end{bmatrix}$$

Algoritmo

$$S_3 = \begin{bmatrix} X_9 & C_9 \\ X_{38} & C_8 \\ \vdots & \vdots \\ X_{45} & C_{45} \end{bmatrix}$$

$$\vdots$$

$$S_B = \begin{bmatrix} Y_1 & Z_1 & C_1 \\ Y_{12} & Z_{12} & C_{12} \\ \vdots & \vdots & \vdots \\ Y_{97} & Z_{97} & C_{97} \end{bmatrix}$$

Algoritmo

- Ajustamos, a partir de cada uma das subamostras criadas, um modelo CART diferente
- Cada um desses modelos terá um número aleatório de variáveis preditoras
- Ou seja, S_1 terá um modelo próprio com k_1 variáveis preditoras, S_2 terá outro modelo próprio com k_2 variáveis preditoras e assim por diante
- Os $k_i, i = 1, \cdots, B$ não serão necessariamente iguais

Algoritmo

- Ao fim, teremos B CARTs diferentes
- Portanto, teremos uma floresta com B árvores
- A partir destes resultados, calculamos a média das árvores estimadas no caso de regressão ou contamos a maioria de votos, no caso de classificação

Algoritmo

- De maneira mais formal, temos o seguinte algoritmo:
    1. Tome B subconjuntos de seu conjunto de dados originais, com reposição
    2. Ajuste uma CART $\widehat{f}_i$ a cada um destes subconjuntos, com um número aleatório de variáveis preditoras
    3. Encontre uma estimativa para a random forest fazendo

$$\widehat{f} = \frac{1}{B} \sum_{i=1}^{B} \widehat{f}_i$$

Algoritmo

- É possível encontrar a importância de cada variável ao rodarmos uma random forest
- Durante o processo de ajuste do modelo, o erro de ajuste para cada nó é medido e registrado
- Para medir a importância da j-ésima variável, basta permutar os seus valores dentro de cada iteração
- Assim, temos os valores dos erros de ajuste dos conjuntos de dados normais e perturbados
- Desta forma medimos a importância de cada variável

Algoritmo

- Cada vez uma divisão ocorre para a variável j, o nível de impureza para os dois nós descendentes é menor do que o do nó original
- Somando os índices de Gini para cada variável sobre todas as árvores, obtemos uma medida da importância da variável que é consistente com o do teste de permutação descrito anteriormente, só que mais rápido

Importância de Gini

- O índice de pureza Gini é definido como

$$G = \sum_{i=1}^{n_c} p_i(1 - p_i)$$

em que n_c é o número de classes na variável j e p_i é a proporção desta classe (note que este G é calculado para cada árvore na floresta)

- A partir disto, a importância é calculada como

$$I = G_{\text{pai}} - G_{\text{filho 1}} - G_{\text{filho 2}}$$

- Por fim, é calculada a média de todos os nós para todas as árvores

Tunning

- Será que todas as variáveis (*mtry*) são importantes para o ajuste do modelo?
- Qual a melhor profundidade (*levels*) para a árvore?
- Estas e outras perguntas podem ser respondidas através do tunning de hiperparâmetros

Tunning

- Valores como *mtry*, *levels* e outros, que ajudam na procura do melhor modelo para os nossos dados, são chamados de hiperparâmetros
- Queremos encontrar a melhor combinação dos hiperparâmetros em cada análise realizada
- Ou seja, queremos maximizar o valor da acurácia (ou de alguma outra medida) considerando diferentes valores para estes hiperparâmetros

Tunning

- Utilizaremos uma grade de procura para isso
- Iremos definir valores específicos para cada hiperparâmetro e ajustaremos modelos para todas as combinações possíveis
- Esse processo é computacionalmente intenso e seu tempo de execução dependerá diretamente de características como a quantidade de combinações de hiperparâmetros, o tamanho do conjunto de dados e a complexidade da validação cruzada, dentre outros

Exemplo

Exemplo

- O conjunto de dados `penguins` faz parte do pacote `palmerpenguins`
- Ele possui 344 observações para 8 variáveis
- Nosso objetivo é classificar as espécies de pinguins baseando-nos nas outras variáveis do conjunto de dados

Exemplo

```r
# pacotes carregados

library(tidymodels)
library(onehot)
library(palmerpenguins)
library(GGally)
library(ggfortify)
library(vip)
```

Exemplo

```
# checagem dos dados

glimpse(penguins)
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie,
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgers
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9,
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8,
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 1
## $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625,
## $ sex               <fct> male, female, female, NA, female, male,
```

Exemplo

```
# criacao de variaveis dummy

pp <-
  penguins %>%
  select(!where(is.numeric)) %>%
  select(-species) %>%
  onehot() %>%
  predict(penguins) %>%
  as.data.frame() %>%
  select(i_Biscoe     = `island=Biscoe`,
         i_Dream      = `island=Dream`,
         i_Torgersen = `island=Torgersen`,
         s_fem        = `sex=female`,
```

Exemplo

```r
pp <-
  penguins %>%
  select(where(is.numeric), species, -year) %>%
  bind_cols(pp) %>%
  relocate(species) %>%
  na.omit()
```

Exemplo

```
# treino/teste

penguins %>%
  group_by(species) %>%
  count()
```

```
## # A tibble: 3 x 2
## # Groups:   species [3]
##   species       n
##   <fct>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

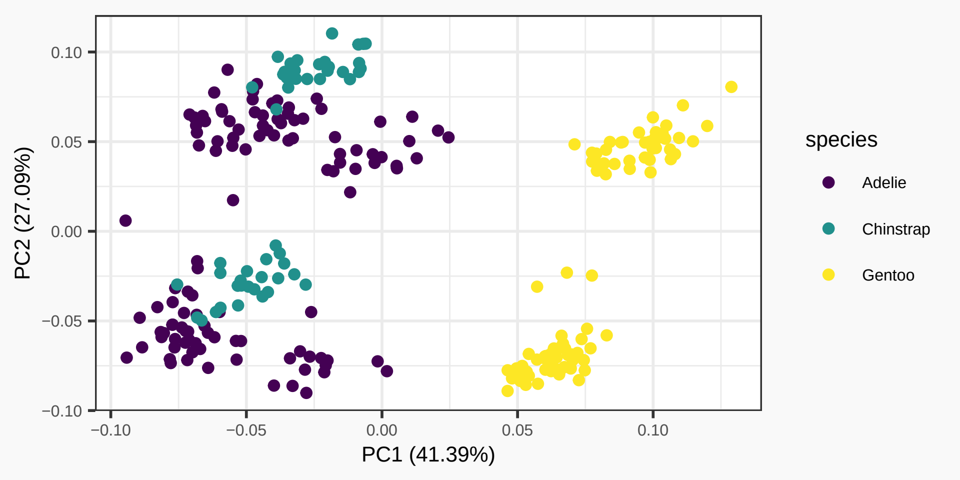
Exemplo

```r
# 75% dos dados como treino

set.seed(1232)

pp_split <- initial_split(pp, prop = .75, strata = species)

# criar os conjuntos de dados de treino e teste

pp_treino <- training(pp_split)
pp_teste  <- testing(pp_split)
```

Exemplo

```r
# eda

autoplot(prcomp(pp_treino %>% select(-species),
                center = TRUE, scale. = TRUE),
         data = pp_treino,
         colour = "species") +
  scale_colour_viridis_d()
```

Exemplo

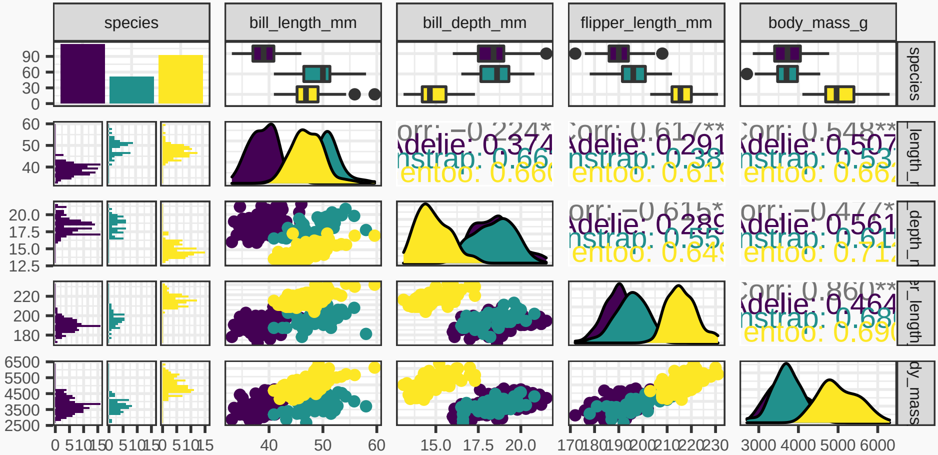


0.10
0.05
0.00
−0.05
−0.10
PC2 (27.09%)
−0.10
−0.05
0.00
0.05
0.10
PC1 (41.39%)
species
Adelie
Chinstrap
Gentoo

Exemplo

```
ggpairs(pp_treino %>% select(species,
                             bill_length_mm,
                             bill_depth_mm,
                             flipper_length_mm,
                             body_mass_g),
        aes(colour = species)) +
  scale_colour_viridis_d() +
  scale_fill_viridis_d()
```

Exemplo


species
bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g

Exemplo

```r
# pre-processamento

pp_rec <-
  recipe(species ~ .,
         data = pp_treino) %>%
  # remover observacoes de modo que todos os niveis de species
  # fiquem com o mesmo numero de observacoes
  themis::step_downsample(species) %>%
  # center/scale
  step_center(-species) %>%
  step_scale(-species) %>%
  # funcao para aplicar a transformacao aos dados
  prep()
```

Exemplo

```r
# aplicar a transformacao aos dados

pp_treino_t <- juice(pp_rec)

# preparar o conjunto de teste

pp_teste_t <- bake(pp_rec,
                   new_data = pp_teste)
```

Exemplo

```
#####################
# definicao do tuning

pp_rf_tune <-
  rand_forest(
    mtry = tune(),
    trees = 1000,
    min_n = tune()
  ) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")
```

Exemplo

```r
# grid de procura

pp_rf_grid <- grid_regular(mtry(range(1, 9)),
                           min_n(range(10, 50)),
                           levels = c(9, 5))
```

Exemplo

```r
# workflow

pp_rf_tune_wflow <-
  workflow() %>%
  add_model(pp_rf_tune) %>%
  add_formula(species ~ .)
```

Exemplo

```r
# definicao da validacao cruzada

set.seed(2389)

pp_treino_cv <- vfold_cv(pp_treino_t, v = 7)
```

Exemplo

```r
# avaliacao do modelo

pp_rf_fit_tune <-
  pp_rf_tune_wflow %>%
  tune_grid(
    resamples = pp_treino_cv,
    grid = pp_rf_grid
  )
```

Exemplo

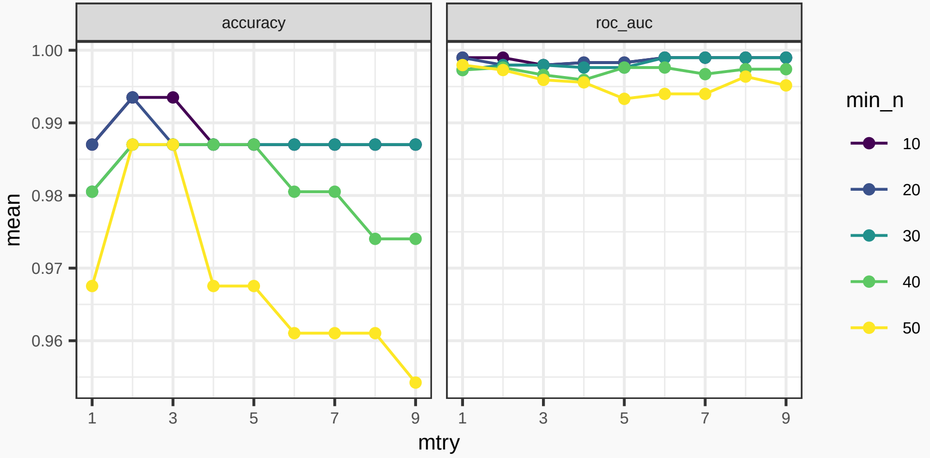
```r
# resultados

collect_metrics(pp_rf_fit_tune)
```

```
## # A tibble: 90 x 8
##     mtry min_n .metric  .estimator  mean     n std_err .config
##    <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1     1    10 accuracy multiclass 0.987     7 0.0130  Preprocesso
##  2     1    10 roc_auc  hand_till  0.999     7 0.00102 Preprocesso
##  3     2    10 accuracy multiclass 0.994     7 0.00649 Preprocesso
##  4     2    10 roc_auc  hand_till  0.999     7 0.00102 Preprocesso
##  5     3    10 accuracy multiclass 0.994     7 0.00649 Preprocesso
##  6     3    10 roc_auc  hand_till  0.998     7 0.00204 Preprocesso
```

Exemplo

```r
pp_rf_fit_tune %>%
  collect_metrics() %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(., aes(x = mtry, y = mean, colour = min_n, group = min_n)) +
  geom_line() +
  geom_point() +
  facet_grid(~ .metric) +
  scale_x_continuous(breaks = seq(1, 9, 2)) +
  scale_colour_viridis_d()
```

Exemplo



accuracy
roc_auc
mean
1.00
0.99
0.98
0.97
0.96
1
3
5
7
9
mtry
min_n
10
20
30
40
50

Exemplo

```r
# melhores modelos

pp_rf_fit_tune %>%
  show_best("roc_auc")
```

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     1    10 roc_auc hand_till  0.999     7 0.00102 Preprocessor1
## 2     2    10 roc_auc hand_till  0.999     7 0.00102 Preprocessor1
## 3     6    10 roc_auc hand_till  0.999     7 0.00102 Preprocessor1
## 4     7    10 roc_auc hand_till  0.999     7 0.00102 Preprocessor1
## 5     8    10 roc_auc hand_till  0.999     7 0.00102 Preprocessor1
```

Exemplo

```
pp_rf_fit_tune %>%
  show_best("accuracy")
```

```
## # A tibble: 5 x 8
##    mtry min_n .metric  .estimator  mean     n std_err .config
##   <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1     2    10 accuracy multiclass 0.994     7 0.00649 Preprocessor
## 2     3    10 accuracy multiclass 0.994     7 0.00649 Preprocessor
## 3     2    20 accuracy multiclass 0.994     7 0.00649 Preprocessor
## 4     1    10 accuracy multiclass 0.987     7 0.0130  Preprocessor
## 5     4    10 accuracy multiclass 0.987     7 0.00838 Preprocessor
```

Exemplo

```
# melhor modelo

pp_rf_best <-
  pp_rf_fit_tune %>%
  select_best("accuracy")

pp_rf_final <-
  pp_rf_tune_wflow %>%
  finalize_workflow(pp_rf_best)

pp_rf_final <- fit(pp_rf_final,
                   pp_treino_t)
```

Exemplo

```r
# resultados no conjunto de teste

resultado_rf <-
  pp_teste_t %>%
  bind_cols(predict(pp_rf_final, pp_teste_t) %>%
              rename(predicao_rf = .pred_class))
```

Exemplo

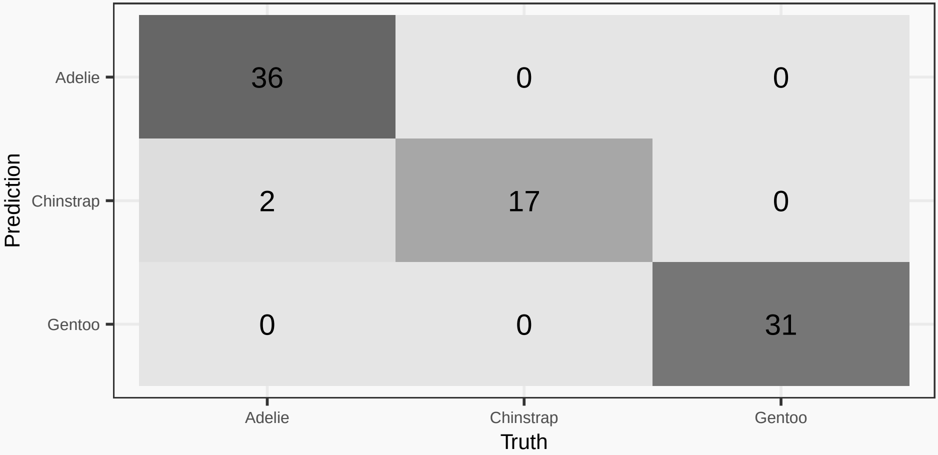
```r
metrics(resultado_rf,
        truth = species,
        estimate = predicao_rf,
        options = "roc")
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.977
## 2 kap      multiclass     0.964
```

Exemplo

```r
conf_mat(resultado_rf,
         truth = species,
         estimate = predicao_rf) %>%
  autoplot(type = "heatmap")
```

Exemplo



Prediction
Adelie
Chinstrap
Gentoo
36
0
0
2
17
0
0
0
31
Adelie
Chinstrap
Gentoo
Truth

Exemplo

```r
# sensitividade

sens(resultado_rf,
     truth = species,
     estimate = predicao_rf)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    macro          0.982
```

Exemplo

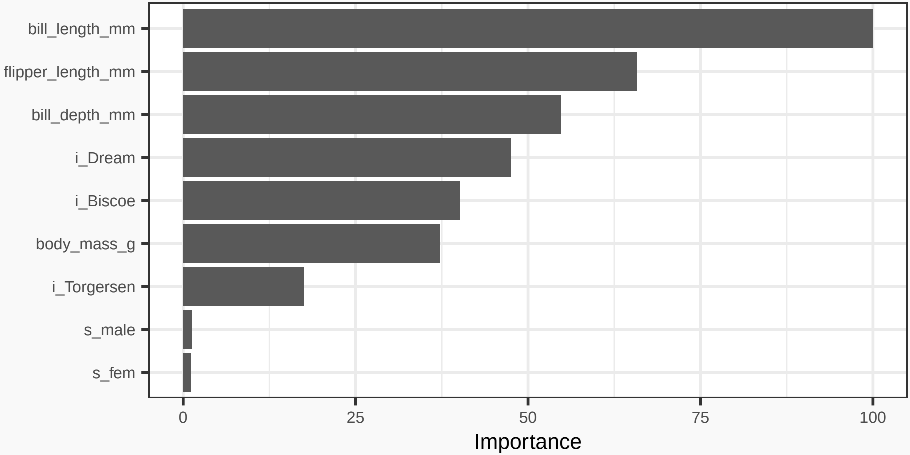
```r
# especificidade

spec(resultado_rf,
     truth = species,
     estimate = predicao_rf)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    macro          0.990
```

Exemplo

```r
# importancia das variaveis

pp_rf_final %>%
  extract_fit_parsnip() %>%
  vip(scale = TRUE)
```

Exemplo



bill_length_mm
flipper_length_mm
bill_depth_mm
i_Dream
i_Biscoe
body_mass_g
i_Torgersen
s_male
s_fem
0
25
50
75
100
Importance

Exercícios

Exercícios

O pacote `MASS` possui um conjunto de dados chamado `Pima.tr`. Este conjunto de dados possui informações a respeito de testes sobre diabetes realizados em mulheres da tribo Pima, dos Estados Unidos. Este conjunto de dados possui as seguintes variáveis:

- `npreg`: - número de gestações
- `glu`: - concentração de glicose
- `bp`: - pressão diastólica (mm Hg)
- `skin`: - medida da dobra do tríceps (mm)
- `bmi`: - índice de massa corporal
- `ped`: - diabetes pedigree function
- `age`: - idade (anos)
- `type`: - diabética ou não

Exercícios

Além disso, este mesmo pacote possui um outro conjunto de dados, com as mesmas colunas, chamado `Pima.te`.

1. Crie um novo conjunto chamado `Pima` unindo os dois conjuntos de dados originais. Utilize este novo conjunto de dados para criar os seus conjuntos de treinamento e teste.
2. Utilize o método random forest para criar um modelo para o diagnóstico de diabetes neste conjunto de dados.
3. Encontre as variáveis mais importantes do modelo ajustado.
4. Avalie se o modelo final é bom o suficiente na sua opinião. Justifique sua resposta.