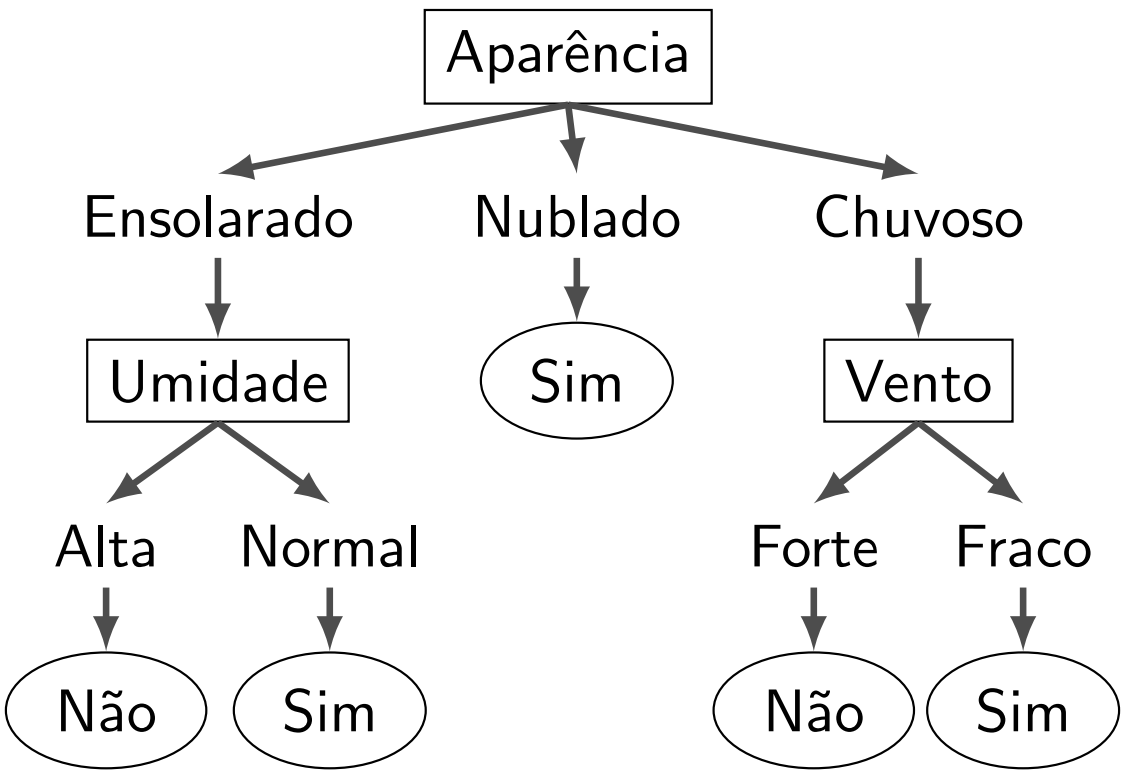
Árvores de decisão

Marcelo Keese Albertini
Faculdade de Computação
Universidade Federal de Uberlândia

16 de Outubro de 2018

Árvores de Decisão

- Nós internos
  - testam o valor de um atributo individual
  - ramificam de acordo com os resultados do teste
- Nós folhas
  - especificam a classe $h(\vec{\mathbf{x}})$
- Exemplo: jogar tênis?



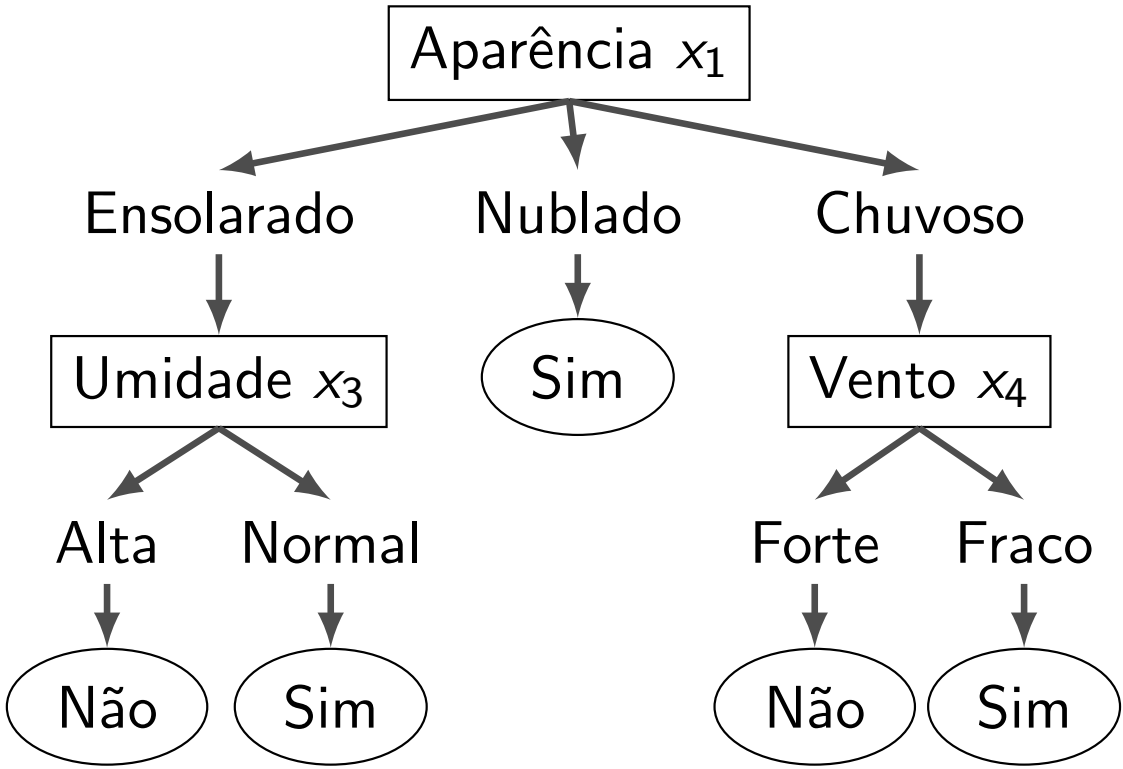
Aprendizado de Árvores de Decisão

Algoritmos de aprendizado para árvores de decisão

- Aprendizado
  - Árvore é construída pela adição de nós
- Constrói hipótese explicitamente (aprendizado impaciente)
- Qualquer função booleana pode ser representada por uma árvore de decisão

Aprendizado: escolha de atributo

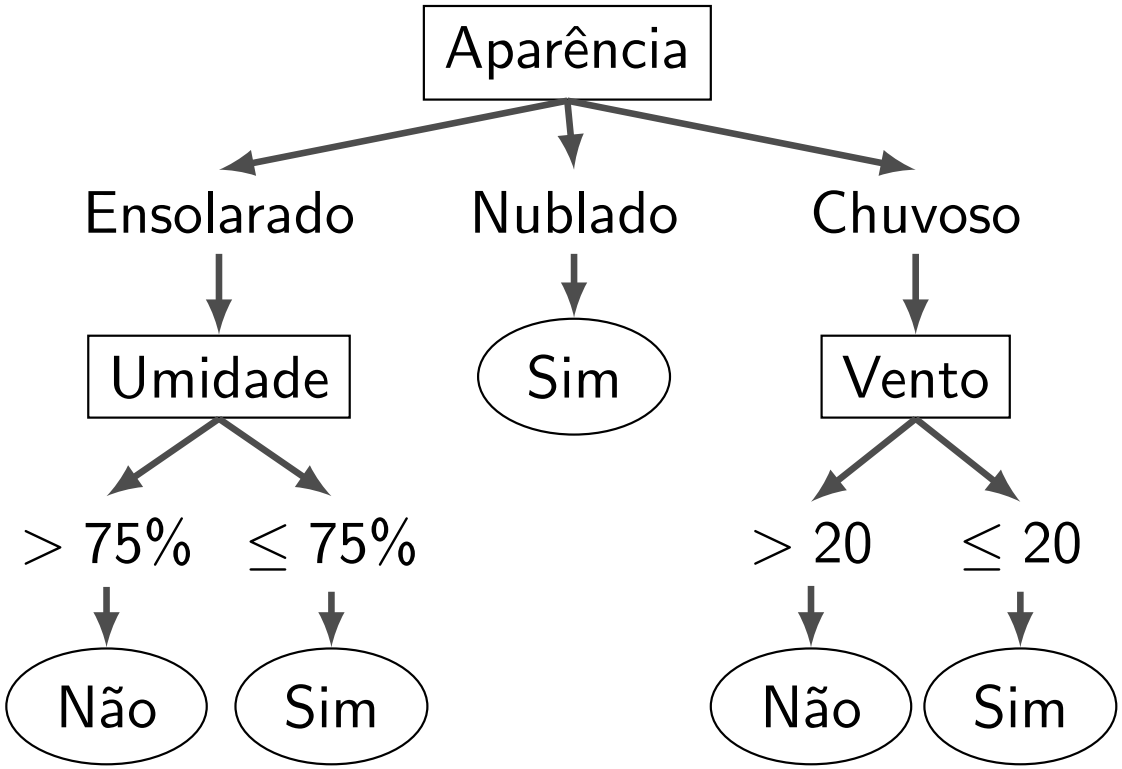
- Atributos são
  - Aparência: x_1
  - Temperatura: x_2
  - Umidade: x_3
  - Vento: x_4



$\vec{\mathbf{x}} = (x_1 = \mathit{Ensolarado}, x_2 = \mathit{Calor}, x_3 = \mathit{Alta}, x_4 = \mathit{Forte})$ será classificado como **Não**

- **temperatura** é irrelevante

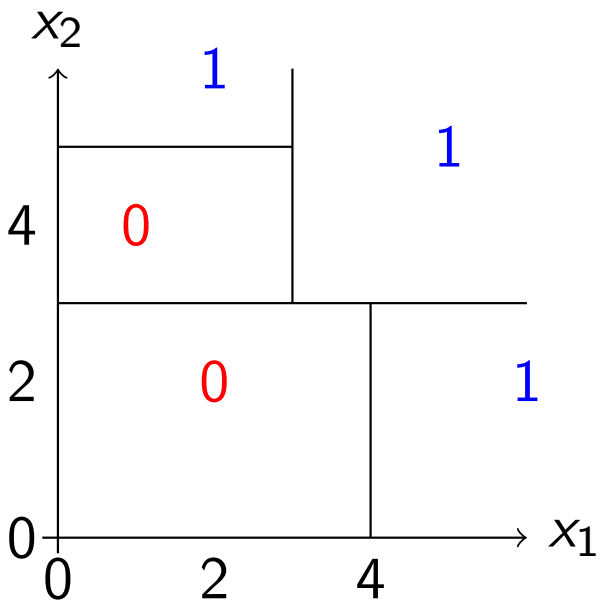
Aprendizado: atributo contínuo

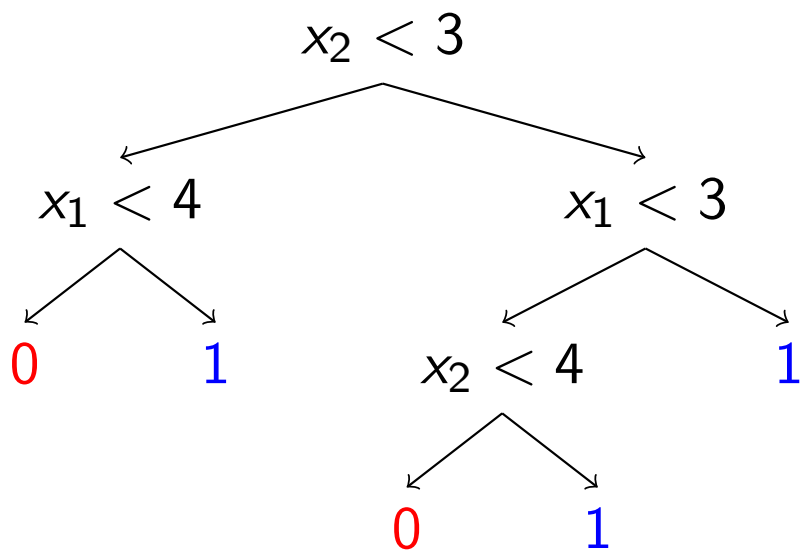


Se atributos são contínuos, nós internos podem testar o valor de um atributo em relação a um limiar.

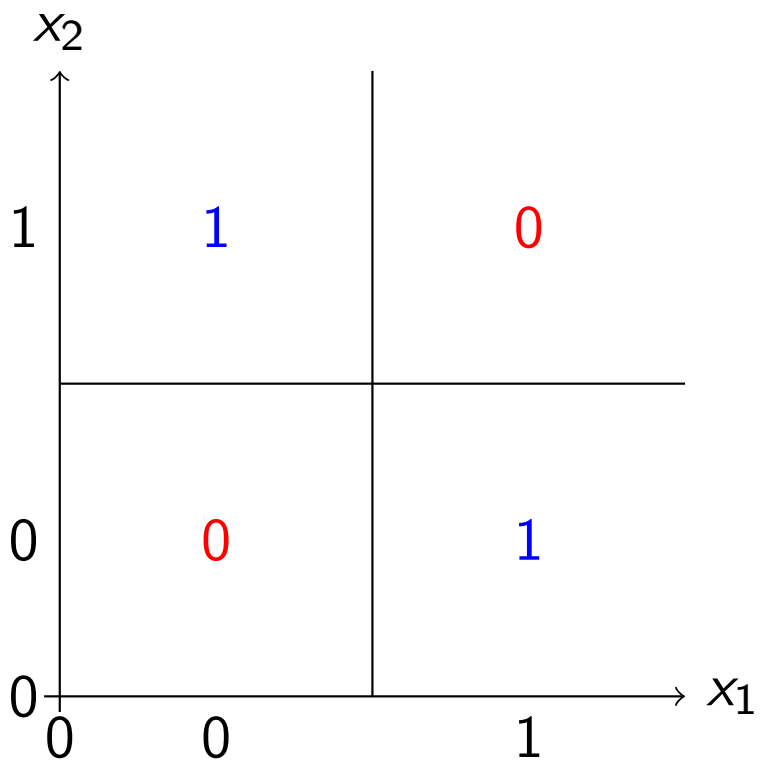
Formato de decisões de Árvores de Decisão

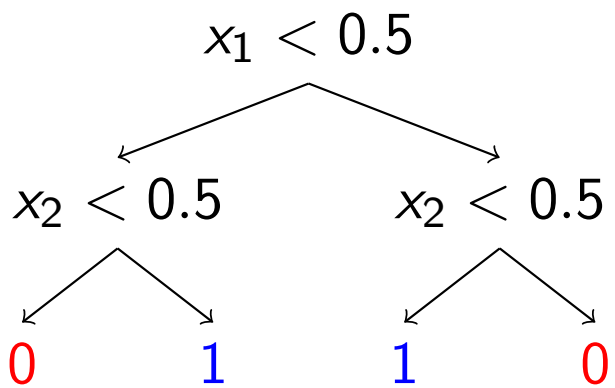
Árvores dividem o espaço de atributos em retângulos paralelos aos eixos e atribuem cada a uma das classes de decisão.





Árvores de decisão podem representar qualquer função booleana

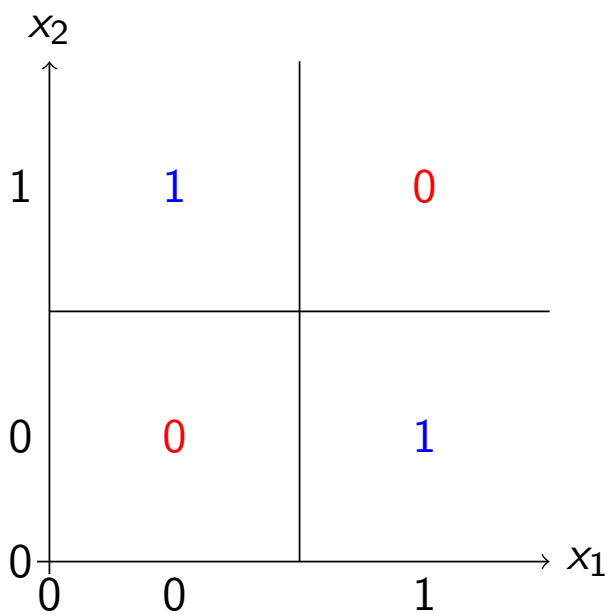


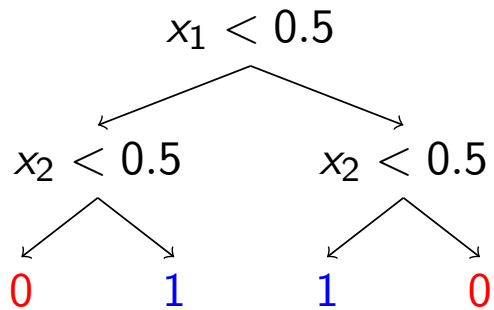


Árvores de decisão podem representar qualquer função booleana

Pior caso

A árvore pode exigir um número exponencial de nós em função do número de atributos.





x_1	x_2	x_3	x_4	y
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	1	0	0	0

Para cada valor de atributo é necessário criar uma ramificação.

Árvores de decisão têm espaço de hipóteses de tamanho variável

- Espaço de hipóteses: conjunto de combinações de elementos da linguagem de representação

Conforme o número de nós (ou altura) da árvore aumenta, o espaço de hipóteses cresce

- Altura 1 pode representar qualquer função booleana de 1 atributo
- Altura 2: qualquer função com 2 atributos e algumas com 3
  - exemplo: $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_3)$

Algoritmo de aprendizado para árvores de decisão

O mesmo algoritmo foi proposto por várias pessoas:

```
1  /** Entrada: S - conjunto de pares <x, y>,
2                sendo x atributos e y classes. */
3  Node criarArvore(S) {
4   if (y == 0 para todo <x, y> em S)
5       return Folha(0);
6   else if (y == 1 para todo <x, y> em S)
7       return Folha(1);
8   else {
9       j  = escolherMelhorAtributo(x, y);
10      S0 = Conjunto{<x, y> em S com x[j] == 0};
11      S1 = Conjunto{<x, y> em S com x[j] == 1};
12
13      A0 = criarArvore(S0);
14      A1 = criarArvore(S1);
15      return Node(x[j], A0, A1);
16  }
17 }
```

Escolha do melhor atributo: taxa de erro

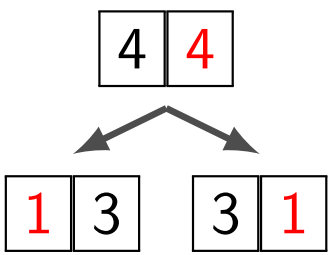
Uma forma é

fazer uma busca um passo adiante e escolher o atributo que resulta em menor taxa de erro nos exemplos de treino.

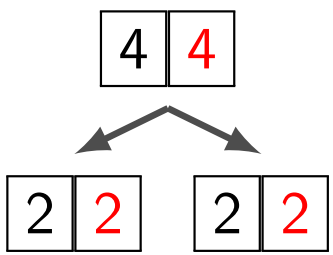
```
1  escolherAtributo(S) {
2    //escolher j para minimizar erros, da seguinte forma:
3    para cada atributo j calcular {
4      S0 = {<x, y> em S com x[j] == 0};
5      S1 = {<x, y> em S com x[j] == 1};
6
7      y0 = o valor mais comum de y em S0
8      y1 = o valor mais comum de y em S1
9
10     J0 = numero de exemplos <x, y> em S0 com y != y0
11     J1 = numero de exemplos <x, y> em S1 com y != y1
12 //  erros totais se dividirmos no atributo j
13     ERRO = J0 + J1
14   }
15   return (j com menor ERRO)
16 }
```

Exemplo: escolha do melhor atributo

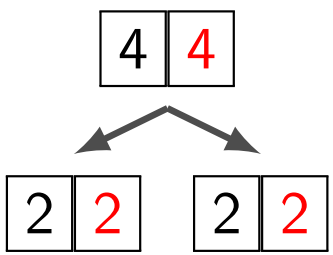
x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Em x_1, $J = 2$



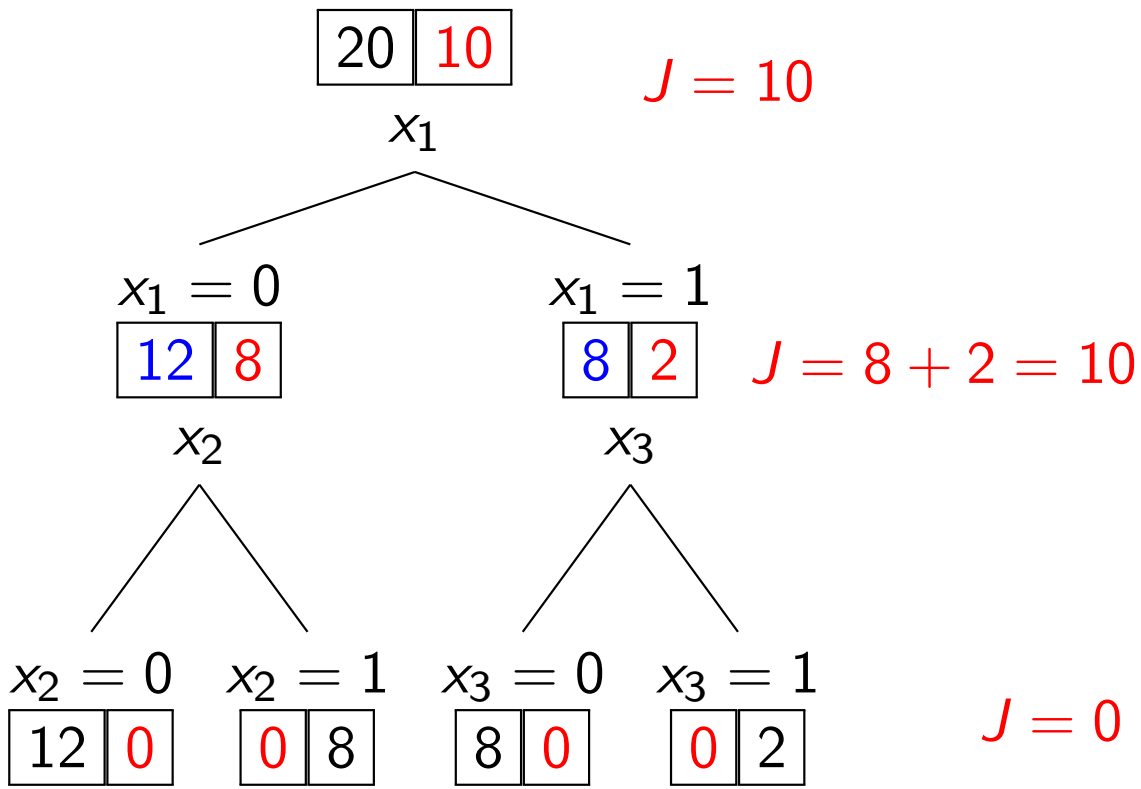
Em x_2, $J = 4$



Em x_3, $J = 4$

Exemplo: escolha do melhor atributo

A medida de erros nem sempre detecta casos em que podemos progredir na construção de uma boa árvore.



Uma heurística baseada em Teoria da Informação

Seja V uma variável aleatória com a seguinte distribuição de probabilidades

$P(V = 0)$	$P(V = 1)$
0.2	0.8

A **surpresa** $S(V = v)$ para cada valor $v \in V$ é:

$$S(V = v) = -\log_2 P(V = v)$$

Uma heurística baseada em Teoria da Informação

A **surpresa** $S(V = v)$ para cada valor de V é definido da forma:

$$S(V = v) = -\log_2 P(V = v)$$

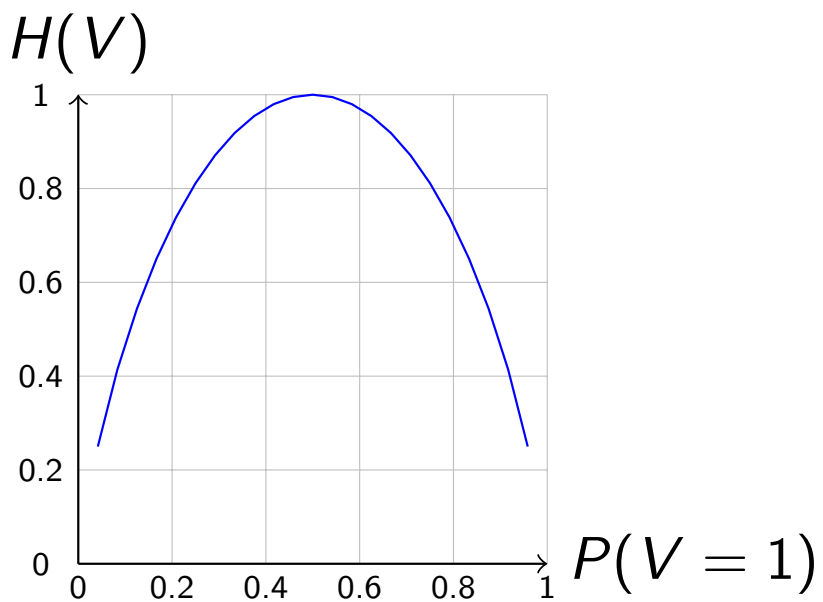
- A surpresa de ocorrer um evento com probabilidade 1 é 0.
- A surpresa de ocorrer um evento com probabilidade 0 é ∞.

Entropia

A entropia $H(V)$ do evento binário $V \in \{0, 1\}$ é definida por

$$H(V) = \sum_{v \in \{0,1\}} -P(V = v) \log_2 P(V = v)$$

que é a surpresa (incerteza) média de V.



Informação mútua

A informação mútua entre variáveis aleatórias A e B é a quantidade de informação que aprendemos sobre B ao saber do valor de A e vice-versa. Isso é calculado com a seguinte fórmula:

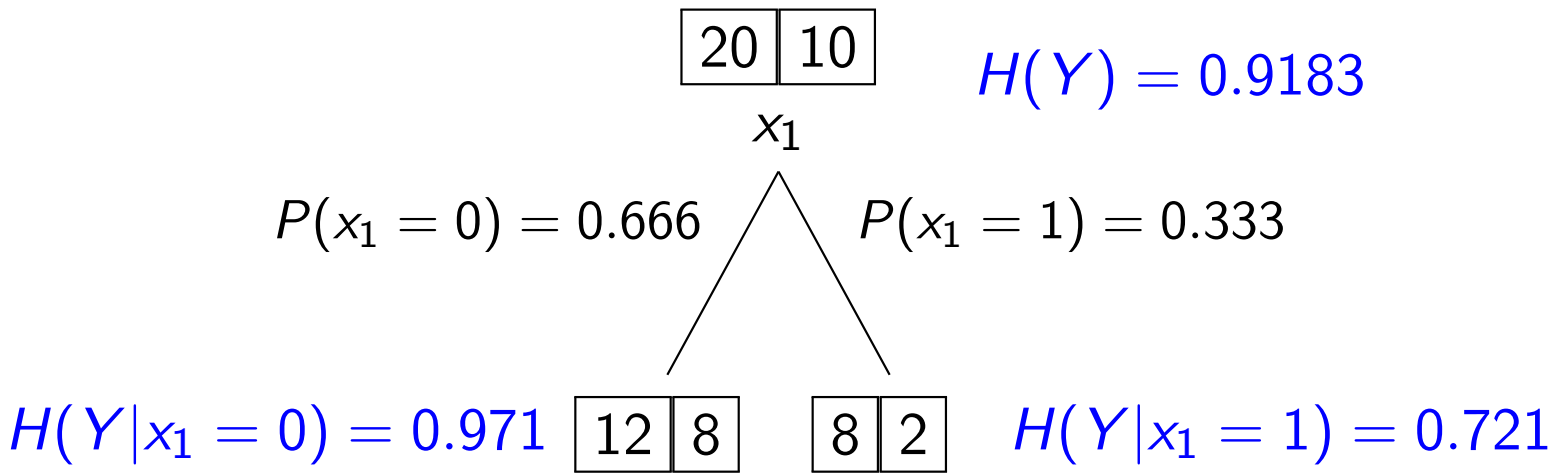
$$I(A;B) = H(B) - \sum_b P(B = b) \cdot H(A|B = b)$$

Informação mútua

A informação mútua entre A e B:

$$I(A;B) = H(B) - \sum_b P(B=b) \cdot H(A|B=b)$$

Considere que a classe Y, os valores de atributos X sejam variáveis aleatórias. Então, a informação mútua mede a utilidade um atributo de $X = x_1$ na decisão sobre a classe Y.



Visualizando heurísticas

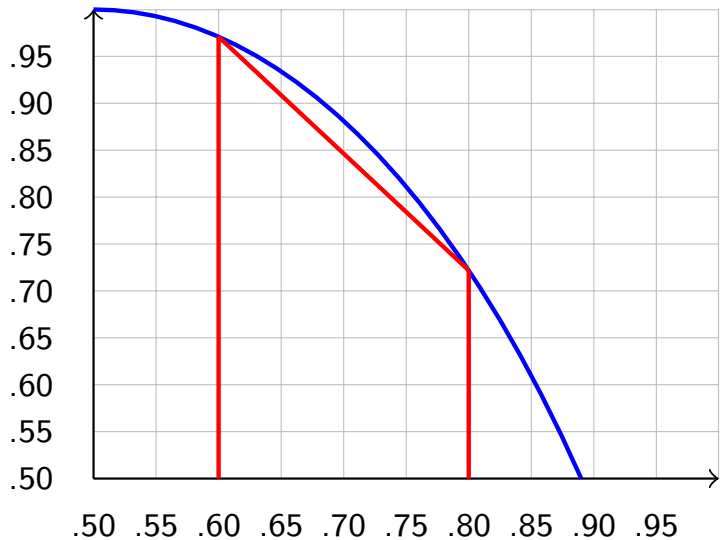


Figura: Entropia

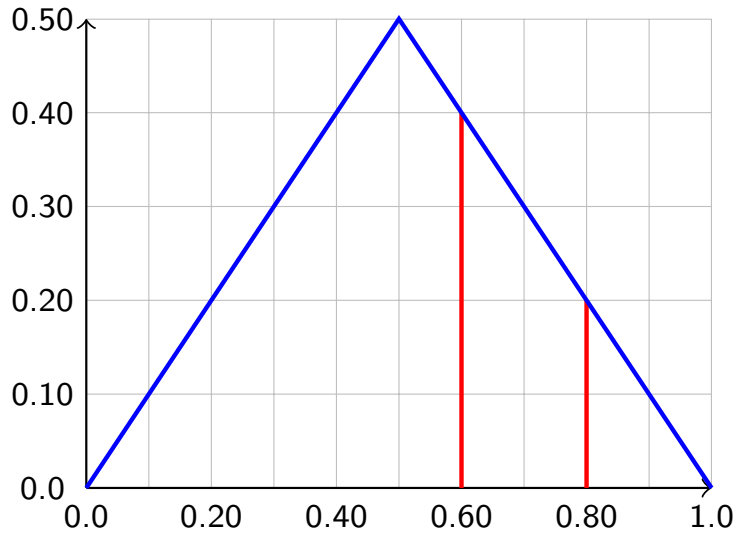


Figura: Erro absoluto

Informação mútua funciona porque é uma medida convexa. A linha da média de entropias está sempre abaixo da linha da entropia antes da divisão.

Atributos não-booleanos

- Atributos com múltiplos valores discretos
  1. Fazer uma divisão para cada valor
  2. Fazer uma divisão de um contra todos
  3. Agrupar valores em dois subconjuntos disjuntos
- Atributos contínuos
  - Procurar um limiar para dividir os valores do atributo
- Usar informação mútua para escolher a melhor divisão

Atributos com muitos atributos

Problema

- Se atributo tem muitos valores, Ganho (informação mútua) vai escolhê-lo
  - $Ganho(S, A) = I(S[y]; A)$
- Imagine usar `CPF` como atributo para aplicação de aprovação crédito

Taxa de ganho

- Usar TaxaDeGanho:

$$TaxaDeGanho(S, A) = \frac{Ganho(S, A)}{InfoDaDivisao(S, A)}$$

$$InfoDaDivisao(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- onde S_i é o subconjunto de S para qual A tem valor v_i
- Usar `InfoDaDivisao` para previlegiar atributos cuja divisão provê mais informação

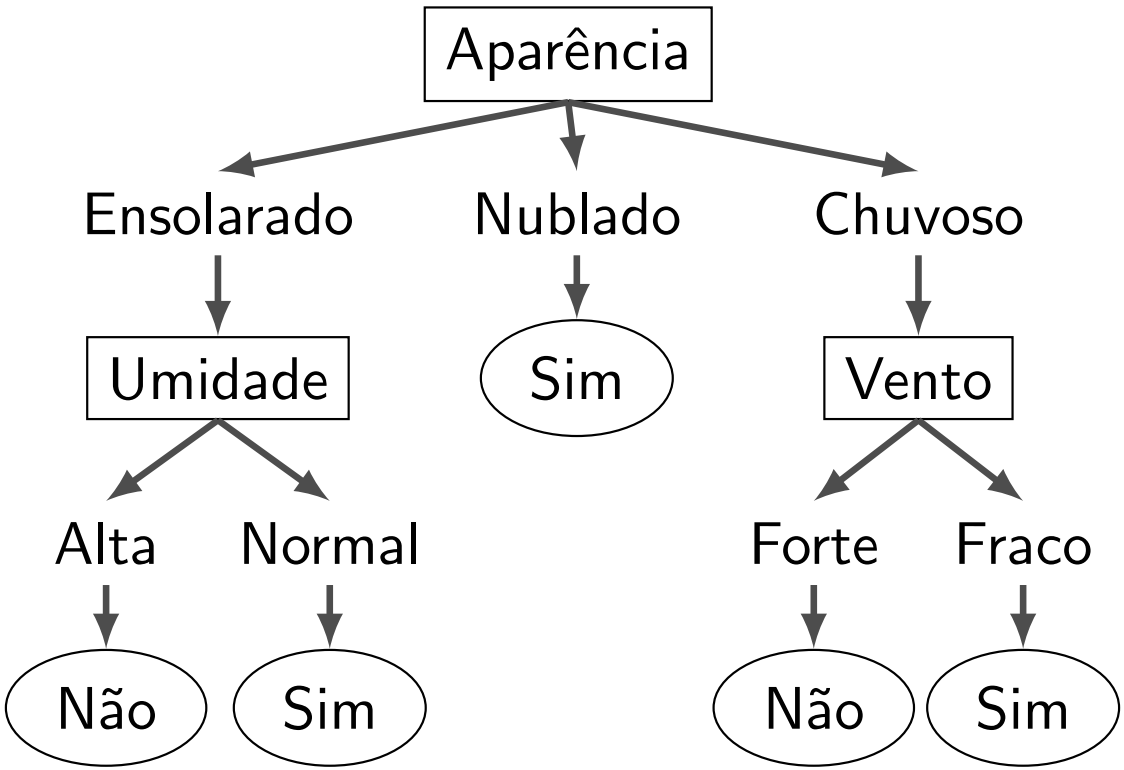
Atributos com valores desconhecidos

E se alguns exemplos têm atributos faltando?

- Usar o exemplo de treino mesmo assim, opções:
  - Se nó n testar atributo A, atribuir o valor mais comum de A entre os outros exemplos
  - Atribuir valor mais comum de A entre outros exemplos com mesmo valor alvo
  - Atribuir probabilidade p_i para cada possível valor v_i de A
  - Atribuir fração p_i do exemplo para cada descendente na árvore

Classificar novos exemplos na mesma maneira

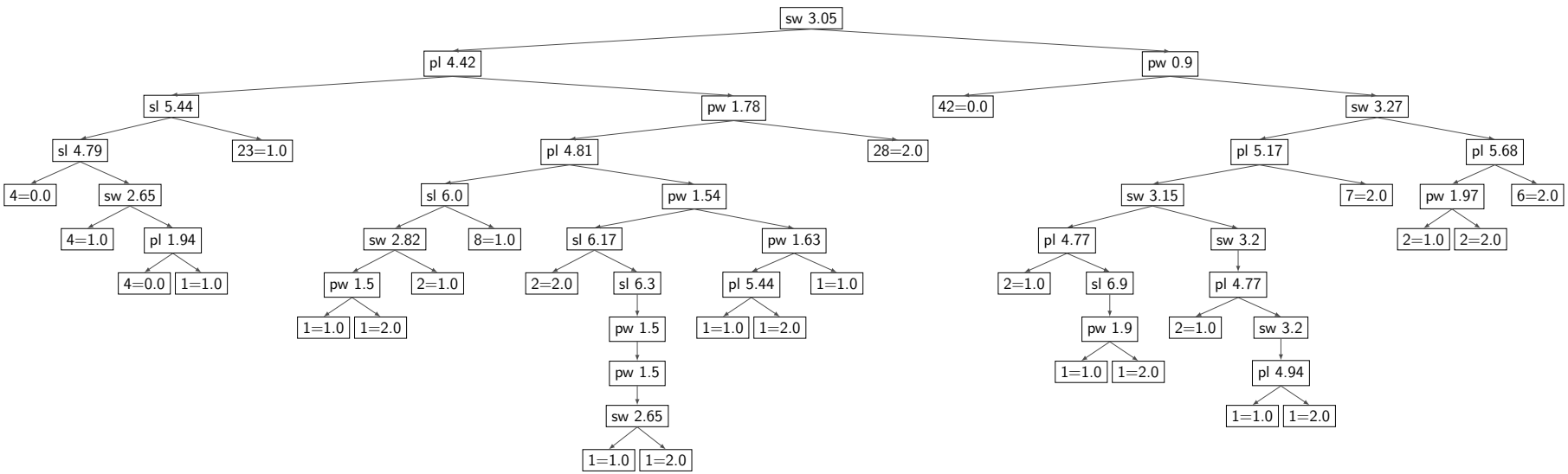
Overfitting (memorização) em árvores de decisão



Qual é o efeito na árvore

ao usar um exemplo problemático com ruído: `Ensolarado, Calor, Normal, Forte, JogarTênis=Não`.

Overfitting: Árvore que escolhe atributo aleatório - Conjunto Íris



Overfitting

Considere o erro de hipótese h sobre

- exemplos de treino: $\text{erro}_{\text{treino}}(h)$
- distribuição completa $\mathbf{D}$ dos dados: $\text{erro}_{\mathbf{D}}(h)$

Overfitting (memorização)

Hipótese $h \in H$ memorizou exemplos de treino se existe uma hipótese alternativa $h' \in H$ tal que

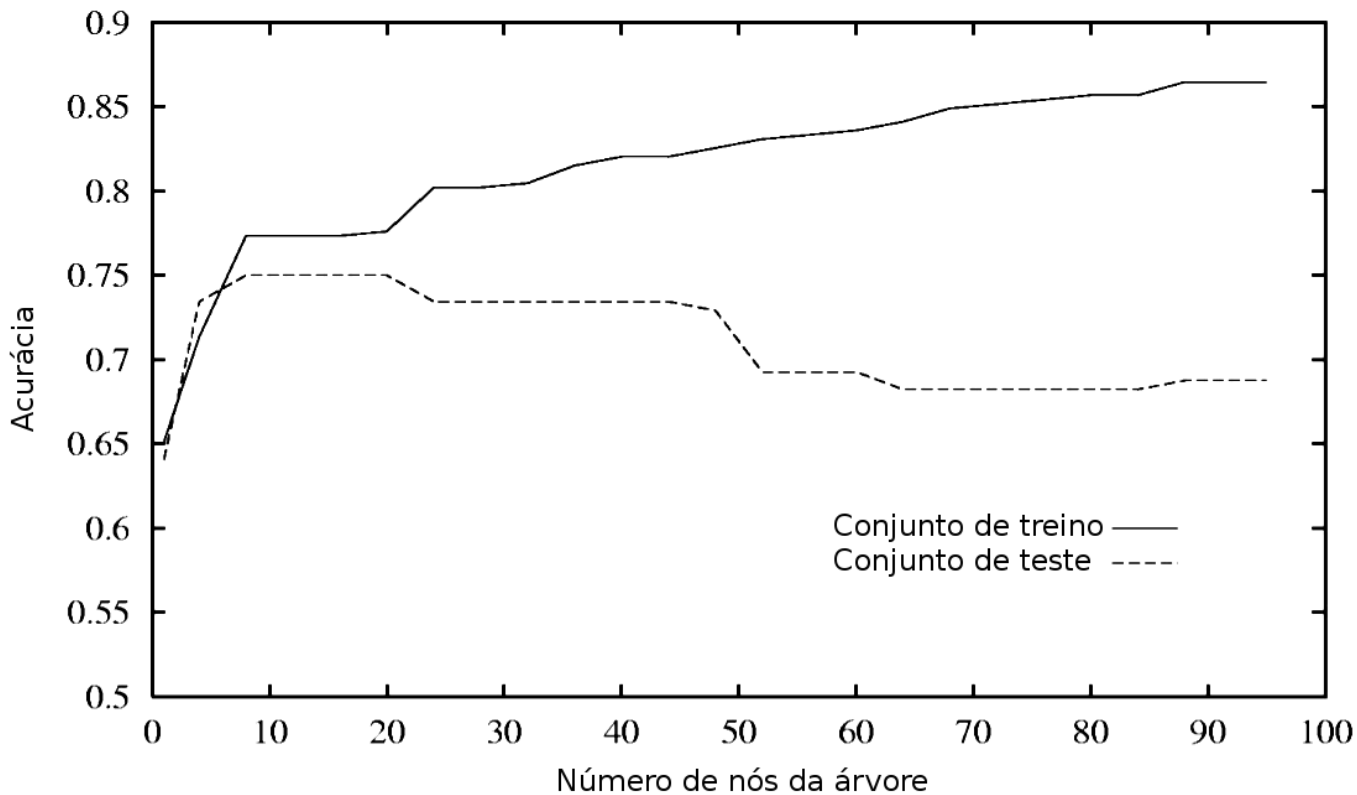
$$\text{erro}_{\text{treino}}(h) < \text{erro}_{\text{treino}}(h')$$

e

$$\text{erro}_{\mathbf{D}}(h) > \text{erro}_{\mathbf{D}}(h')$$

Overfitting durante o aprendizado de árvore de decisão

- A acurácia no conjunto de treino aumenta com maior número de nós
- Com maior número de nós, a acurácia no conjunto de teste diminui



Evitando overfitting

Como evitar overfitting

- Parar de crescer a árvore quando divisões não são estatisticamente significativas
- Construir árvore completa e depois podá-la

Como selecionar melhor árvore

- Medir desempenho nos exemplos de treino
- Medir desempenho em um conjunto de dados separado para validação
- Usar penalidade de complexidade para a medida de desempenho

Podagem de redução de erros

Separar exemplos em conjuntos de treino e validação

Podar enquanto não for prejudicial:

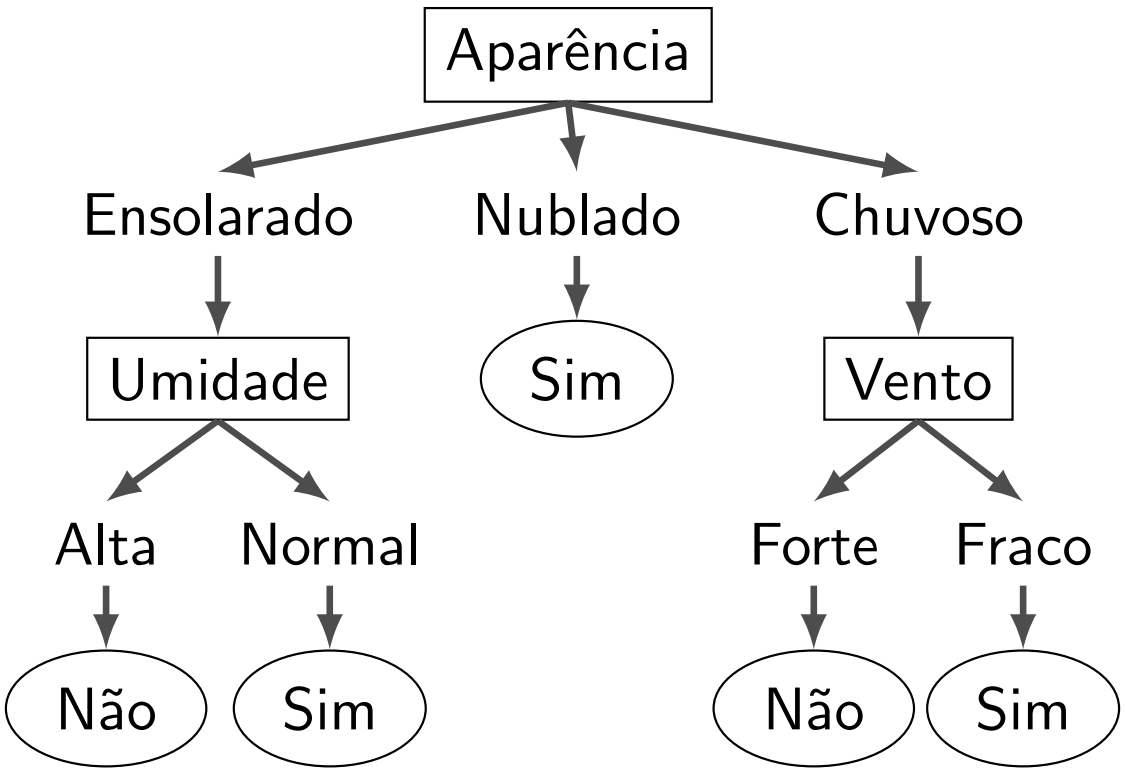
1. Medir com o conjunto de validação o impacto da podagem de cada nó possível
2. Podar o nó que melhorar mais a acurácia

Podagem de regras após geração da árvore

1. Converter árvore em conjunto de regras
   - Exemplo: obter uma regra para cada folha
2. Podar cada regra independentemente das outras
   - Exemplo: remover condição se isso melhorar a acurácia no conjunto de validação
3. Ordenar regras podadas na sequência de uso
   - Exemplo: usar regras com maior acurácia primeiro

Método frequentemente utilizado (exemplo, C4.5/J.48 Weka)

Conversão de uma árvore em regras



Algoritmos

- Iterative Dichotomiser 3 (ID3) de R. Quinlan
  - Entropia
  - Atributos discretos
- C4.5 (Weka J48) sucessor de ID3 de R. Quinlan
  - Ganho de informação normalizado
  - Atributos contínuos/numéricos
  - Poda
- C5.0 de R. Quinlan
  - Melhor corte de "fatores ordenados" e outros tipos de dados
  - Paralelismo = + rápido
  - Boosting
- Logistic Model Trees
  - Mistura de atributos para decisão

Árvores na prática: R

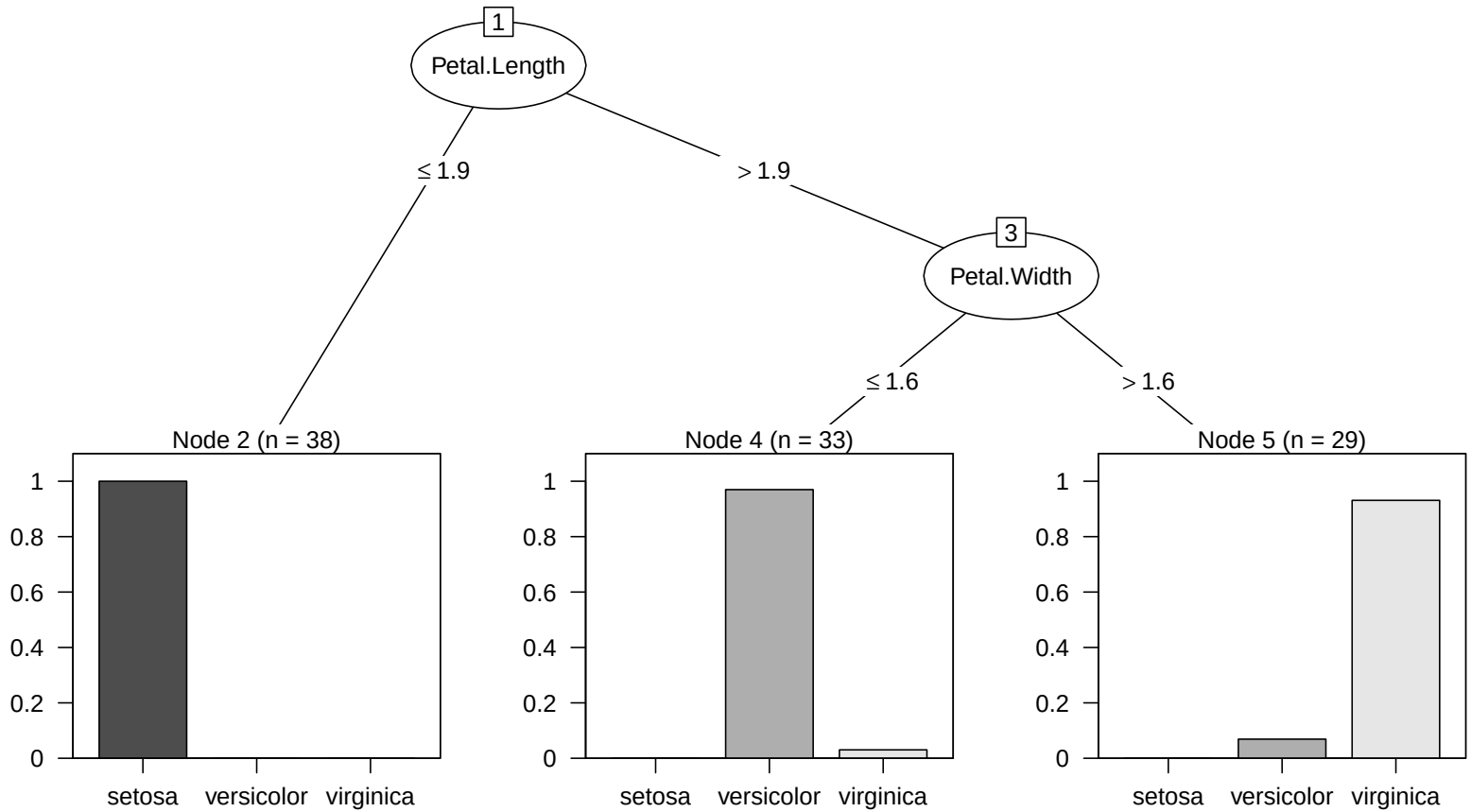
- Sucessora C4.5: `C50`
- Fast and Frugal Decision Trees: `FFTrees`
- Random Forest: `randomForest`
  - A Fast Implementation of Random Forests: `ranger`

Árvores C50

```
require(C50)
data(iris)
train.idx <- sample(1:nrow(iris), 100)
iris.train <- iris[train.idx, ]
iris.test  <- iris[-train.idx, ]
arvore <- C5.0(Species ~ ., iris.train)
arvore

##
## Call:
## C5.0.formula(formula = Species ~ ., data = iris.train)
##
## Classification Tree
## Number of samples: 100
## Number of predictors: 4
##
## Tree size: 3
```

Árvores C50: visualização

```
plot(arvore)
```



Árvores C50: desempenho

```
res <- predict(arvore, iris.train)
table(res, iris.train$Species)

##
## res           setosa versicolor virginica
##   setosa          38          0         0
##   versicolor       0         32         1
##   virginica        0          2        27

res <- predict(arvore, newdata=iris.test)
table(res, iris.test$Species)

##
## res           setosa versicolor virginica
##   setosa          12          0         0
##   versicolor       0         16         3
##   virginica        0          0        19
```

Árvores C50: extração de regras

```
regras <- C5.0(Species ~ ., iris.train, rules=TRUE)
regras

##
## Call:
## C5.0.formula(formula = Species ~ ., data =
##  iris.train, rules = TRUE)
##
## Rule-Based Model
## Number of samples: 100
## Number of predictors: 4
##
## Number of Rules: 3
##
## Non-standard options: attempt to group attributes
```

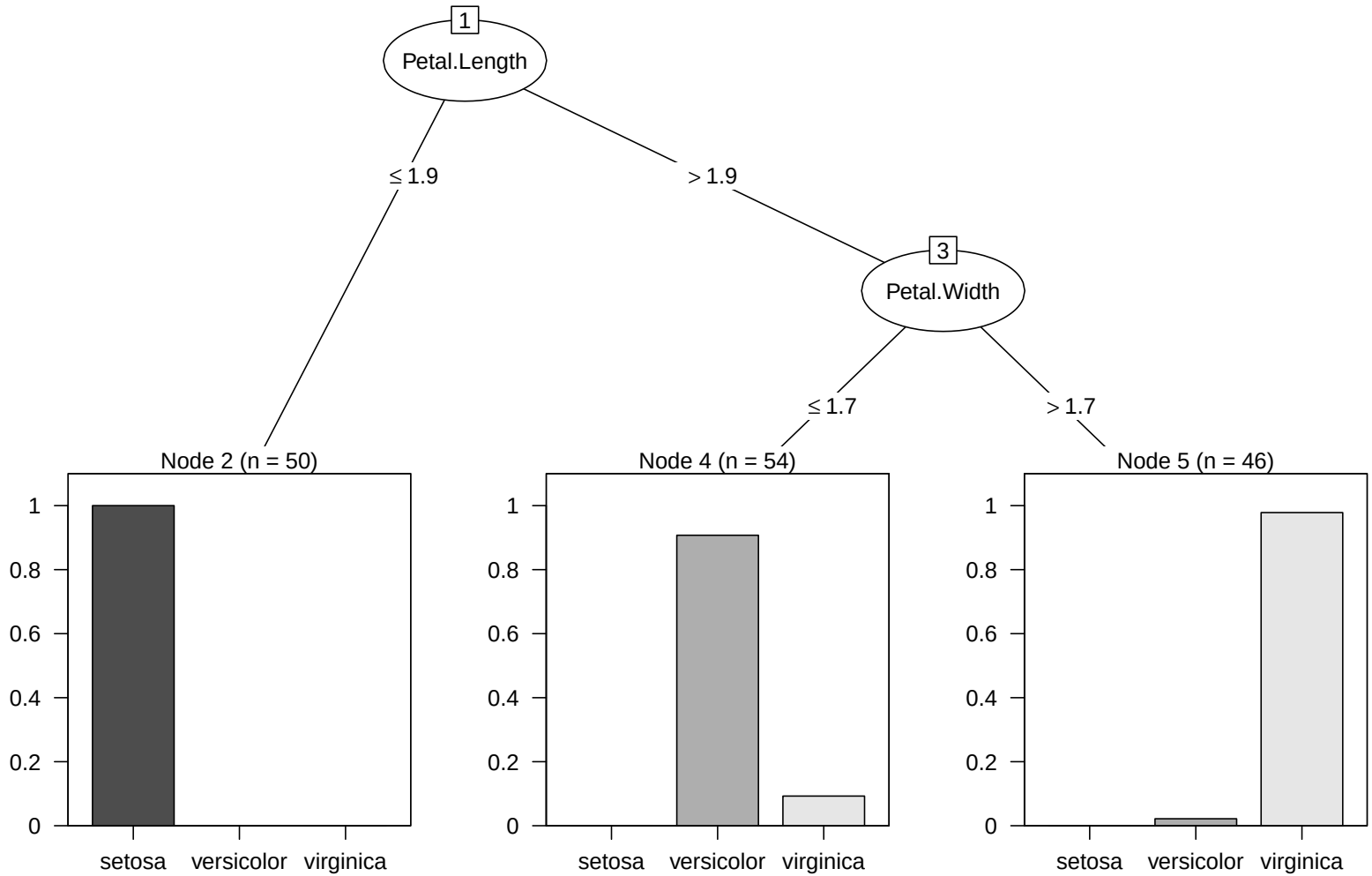
Árvores C50: desempenho de regras

```
table(predict(regras, iris.train), iris.train$Species)

##
##                setosa versicolor virginica
##   setosa           38          0         0
##   versicolor        0         32         1
##   virginica         0          2        27

table(predict(regras, iris.test), iris.test$Species)

##
##                setosa versicolor virginica
##   setosa           12          0         0
##   versicolor        0         16         3
##   virginica         0          0        19
```

Árvores C50: controle

```
cfg <-C5.0Control(
        subset = TRUE,#usar preditores discretos em splits?
        bands = 0, # agrupar regras em bands caso rules=T
        winnow = FALSE,# usar winnowing?
        noGlobalPruning = FALSE, # poda no final?
        CF = 0.5,  # fator de confianca
        minCases = 15, # minimo exemplos em folhas
        fuzzyThreshold = FALSE,
        sample = 0, # proporcao amostra de treino
        seed = 0, # aleatoriedade em empates
        earlyStopping = TRUE, # parar boosting
        label = "outcome")

arvore <- C5.0(Species ~ ., iris, control = cfg)
```

```
plot(arvore)
```


1
Petal.Length
≤ 1.9
> 1.9
3
Petal.Width
≤ 1.7
> 1.7
Node 2 (n = 50)
Node 4 (n = 54)
Node 5 (n = 46)
1
0.8
0.6
0.4
0.2
0
setosa
versicolor
virginica

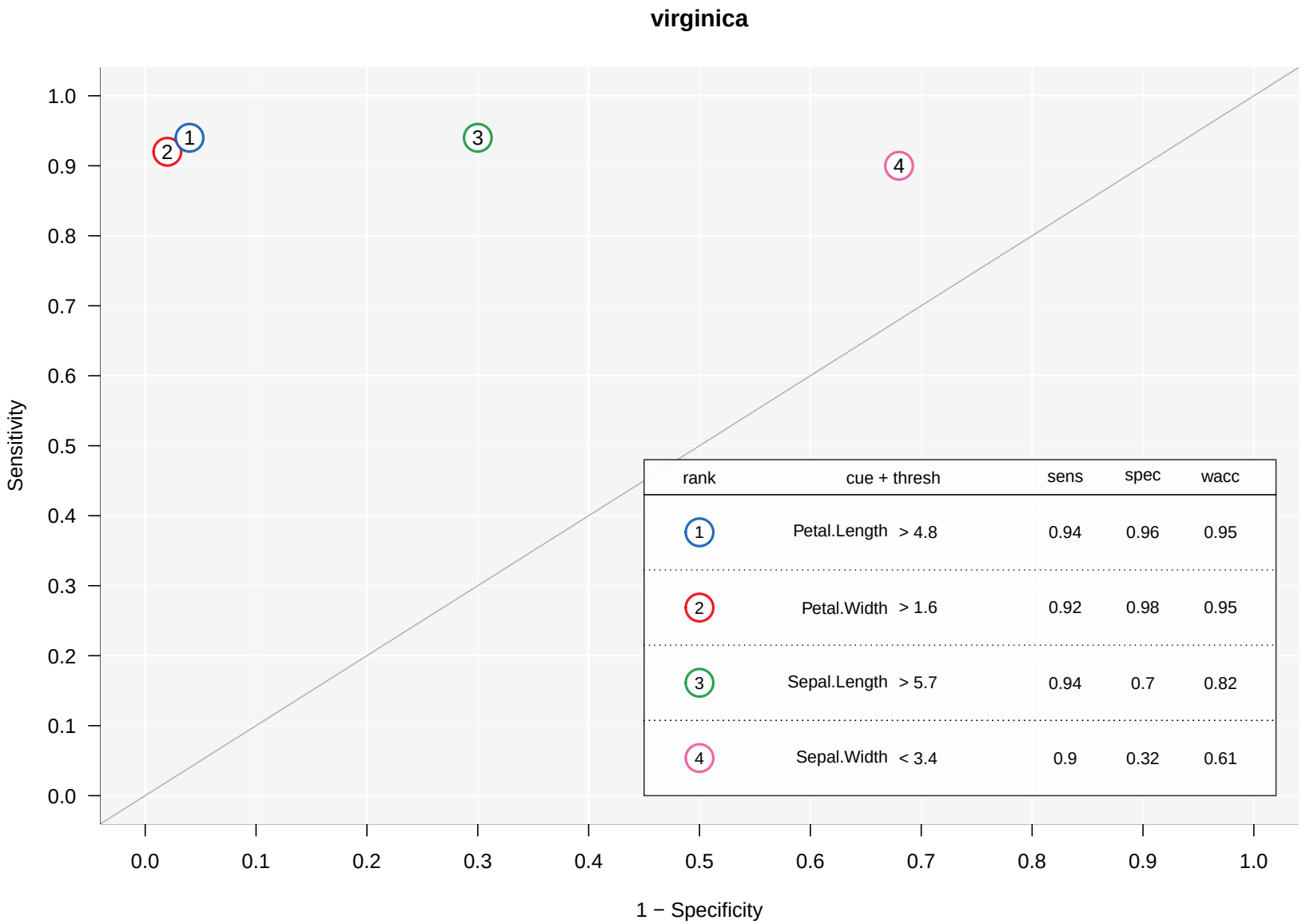
Árvores FFTrees

- Fácil uso: poucos atributos para decisão BINÁRIA

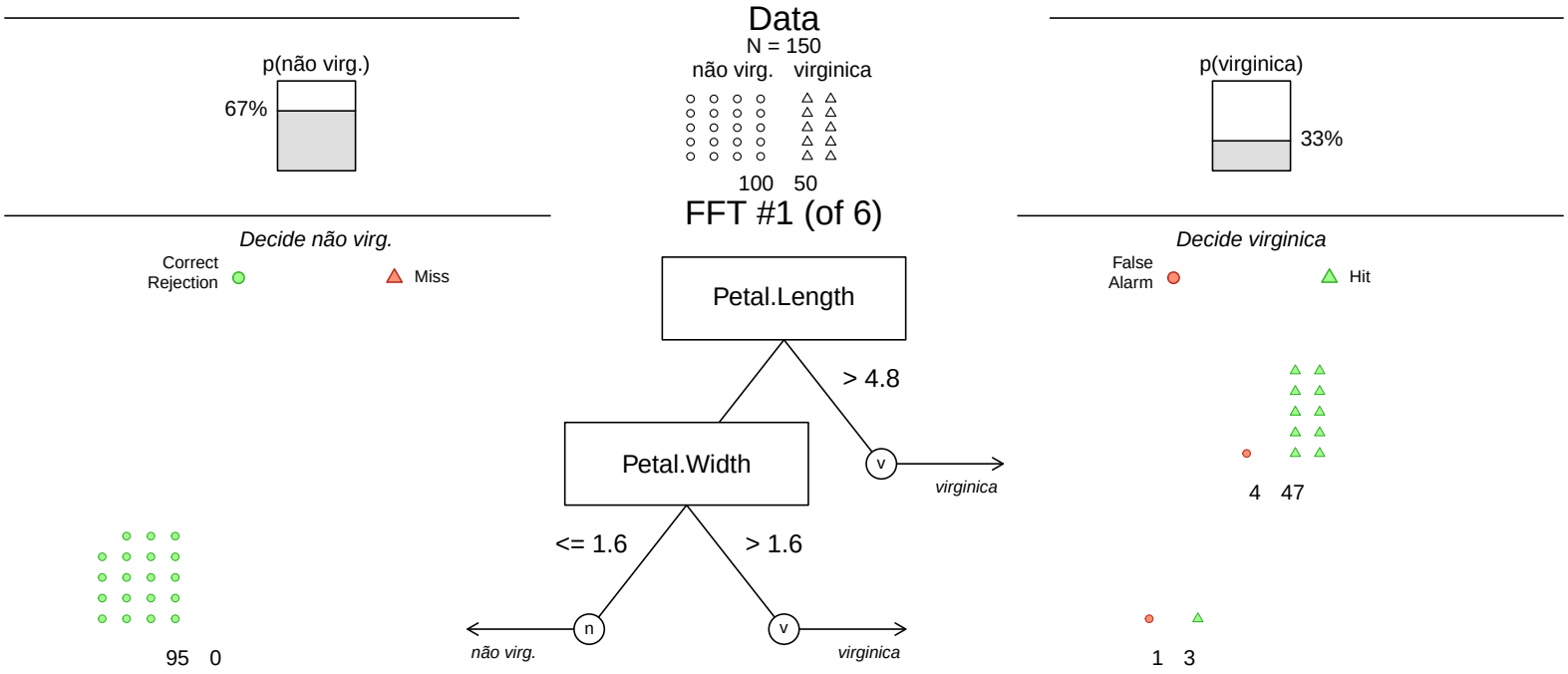
```
require(FFTrees) # Fast and Frugal Trees
irisBin <- iris
irisBin[,5] <- irisBin$Species == "virginica"
iris.fft <- FFTrees(Species ~ ., irisBin)
iris.fft

## FFT #1 predicts Species using 2 cues: {Petal.Length,Petal.Width}
##
## [1] If Petal.Length > 4.8, predict True.
## [2] If Petal.Width <= 1.6, predict False, otherwise, predict True.
##
##                    train
## cases       :n    150.00
## speed       :mcu    1.66
## frugality   :pci    0.67
## accuracy    :acc    0.97
## weighted    :wacc   0.98
## sensitivity :sens   1.00
## specificity :spec   0.95
##
## pars: algorithm = 'ifan', goal = 'wacc', goal.chase = 'bacc', sens.w = 0.5, max.le
```

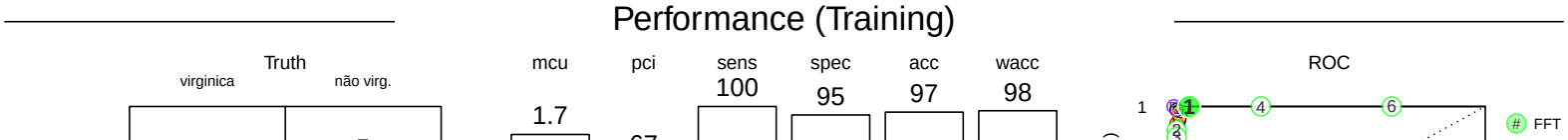
```
# cue == característica == atributo== descritor
# HR: hit rate, FAR: false alarm rate
iris.fft$cue.accuracies

## $train
##           cue   class threshold direction   n hi mi
## 1 Sepal.Length numeric       5.7         > 150 47  3
## 2  Sepal.Width numeric       3.4         < 150 45  5
## 3 Petal.Length numeric       4.8         > 150 47  3
## 4  Petal.Width numeric       1.6         > 150 46  4
##   fa cr sens spec       ppv       npv  far       acc
## 1 30 70 0.94 0.70 0.6103896 0.9589041 0.30 0.7800000
## 2 68 32 0.90 0.32 0.3982301 0.8648649 0.68 0.5133333
## 3  4 96 0.94 0.96 0.9215686 0.9696970 0.04 0.9533333
## 4  2 98 0.92 0.98 0.9583333 0.9607843 0.02 0.9600000
##   bacc wacc       bpv    dprime       cost cost.cue
## 1 0.82 0.82 0.7846469 2.0791741 0.22000000        0
## 2 0.61 0.61 0.6315475 0.8138528 0.48666667        0
## 3 0.95 0.95 0.9456328 3.3054597 0.04666667        0
```

```
showcues(iris.fft, main = "virginica")
```

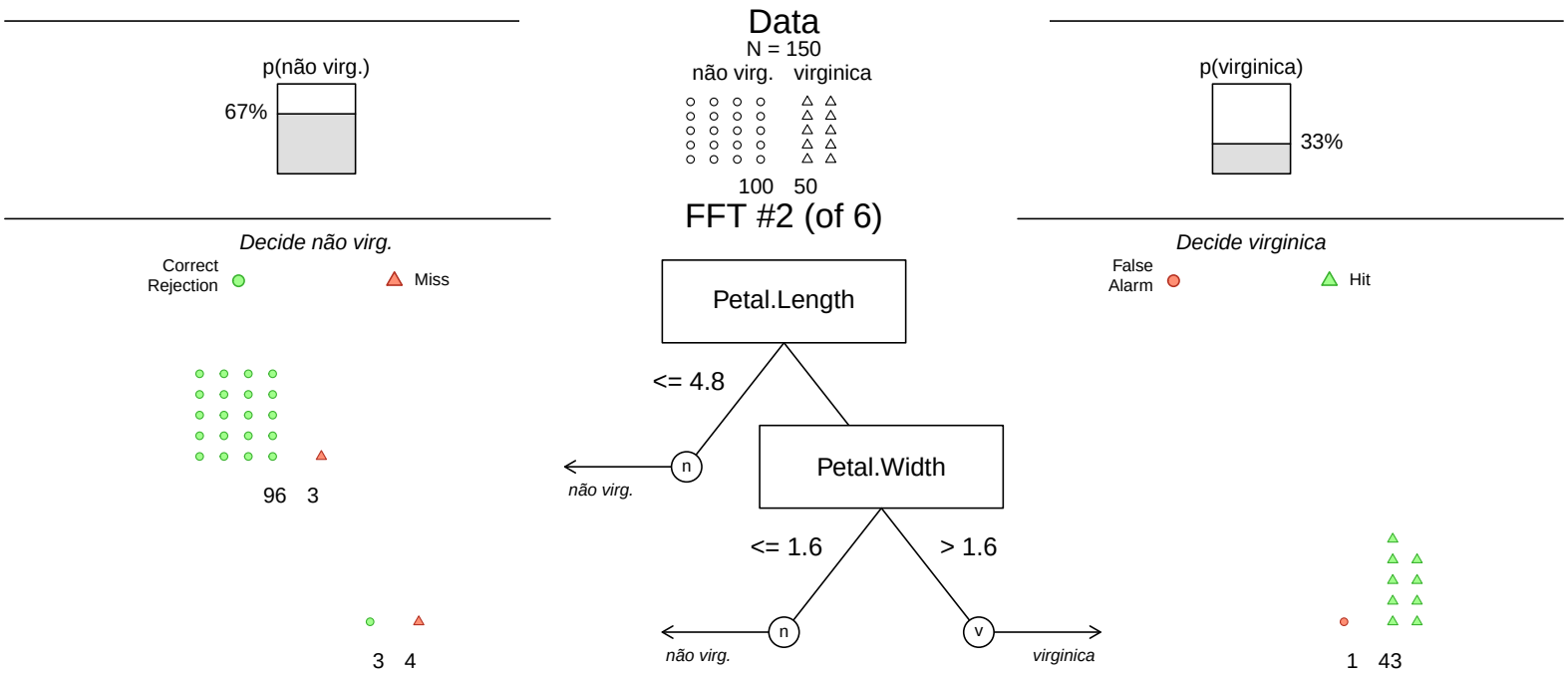


```
plot(iris.fft,  # spec == specificity == True Neg Rate
     data = "train",
     description = "Iris FFT",
     decision.names = c("não virg.", "virginica"))

## decision.names is depricated, use decision.lables instead
```





```
plot(iris.fft,  # spec == specificity == True Neg Rate
     data = "train", tree=2,
     description = "Iris FFT",
     decision.names = c("não virg.", "virginica"))

## decision.names is depricated, use decision.lables
instead
```



Florestas aleatórias: pacote `randomForest`

- Objetivo: evitar *overfitting*
- Aprendizado por ensemble/bagging com muitas árvores
- Selecionar atributos aleatoriamente
- Votação de decisões de diferentes árvores

```
require(randomForest)
iris.rFor <- randomForest(Species ~ ., iris.train)
importance(iris.rFor)
```

```
##                MeanDecreaseGini
## Sepal.Length           7.038522
## Sepal.Width            1.136503
## Petal.Length          26.329144
## Petal.Width           30.832627
```

Florestas aleatórias: pacote randomForest

```
iris.rFor

##
## Call:
##  randomForest(formula = Species ~ ., data = iris.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         38          0         0  0.00000000
## versicolor      0         31         3  0.08823529
## virginica       0          3        25  0.10714286
```

Florestas aleatórias: pacote randomForest

```
table(predict(iris.rFor, iris.test),iris.test$Species)

##
##                setosa versicolor virginica
##   setosa           12          0         0
##   versicolor        0         16         2
##   virginica         0          0        20
```

Florestas aleatórias: pacote randomForest

```
getTree(iris.rFor,143, labelVar=TRUE)
```

```
##    left daughter right daughter    split var
## 1              2               3  Petal.Width
## 2              0               0         <NA>
## 3              4               5  Petal.Width
## 4              6               7 Sepal.Length
## 5              8               9  Petal.Width
## 6              0               0         <NA>
## 7             10              11  Sepal.Width
## 8             12              13 Sepal.Length
## 9              0               0         <NA>
## 10            14              15 Petal.Length
## 11             0               0         <NA>
## 12            16              17 Petal.Length
## 13             0               0         <NA>
## 14             0               0         <NA>
```

Florestas aleatórias: pacote ranger

- A Fast Implementation of Random Forests: ranger

```
require(ranger)
iris.rgr = ranger(Species ~.,iris.train)
res = predict(iris.rgr, iris.test,type="response")
table(res$predictions, iris.test$Species)

##
##                setosa versicolor virginica
##   setosa           12          0         0
##   versicolor        0         16         2
##   virginica         0          0        20
```

Exercício

- Aplicar árvores no dataset "Ocorrências Aeronáuticas na Aviação Civil Brasileira"
  - Objetivo: prever o tipo de classificação da ocorrência
  - `http://www.cenipa.aer.mil.br/cenipa/Anexos/article/1451/ocorrencia.csv`
  - Maiores informações em [link]

Outros pacotes interessantes

- `rpart`: árvores CART com vários tipos de splits
- `tree`: pacote de uso simples e direto (CART)
- `evtree`: busca global para montar árvores
- `partykit`: infraestrutura unificada para árvores, visualização, critério de parada estatístico
- `CORElearn`: várias técnicas/algoritmos, paralelismo
- `varSelRF`: seleção de atributos com random Forests
- `maptree`: visualização, poda de árvores
- `REEMtree`: "regression trees with random effects"
- `Cubist`: regras de descisão com boosting