

# Tema 10

## Árvores de Decisão

Professora:

Ariane Machado Lima

# Terminologia Básica de Árvores

- Nós, raiz, pai, filhos, arestas
- Subárvore
- Caminho, ancestral/descendente
- Folhas/nós terminais, nós internos/não-terminais

Livro

C1

s1.1

s1.2

C2

s2.1

s2.1.1

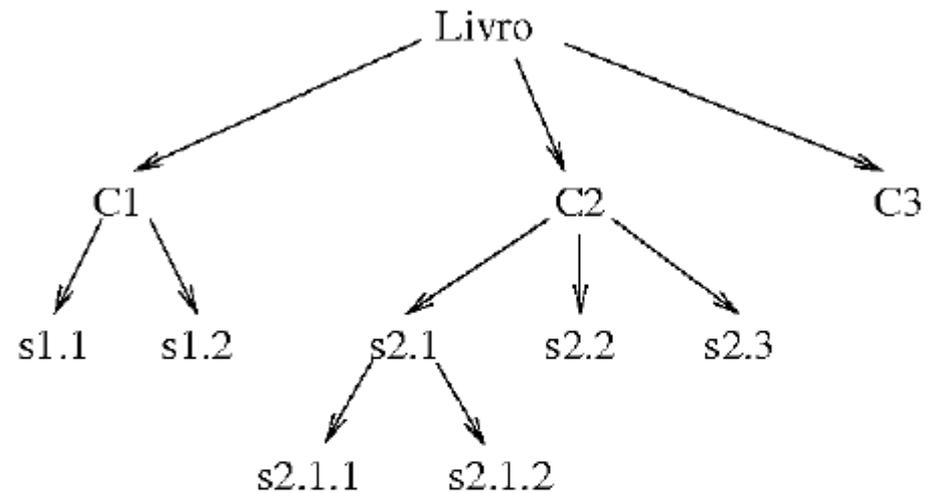
s2.1.2

s2.2

s2.3

C3

(a)



(b)

Exemplo: Sumário de um livro (a) e sua respectiva representação como uma árvore (b)

# Ávore de Decisão - Definição

Uma *Ávore de Decisão* é:

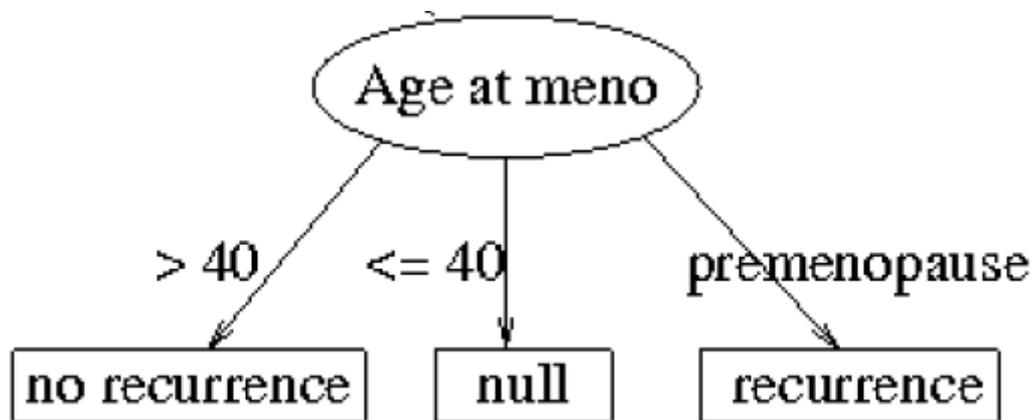
- um nó folha (ou nó resposta) que contém o nome de uma classe ou o símbolo *nulo* (*nulo* indica que não é possível atribuir nenhuma classe ao nó por não haver nenhum exemplo que corresponda a esse nó); ou

no recurrence

# Ávore de Decisão - Definição

Uma *Árvore de Decisão* é:

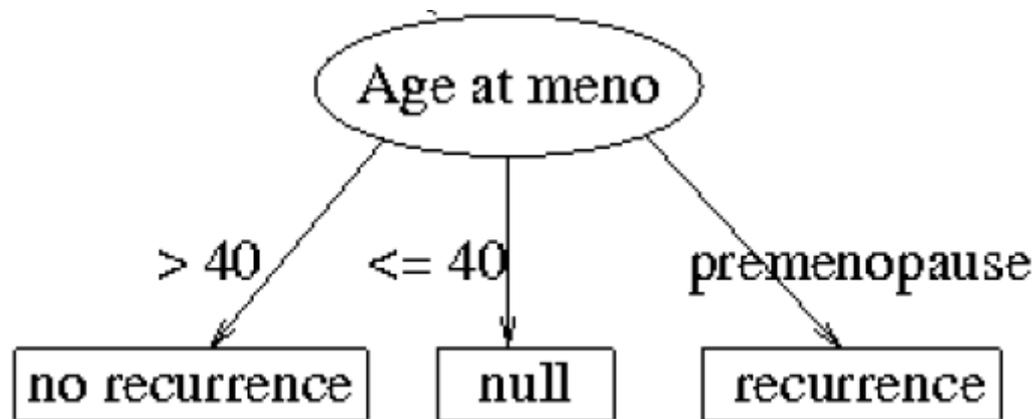
- um nó folha (ou nó resposta) que contém o nome de uma classe ou o símbolo *nulo* (*nulo* indica que não é possível atribuir nenhuma classe ao nó por não haver nenhum exemplo que corresponda a esse nó); ou
- um nó interno (ou nó de decisão) que contém o nome de um atributo; para cada possível valor do atributo, corresponde um ramo para uma outra árvore de decisão.



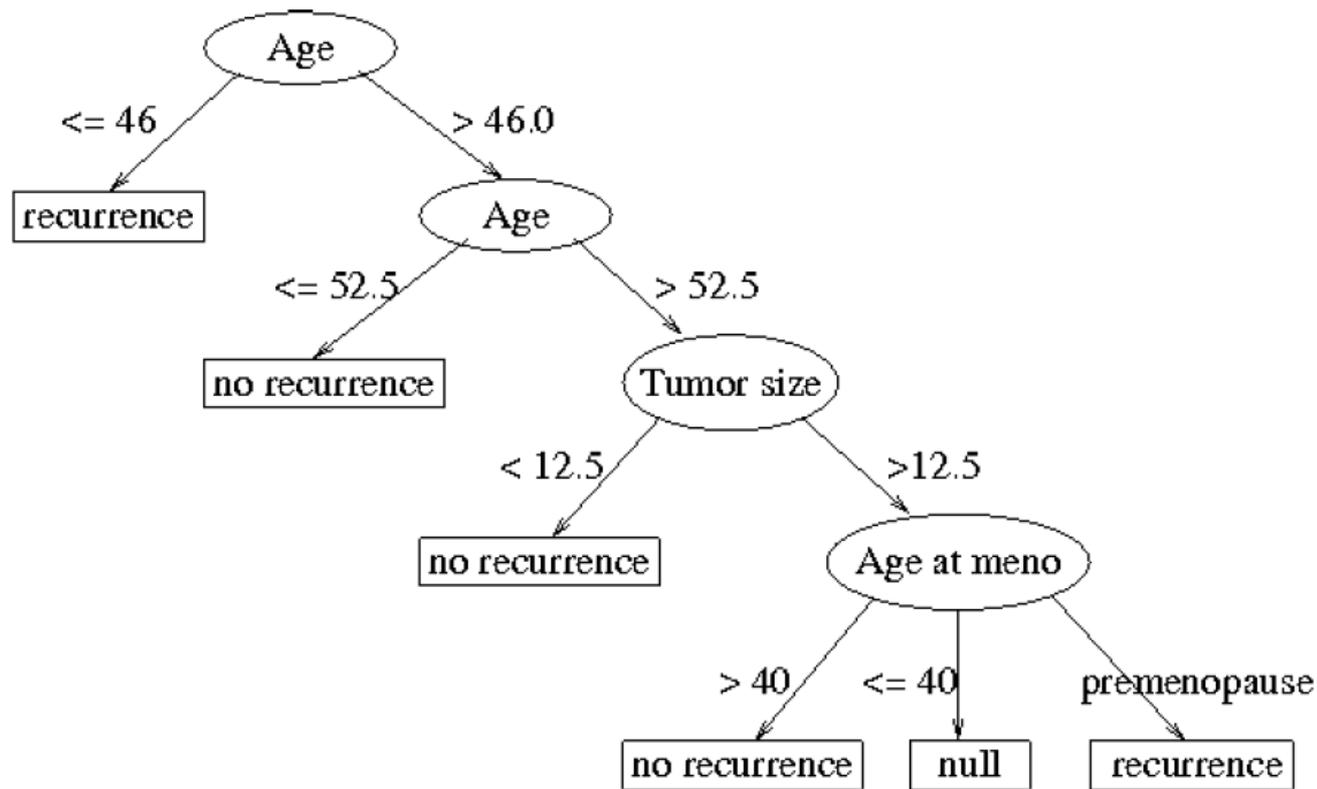
# Ávore de Decisão - Definição

Uma árvore de decisão possui a seguinte estrutura típica:

- Nós internos são rotulados com atributos;
- Folhas são rotuladas com classes;
- Ramos são rotulados com valores (atributos categóricos) ou com intervalos (atributos numéricos).



# Ávore de Decisão - Exemplo

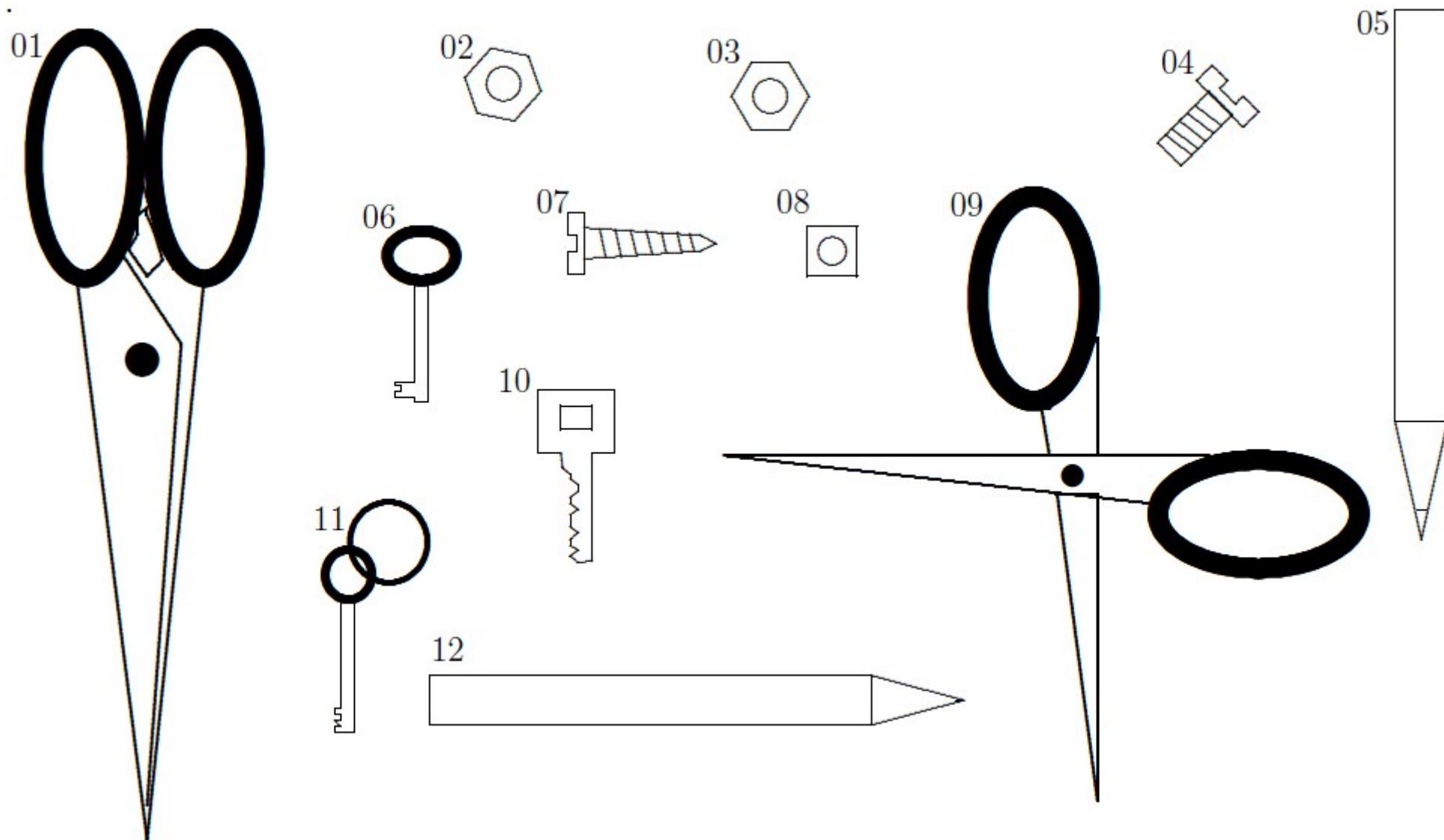


Exemplo: Predição de recorrência de câncer de mama

# Exemplo de construção da árvore

- Queremos construir uma árvore para classificar imagens de objetos nas classes: porca, parafuso, tesoura, chave, caneta
- Atributos considerados:
  - Tamanho: pequeno, grande
  - Formato: longo, compacto, outro
  - Orifícios: 0, 1, 2, 3, muitos

# Exemplo de construção da árvore – amostra de treinamento



# Exemplo de construção da árvore – amostra de treinamento

Objeto No.	TAMANHO	FORMATO	ORIFICIOS	CLASSE
01	grande	longo	2	tesoura
02	pequeno	compacto	1	porca
03	pequeno	compacto	1	porca
04	pequeno	longo	0	parafuso
05	grande	longo	0	caneta
06	pequeno	longo	1	chave
07	pequeno	longo	0	parafuso
08	pequeno	compacto	1	porca
09	grande	longo	2	tesoura
10	pequeno	longo	1	chave
11	pequeno	longo	2	chave
12	grande	longo	0	caneta

# Processo de construção da árvore (informal)

# Processo de construção da árvore (informal)

- Árvore com um nó, todos os exemplos associados a ele
- São todos os exemplo da mesma classe?
  - Sim: páre
  - Não:

# Processo de construção da árvore (informal)

- Árvore com um nó, todos os exemplos associados a ele
- São todos os exemplo da mesma classe?
  - Sim: páre
  - Não: subdivida os exemplos de acordo com um atributo (filhos da raiz)

# Processo de construção da árvore (informal)

- Árvore com um nó, todos os exemplos associados a ele
- São todos os exemplo da mesma classe?
  - Sim: páre
  - Não: subdivida os exemplos de acordo com um atributo (filhos da raiz)
    - Para cada nó criado:

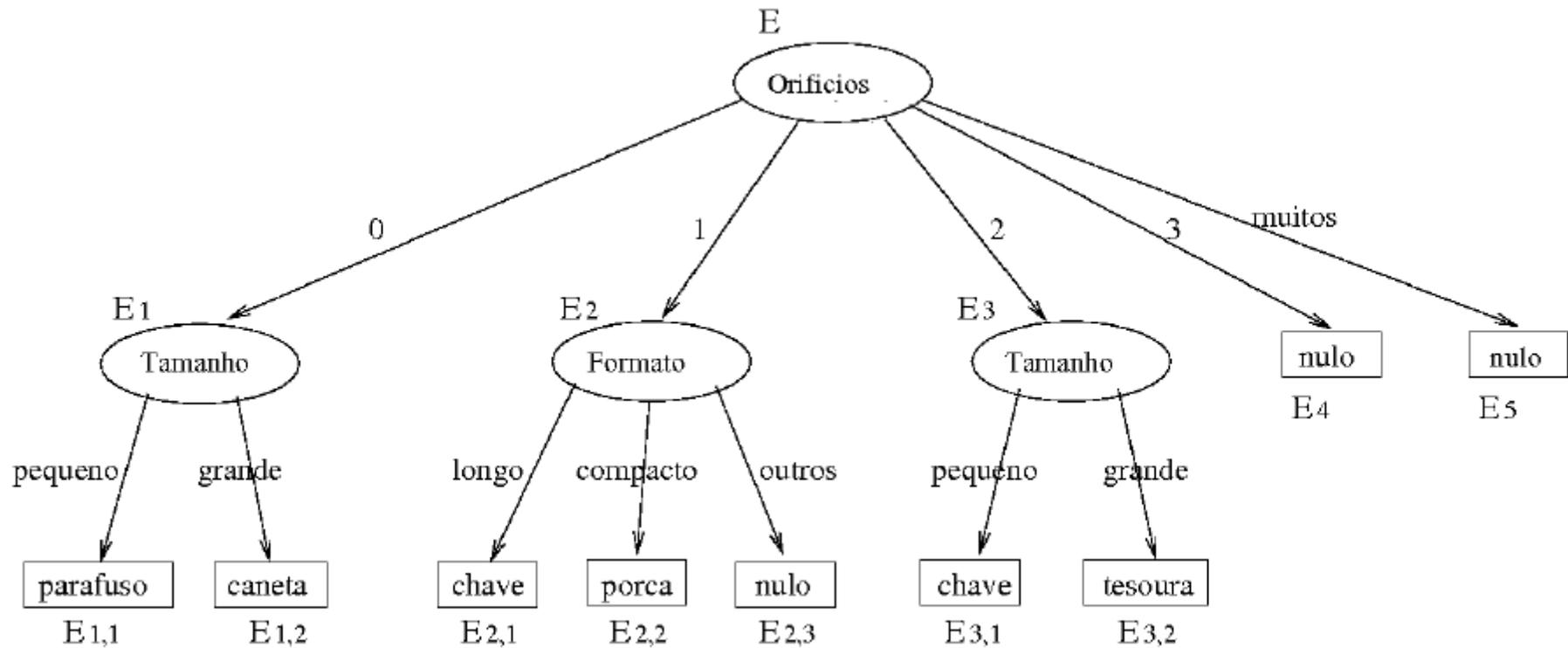
# Processo de construção da árvore (informal)

- Árvore com um nó, todos os exemplos associados a ele
- São todos os exemplo da mesma classe?
  - Sim: páre
  - Não: subdivida os exemplos de acordo com um atributo (filhos da raiz)
    - Para cada nó criado: são todos os exemplos da mesma classe?
      - Sim: páre
      - Não: subdivida os exemplos de acordo com outro atributo....

# Vamos tentar construir?

Objeto No.	TAMANHO	FORMATO	ORIFICIOS	CLASSE
01	grande	longo	2	tesoura
02	pequeno	compacto	1	porca
03	pequeno	compacto	1	porca
04	pequeno	longo	0	parafuso
05	grande	longo	0	caneta
06	pequeno	longo	1	chave
07	pequeno	longo	0	parafuso
08	pequeno	compacto	1	porca
09	grande	longo	2	tesoura
10	pequeno	longo	1	chave
11	pequeno	longo	2	chave
12	grande	longo	0	caneta

# Possível árvore



# Funcionamento (classificação)

- Classificando um novo objeto:  
(tamanho = grande, formato = longo, orifícios = 0)

# Problemas?

# Problemas?

- Várias árvores podem ser criadas dependendo da ordem de escolha dos atributos
  - Solução:

- 

-

# Problemas?

- Várias árvores podem ser criadas dependendo da ordem de escolha dos atributos
  - Solução: utilizar uma função de escolha do próximo atributo (por exemplo, que melhor divide os exemplos)
- -

# Problemas?

- Várias árvores podem ser criadas dependendo da ordem de escolha dos atributos
  - Solução: utilizar uma função de escolha do próximo atributo (por exemplo, que melhor divide os exemplos)
- A árvore pode ter muitos nós, alguns deles com poucos ou nenhum objeto
  -

# Problemas?

- Várias árvores podem ser criadas dependendo da ordem de escolha dos atributos
  - Solução: utilizar uma função de escolha do próximo atributo (por exemplo, que melhor divide os exemplos)
- A árvore pode ter muitos nós, alguns deles com poucos ou nenhum objeto
  - Solução: realizar podas na árvore ao final do treinamento

# Ideia do algoritmo de construção

- Expansão da árvore, através de sucessivas partições do conjunto de treinamento, até que a condição de parada seja satisfeita;
- Eliminação de algumas partes inferiores (*poda*) da árvore, através de reagrupamentos dos sub-conjuntos da partição.

**Dados** um conjunto de aprendizado  $E$  (dataset de treinamento)

uma condição de parada  $P(E)$

uma função de avaliação  $score(E, atributo)$  (avaliação do atributo)

uma função de classificação  $classe(t)$  (retorna a classe associada ao nó  $t$ )

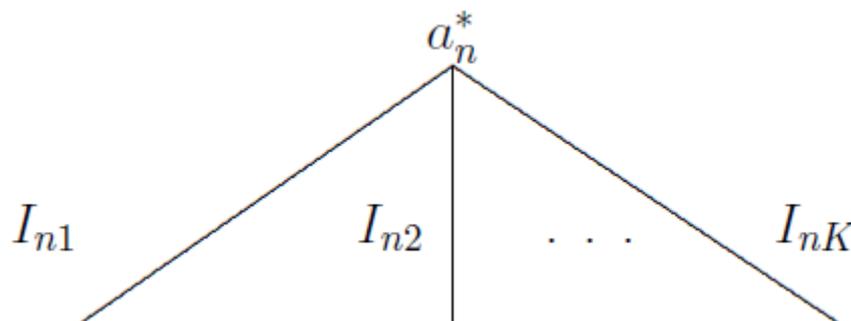
uma função de categorização  $categ(E, atributo)$  (para dividir em intervalos)

Se o conjunto  $E$  satisfaz a condição de parada  $P(E)$ ,

**então** a árvore resultante é um único nó  $t$  rotulado conforme a regra  $classe(t)$

**caso contrário**

1. para cada atributo  $a_i$ , calcule a função  $score(E, a_i)$ ;  
selecione o atributo  $a_n^* = \arg \max score(E, a_i)$
2. sejam  $I_{n1}, I_{n2}, \dots, I_{nK}$  os intervalos gerados segundo  $categ(E, a_n^*)$ ;  
crie o nó e os ramos



**Dados** um conjunto de aprendizado  $E$   
uma condição de parada  $P(E)$   
uma função de avaliação  $score(E, atributo)$   
uma função de classificação  $classe(t)$   
uma função de categorização  $categ(E, atributo)$

**Se** o conjunto  $E$  satisfaz a condição de parada  $P(E)$ ,  
**então** a árvore resultante é um único nó  $t$  rotulado conforme  
a regra  $classe(t)$

### **caso contrário**

1. para cada atributo  $a_i$ , calcule a função  $score(E, a_i)$ ;  
selecione o atributo  $a_n^* = \arg \max score(E, a_i)$
2. sejam  $I_{n1}, I_{n2}, \dots, I_{nK}$  os intervalos gerados segundo  $categ(E, a_n^*)$ ;  
crie o nó e os ramos
3. particione  $E$  em  $K$  sub-conjuntos  $E_1, E_2, \dots, E_K$  segundo os intervalos  
(ou valores) de  $a_n$  na árvore de decisão
4. aplique o algoritmo recursivamente para cada um dos subconjuntos  $E_i$
5. após a expansão da árvore ter terminado, ajuste seu tamanho,  
eliminando alguns ramos  $T_t$  se necessário.

# Principais elementos do algoritmo

- Regra de parada  $P(E)$ : é uma condição que o conjunto de treinamento  $E$  deve satisfazer para que seu nó correspondente seja considerado um nó terminal.

Casos mais simples:  $E$  é vazio ou quando todos os exemplos incidentes sobre  $t$  pertencem à mesma classe.

Contudo, existem algoritmos que interrompem a expansão do nó sob condições mais complexas. Nesses casos, dizemos que ocorre uma *pré-poda* da árvore de decisão.

- Regra  $classe(t)$ : função que atribui um rótulo de classe a um nó  $t$ .

Critério mais simples: classificação por maioria.

- Função  $score(a_i)$ : é utilizada para tentar identificar o atributo mais relevante existente sobre  $E$ , isto é, o atributo com maior poder discriminante das classes.

# Principais elementos do algoritmo

- Método de categorização de atributos  $categ(E, a)$ : Atributos numéricos precisam ser “discretizados” em sub-intervalos do tipo

$$\begin{aligned} I_{n1} &= ]c_{n0}, c_{n1}] \\ I_{n2} &= ]c_{n1}, c_{n2}] \\ &\vdots \\ I_{nK} &= ]c_{nK-1}, c_{nK}[ \end{aligned}$$

onde  $c_{n0}, c_{n1}, \dots, c_{nK}$  são constantes definidas segundo o critério de categorização  $categ(E, a)$ . A forma mais simples de discretização é a binarização dos atributos, através de condições do tipo  $x_n \leq c?$ , onde a constante  $c$  é obtida através de testes exaustivos sobre o conjunto de treinamento disponível.

Uma vez que os atributos categóricos também podem ser reduzidos a conjuntos  $I_{n1} = \{a_{n1}\}, \dots, I_{nK} = \{a_{nK}\}$ , para simplificar a notação não faremos distinção entre atributos ordenados e categóricos, exceto em casos explícitos.

# Principais elementos do algoritmo

- Critério de poda: Após a expansão da árvore, pode ocorrer que os subconjuntos de  $E$  correspondentes aos nós folhas sejam muito pequenos e com grande efeito de *overfitting* (ajuste muito preciso aos dados de treinamento, porém com baixa precisão na classificação de novos exemplos).

É necessário então eliminar-se alguns dos ramos inferiores da árvore, de forma a atenuar o efeito dos ruídos (informações inúteis para a classificação) e de forma a manter na árvore apenas regras com mais alto poder de discriminação das classes.

Esse processo é denominado *pós-poda* (ou simplesmente *poda*) das árvores.

# Vendo mais em detalhes...

- Função classe( $t$ )
- Função score( $E, a$ )
- Poda

# Função classe(t)

- Atribui uma classe ao nó  $t$  com base nos exemplos que lá “estão”
- Possíveis funções:
  - Critério de minimização do erro esperado
  - Critério de minimização do custo do erro de classificação
  - Rotulação dos nós pelas probabilidades das classes

# Função classe( $t$ )

Denotaremos por  $p(j|t)$  a probabilidade estimada de um exemplo incidente em  $t$  possuir classe  $j$ . Por exemplo,  $p(j|t)$  pode ser definido como a proporção de exemplos de  $t$  que possuem classe  $j$ .

# Função classe( $t$ ): Critério de minimização do erro esperado

O critério mais simples de atribuição é adotar a classe de maior probabilidade estimada no nó, isto é, a classe que maximiza  $p(j|t)$ . Assim, a probabilidade estimada de um exemplo incidente sobre  $t$  ser classificado erroneamente é:

$$\begin{aligned} r(t) &= 1 - \max_j p(j|t) \\ &= \min_j \sum_{i \neq j} p(i|t) \end{aligned} \tag{1}$$

# Função classe( $t$ ): Critério de minimização do custo do erro de classificação

O *custo de erro* em classificar um objeto pertencente à classe  $j$  como sendo de classe  $i$  é denotado por  $C(i, j)$  e satisfaz

- $C(i, j) \geq 0, i \neq j$ ;
- $C(i, i) = 0$ .

Se um objeto de classe desconhecida incidente sobre um nó  $t$ , for classificado com classe  $i$ , então o custo esperado do erro na classificação será  $\sum_j C(i, j)p(j|t)$ . Assim, um critério natural de atribuição é escolher a classe  $i$  que minimiza a expressão acima.

$$r(t) = \min_i \sum_j C(i, j)p(j|t).$$

# Função classe(t): rotulação pelas probabilidades das classes

Árvore probabilística:

$$\text{classe}(t) = (p(1|t), p(2|t), \dots, p(J|t))$$

Custo estimado em se atribuir a um objeto a classe  $i$  (estando no nó  $t$ ):

$$\sum_j C(i, j)p(j|t).$$

Custo estimado de erro no nó  $t$ :

$$r(t) = \sum_i p(i|t) \left( \sum_{i \neq j} C(i, j)p(j|t) \right) = \sum_{i,j} C(i, j)p(i|t)p(j|t).$$

# Função classe( $t$ ): Custo total de erro de classificação da árvore

Seja  $R(t)$  o custo total de erro no nó  $t$ , dado por

$$R(t) = r(t)p(t),$$

onde  $p(t)$  é a proporção de exemplos em  $E$  que incidem sobre o nó  $t$ .

# Função classe( $t$ ): Custo total de erro de classificação da árvore

Seja  $R(t)$  o custo total de erro no nó  $t$ , dado por

$$R(t) = r(t)p(t),$$

onde  $p(t)$  é a proporção de exemplos em  $E$  que incidem sobre o nó  $t$ .

O custo total de erros de classificação da árvore  $R(T)$  é dado por

$$R(T) = \sum_{t \in \bar{T}} R(t),$$

onde  $\bar{T}$  é o conjunto de folhas de  $T$ .

# Função $\text{score}(E,a)$ – Escolha do atributo ótimo

- Função baseada em uma medida de impureza
- Possíveis funções utilizadas como medida de impureza:
  - Entropia
  - Índice Gini

# Medida de impureza

**Definição 5.1** *Uma função de impureza é uma função  $\phi$  definida sobre o conjunto de todas as  $J$ -tuplas de números  $(p_1, p_2, \dots, p_J)$  satisfazendo  $p_j \geq 0$ ,  $j = 1, 2, \dots, J$   $\sum_j p_j = 1$  com as propriedades*

- $\phi$  atinge seu máximo somente no ponto  $(\frac{1}{J}, \frac{1}{J}, \dots, \frac{1}{J})$ ;
- $\phi$  atinge seu mínimo somente nos pontos  $(1, 0, \dots, 0), (0, 1, 0, \dots), \dots, (0, \dots, 0, 1)$ ;
- $\phi$  é uma função simétrica de  $p_1, p_2, \dots, p_J$ .

As restrições acima retratam as noções intuitivas de impureza: de fato, a impureza de um nó é:

- máxima quando todas as classes estão igualmente presentes no nó
- mínima quando o nó contém apenas uma classe.

# Medida de impureza

**Definição 5.2** *Dada uma função de impureza  $\phi$ , define-se a medida de impureza de um nó  $t$  como*

# Medida de impureza

**Definição 5.2** *Dada uma função de impureza  $\phi$ , define-se a medida de impureza de um nó  $t$  como*

$$\text{imp}(t) = \phi(p(1|t), p(2|t), \dots, p(J|t))$$

*onde  $p(j|t)$  é a probabilidade de um objeto incidente sobre o nó  $t$  pertencer à classe  $j$ .*

# Medida de impureza

Para todo nó  $t$ , suponha que haja uma divisão candidata  $s$  que divide  $t$  em  $t_v$  e  $t_f$ , de forma que uma proporção  $p_v$  dos exemplos de  $t$  vão para  $t_v$  e uma proporção  $p_f$  dos exemplos de  $t$  vão para  $t_f$ . O *critério de qualidade da divisão* é definido pelo decréscimo na impureza

$$\Delta imp(s, t) = imp(t) - p_v imp(t_v) - p_f imp(t_f) .$$

$$s^* =$$

# Medida de impureza

Para todo nó  $t$ , suponha que haja uma divisão candidata  $s$  que divide  $t$  em  $t_v$  e  $t_f$ , de forma que uma proporção  $p_v$  dos exemplos de  $t$  vão para  $t_v$  e uma proporção  $p_f$  dos exemplos de  $t$  vão para  $t_f$ . O *critério de qualidade da divisão* é definido pelo decréscimo na impureza

$$\Delta imp(s, t) = imp(t) - p_v imp(t_v) - p_f imp(t_f) .$$

$$s^* = \arg \max_s \Delta imp(s, t).$$

# Medida de impureza

$$\text{Impureza global da árvore } I(T) = \sum_{t \in \bar{T}} I(t) = \sum_{t \in \bar{T}} p(t) \text{imp}(t).$$

**Teorema 5.3** *A impureza global  $I(T)$  da árvore será mínima se, a cada nó não terminal  $t$ , tivermos escolhido a divisão  $s^* = \arg \max_s \Delta i(s, t)$ .*

**Dem.** Tome um nó qualquer  $t \in \bar{T}$  e, usando uma divisão  $s$ , particione o nó em  $t_v$  e  $t_f$ . A nova árvore  $T'$  terá impureza

$$I(T') = \sum_{\bar{T}-\{t\}} I(t) + I(t_v) + I(t_f).$$

O decréscimo global de impureza será

$$I(T) - I(T') = I(t) - I(t_v) - I(t_f).$$

Por depender somente do nó  $t$  e da divisão  $s$ , o decréscimo global de impureza pode ser escrito como

$$\begin{aligned} \Delta I(s, t) &= I(t) - I(t_v) - I(t_f) \\ &= p(t)imp(t) - p(t_v)imp(t_v) - p(t_f)imp(t_f) \end{aligned} \quad (3)$$

Definindo as proporções  $p_v$  e  $p_f$  dos exemplos de  $t$  que vão para  $t_v$  e  $t_f$ , respectivamente, como

$$p_v = p(t_v)/p(t) , \quad p_f = p(t_f)/p(t)$$

podemos reescrever a expressão 3 como

$$\begin{aligned} \Delta I(s, t) &= p(t)[imp(t) - p_v imp(t_v) - p_f imp(t_f)] \\ &= p(t)\Delta imp(s, t). \end{aligned}$$

Uma vez que  $\Delta I(s, t)$  difere de  $\Delta i(s, t)$  apenas pelo fator  $p(t)$ , a mesma divisão  $s^*$  maximiza as duas expressões. Portanto, o critério de seleção da melhor divisão de cada nó pode ser visto como uma seqüência de passos de minimização da impureza global da árvore.

# Entropia como medida de impureza

- Medida de incerteza:  $\mathcal{E}(p_1, p_2, \dots, p_J) = - \sum_{i=1}^J p_i \log p_i.$

Khinchin (1953) demonstrou que  $\mathcal{E}(p_1, p_2, \dots, p_J)$  possui uma série de propriedades que se pode esperar de uma medida de impureza.

Note que é imediato ver que  $\mathcal{E}$  é simétrica e que  $\mathcal{E}(p_1, p_2, \dots, p_J) = 0$  se, e somente se, algum dos números  $p_i$  for igual a 1 e todos os demais iguais a 0.

$$\mathcal{E}(p_1, p_2, \dots, p_J) \leq \mathcal{E}\left(\frac{1}{J}, \frac{1}{J}, \dots, \frac{1}{J}\right).$$

# Entropia como medida de impureza

- Medida de incerteza:  $\mathcal{E}(p_1, p_2, \dots, p_J) = - \sum_{i=1}^J p_i \log p_i.$

Khinchin (1953) demonstrou que  $\mathcal{E}(p_1, p_2, \dots, p_J)$  possui uma série de propriedades que se pode esperar de uma medida de impureza.

Note que é imediato ver que  $\mathcal{E}$  é simétrica e que  $\mathcal{E}(p_1, p_2, \dots, p_J) = 0$  se, e somente se, algum dos números  $p_i$  for igual a 1 e todos os demais iguais a 0.

$$\mathcal{E}(p_1, p_2, \dots, p_J) \leq \mathcal{E}\left(\frac{1}{J}, \frac{1}{J}, \dots, \frac{1}{J}\right).$$

$$\Delta imp(s, t) = imp(t) - p_v imp(t_v) - p_f imp(t_f)$$

$$s^* = \arg \max_s \Delta imp(s, t).$$

# Índice Gini como medida de impureza

O índice de diversidade Gini possui a forma

$$\mathcal{G}(p_1, p_2, \dots, p_J) = \sum_{i \neq j} p_i p_j$$

e também pode ser escrita como

$$\begin{aligned} \mathcal{G}(p_1, p_2, \dots, p_J) &= \sum_j p_j \sum_{i \neq j} p_i \\ &= \sum_j p_j (1 - p_j) \\ &= 1 - \sum_j p_j^2 \end{aligned}$$

- **Interpretação: erro de classificação:** Ao invés de definir  $classe(t)$  com o rótulo da classe majoritária, defina  $classe(t)$  como uma função que atribui, para um objeto selecionado aleatoriamente dentro do nó, a classe  $i$  com probabilidade  $p(i|t)$ . A probabilidade deste objeto ser da classe  $j$  é  $p(j|t)$ . Portanto, a probabilidade estimada de erro sob esta regra é o índice de Gini

$$\sum_{i \neq j} p_i p_j.$$

# Índice Gini para custos não unitários

$$C(i, i) = 0 ; \quad C(i, j) \geq 0, \quad i \neq j$$

$$\mathcal{G}(p(1|t), p(2|t), \dots, p(J|t)) = \sum_{i,j} C(i, j) p(i|t) p(j|t)$$

# Poda da árvore

- Árvore com muitos níveis: perda de acurácia
  - inserção de atributos com pouco poder preditivo ou altamente correlacionados com outros em níveis superiores (*overfitting*)
  - nós inferiores suscetíveis a ruídos
- Árvore com poucos níveis
  - uso de uma pequena parte da informação disponível (generaliza demais - *underfitting*)
- Problema: encontrar um tamanho ideal

# Poda da árvore

- Duas abordagens:
  - **pré-poda**: regras de parada durante a subdivisão dos nós
  - **pós-poda**: construção da árvore completa para posterior poda das sub-árvores consideradas não-confiáveis

# Pré-poda da árvore

- Exemplos de critérios de parada:
  1. Se, para toda divisão  $s$  do nó  $t$ ,  $\Delta I(s, t) < \beta$ , onde  $\beta > 0$  é um parâmetro definido pelo usuário.
  2. Se o número de exemplos que incidir sobre  $t$  for inferior a um parâmetro  $n$ .
  3. Se a proporção dos exemplos incidentes no nó em relação ao número total de exemplos em  $E$  for inferior a um parâmetro  $p$ .
  4. Se a estimativa de erro (ou o custo de erro)  $r(t)$  naquele nó for menor do que um parâmetro  $r$ .

Alguns dos algoritmos que realizam a pré-poda são o REAL (Laureto (1996)), o Cal5 (Müller & Wysotzki (1994)) , o C4 (Quinlan et al. (1986)) e o Assistant-86 (Cestnik et al. (1987)).

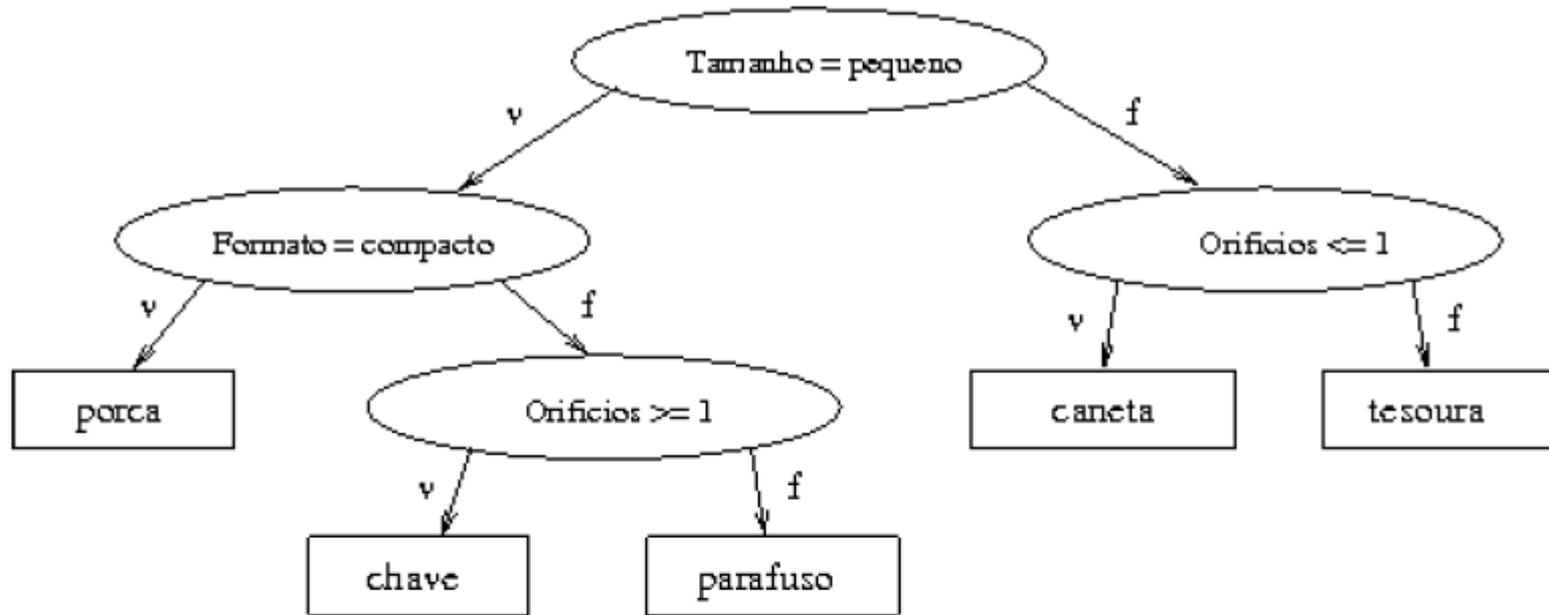
# Pós-poda da árvore

Dada uma árvore  $T$  e um nó interno  $t \in T$ , a *pós-poda* (ou simplesmente *poda*) do ramo  $T_t$  de  $T$  consiste em remover todos os descendentes próprios de  $t$ , declarando-o como nó terminal. Rotula-se  $t$  conforme a função  $classe(t)$  (definida anteriormente). A árvore podada desta forma será denotada por  $T - T_t$ .

Se  $T'$  é obtida de  $T$  através de podas sucessivas de ramos, então  $T'$  é denominada *sub-árvore podada* de  $T$  e é denotada por  $T' \preceq T$ .

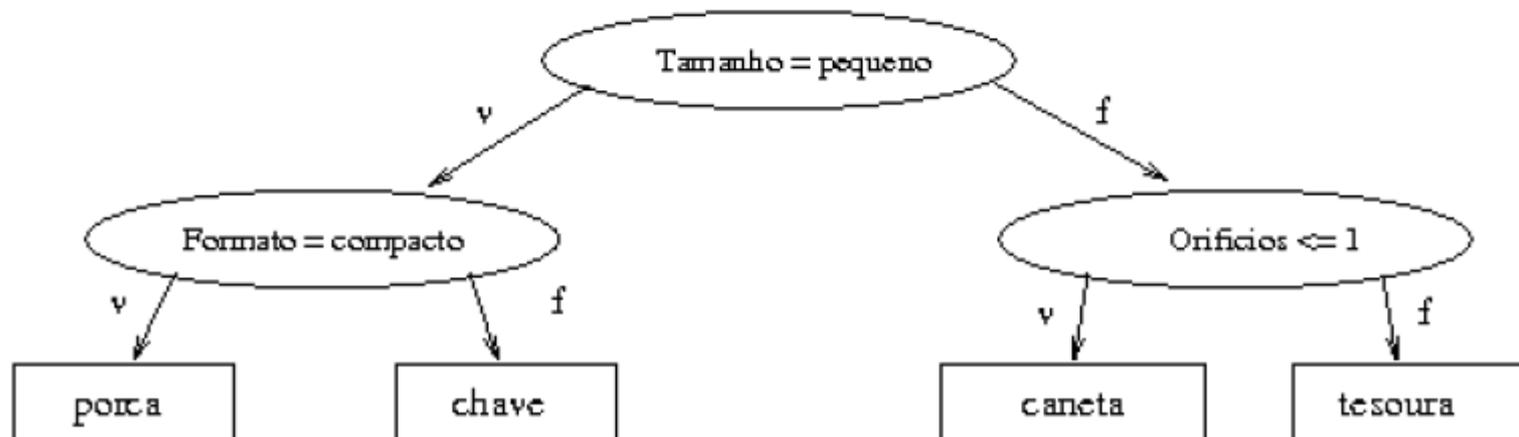
# Pós-poda da árvore - Exemplo

original



(a)

podada



# Poda por custo-complexidade

- Método mais conhecido, implementado no algoritmo CART (árvores binárias)
- Dois estágios:
  - Estágio 1: geração de uma sequência de árvores  $T_0, T_1, \dots, T_K$ , onde  $T_0$  é a árvore original,  $T_{i+1}$  é obtida a partir de  $T_i$  (substituindo uma ou mais subárvores por folhas), e  $T_K$  é a árvore contendo apenas uma folha (a raiz original)
  - Estágio 2: seleção da melhor árvore da sequência que minimiza o custo de erro de classificação

# Poda por custo-complexidade

- Método mais conhecido, implementado no algoritmo CART (árvores binárias)
- Dois estágios:
  - Estágio 1: geração de uma sequência de árvores  $T_0, T_1, \dots, T_K$ , onde  $T_0$  é a árvore original,  $T_{i+1}$  é obtida a partir de  $T_i$  (substituindo uma ou mais subárvores por folhas), e  $T_K$  é a árvore contendo apenas uma folha (a raiz original)
  - Estágio 2: seleção da melhor árvore da sequência que minimiza o custo de erro de classificação

# Poda por custo-complexidade - Estágio 1: obtenção das subárvores

- A árvore original ( $T_0$ ) não precisa ser construída exaustivamente
  - No lugar,  $T_{\max}$  (suficientemente grande), construída, por exemplo, usando um número mínimo de elementos por folha como critério de parada
- Para uma dada árvore  $T$ :
  - Complexidade de  $T$ : número de folhas ( $|F|$ )
  - $R(T)$  é o custo total de erro de classificação de  $T$
  - $\alpha$  é o parâmetro de complexidade
  - Custo-complexidade:  $R_\alpha(T) = R(T) + \alpha |F|$
- Variando  $\alpha$  em  $[0; +\infty)$  eu obtenho sub-árvores de  $T_{\max}$  que minimizam  $R_\alpha$  (chamadas  $T(\alpha)$ )
  - Para valores  $\alpha_1, \alpha_2, \dots, \alpha_k$ , a árvore  $T_k$  é a que minimiza  $R_\alpha$  para  $\alpha$  em  $[\alpha_k; \alpha_{k+1})$

# Poda por custo-complexidade - Estágio 1: obtenção das subárvores

- $T_0 = T_{\max}$
- $T_1 = T(0)$ , ou seja, uma subárvore de  $T_{\max}$  que minimize  $R(T)$  apenas ( $\alpha = 0$ )
  - No entanto  $R(T_{\max}) \leq R(T)$ , qualquer  $T \leq T_{\max}$

**Proposição 6.1** *Seja  $T$  uma árvore binária. Dada uma folha  $t \in T$ , para qualquer divisão de  $t$  em  $t_v$  e  $t_f$  tem-se*

$$R(t) \geq R(t_v) + R(t_f).$$

- Portanto,  $T_1$  deve ser a menor subárvore de  $T_{\max}$  tal que  $R(T_{\max}) = R(T)$

# Poda por custo-complexidade - Estágio 1: obtenção das subárvores

- Para encontrar  $T_1$  :

- $T = T_{\max}$

- repita:

- para  $t$  um nó interno de  $T$ , se  $R(t) = R(t_v) + R(t_f)$ ,

- $t_v$  e  $t_f$  filhos de  $t$ , pode o ramo  $Tt$  substituindo-o  
pelo nó  $t$

- até que não seja mais possível fazer podas

# Poda por custo-complexidade - Estágio 1: obtenção das subárvores

- Encontrando  $T_{k+1} = T(\alpha_{k+1})$  a partir de  $T_k = T(\alpha_k)$ , onde  $\alpha_{k+1} > \alpha_k$ :

Para todo nó interno  $t$  de  $T_k$  e seu respectivo ramo  $T_{k,t}$ , defina

$$R_\alpha(t) = R(t) + \alpha \quad (10)$$

$$R_\alpha(T_{k,t}) = R(T_{k,t}) + \alpha |\overline{T_{k,t}}|. \quad (11)$$

Note que  $R_\alpha(T_{k,t}) < R_{\alpha_k}(t)$  para todo nó interno de  $T_k$ , pois caso contrário  $T_k$  não seria minimal. Suponha agora que  $\alpha$  comece a subir de forma contínua. Enquanto  $R_\alpha(T_{k,t}) < R_\alpha(t)$  para cada  $t$ , teremos ainda  $T_k = T(\alpha)$ . No instante em que

$$R_\alpha(T_{k,t}) = R_\alpha(t) \quad (12)$$

para algum nó  $t$ , teremos então  $R_\alpha(T_k) = R_\alpha(T_k - T_{k,t})$  e  $T_k - T_{k,t}$  conterà menos folhas do que  $T_k$ ; logo,  $T_k \neq T(\alpha)$ . Nesse momento, o ramo  $T_{k,t}$  deverá ser podado.

# Poda por custo-complexidade - Estágio 1: obtenção das subárvores

As equações 10 e 11 fornecem o valor de  $\alpha$  para o qual a igualdade 12 ocorre:

$$\alpha = \frac{R(t) - R(T_{k,t})}{|\overline{T_{k,t}}| - 1}.$$

O ponto crítico  $\alpha_{k+1}$  será o menor valor de  $\alpha$  para o qual a igualdade 12 pode ocorrer. Ou seja,

$$\alpha_{k+1} = \min_{t \in \overline{T_k}^C} \frac{R(t) - R(T_{k,t})}{|\overline{T_{k,t}}| - 1}$$

onde  $\overline{T_k}^C$  é o conjunto de nós internos de  $T_k$ .

# Poda por custo-complexidade - Estágio 1: obtenção das subárvores

As equações 10 e 11 fornecem o valor de  $\alpha$  para o qual a igualdade 12 ocorre:

$$\alpha = \frac{R(t) - R(T_{k,t})}{|\overline{T_{k,t}}| - 1}.$$

O ponto crítico  $\alpha_{k+1}$  será o menor valor de  $\alpha$  para o qual a igualdade 12 pode ocorrer. Ou seja,

$$\alpha_{k+1} = \min_{t \in \overline{T_k}^C} \frac{R(t) - R(T_{k,t})}{|\overline{T_{k,t}}| - 1}$$

onde  $\overline{T_k}^C$  é o conjunto de nós internos de  $T_k$ .

Para obter a árvore  $T_{k+1}$ , execute o seguinte procedimento: seja  $t$  um nó interno de  $T_k$ ; se  $R_{\alpha_{k+1}}(t) = R_{\alpha_{k+1}}(T_{k,t})$ , pode o ramo  $T_{k,t}$ , substituindo-o pelo nó  $t$ . Repita este procedimento até que não seja mais possível nenhuma poda. A árvore resultante será  $T_{k+1}$ .

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

- Seleção da subárvore com menor custo de erro de classificação
- Problema?

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

- Seleção da subárvore com menor custo de erro de classificação
- Problema?

$T_{max}$  é a de menor custo para a amostra de treinamento utilizada

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

- Seleção da subárvore com menor custo de erro de classificação
- Problema?  
Tmax é a de menor custo para a amostra de treinamento utilizada
- Solução:

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

- Seleção da subárvore com menor custo de erro de classificação
- Problema?  
Tmax é a de menor custo para a amostra de treinamento utilizada
- Solução:
  - utilizar outra amostra  $E_A$  (de teste) para estimar adequadamente esse custo (com exemplos não utilizados na construção da árvore)

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

Seja  $T_k$  uma sub-árvore da seqüência. Submeta a árvore  $T_k$  ao conjunto de testes  $E_A$ , ou seja, utilize  $T_k$  para classificar cada uma das instâncias de  $E_A$ .

Denote por  $N^A$  a cardinalidade do conjunto  $E_A$ . Sejam  $N_j^A$  o número de instâncias da classe  $j$  em  $E_A$  e  $N_{ij}^A$  o número de instâncias de classe  $j$  em  $E_A$  que tenham sido classificados por  $T_k$  como classe  $i$ . Então, a probabilidade estimada de um objeto de classe  $j$  ser classificado por  $T_k$  como classe  $i$  será

$$Q(i|j) = N_{ij}^A / N_j^A.$$

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

Denote por  $R(j)$  o custo esperado no erro de classificação dos objetos de classe  $j$ .  $R(j)$  será dado por

$$R(j) = \sum_{i=1}^J C(i, j)Q(i|j)$$

onde  $C(i, j)$  é o custo de erro.

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

Denote por  $R(j)$  o custo esperado no erro de classificação dos objetos de classe  $j$ .  $R(j)$  será dado por

$$R(j) = \sum_{i=1}^J C(i, j)Q(i|j)$$

onde  $C(i, j)$  é o custo de erro.

Finalmente, seja  $\pi(j)$  a probabilidade *a priori* de um objeto qualquer de  $E_A$  ser de classe  $j$ . A estimativa do custo da árvore  $T_k$  é dada por,

$$R^C(T_k) = \sum_{j=1}^J R(j)\pi(j).$$

# Poda por custo-complexidade - Estágio 2: Escolha da melhor subárvore

Denote por  $R(j)$  o custo esperado no erro de classificação dos objetos de classe  $j$ .  $R(j)$  será dado por

$$R(j) = \sum_{i=1}^J C(i, j)Q(i|j)$$

onde  $C(i, j)$  é o custo de erro.

Finalmente, seja  $\pi(j)$  a probabilidade *a priori* de um objeto qualquer de  $E_A$  ser de classe  $j$ . A estimativa do custo da árvore  $T_k$  é dada por,

$$R^C(T_k) = \sum_{j=1}^J R(j)\pi(j).$$

Após calculada a estimativa de custo  $R^C(T_k)$  para cada sub-árvore  $T_k$  da seqüência, pode-se simplesmente escolher a sub-árvore

$$T_k^* = \arg \min_{1 \leq k \leq K} R^C(T_k).$$

# Missing values

- Alternativas:
  - “missing” como um valor possível do atributo
  - Treinamento: ir até onde der na árvore. Se chegar em uma folha, ótimo. Se parar em um nó interno  $t$ , dividir os exemplos pelos ramos
  - Classificação: ir até onde der na árvore. Se chegar em uma folha, ótimo. Se parar em um nó interno  $t$ , usar a distribuição deste nó para classificar a instância

# Seleção de Características

- A seleção de características está embutida no aprendizado da árvore

# Características de Árvores de Decisão

- Método supervisionado ou não-supervisionado?

- 

-

# Características de Árvores de Decisão

- Método supervisionado ou não-supervisionado?

**Supervisionado!** Tanto a amostra de treinamento (para construção da árvore) quanto a de teste (para poda da árvore) devem ser rotuladas.

- 

-

# Características de Árvores de Decisão

- Método supervisionado ou não-supervisionado?

**Supervisionado!** Tanto a amostra de treinamento (para construção da árvore) quanto a de teste (para poda da árvore) devem ser rotuladas.

- Método paramétrico ou não-paramétrico?

-

# Características de Árvores de Decisão

- Método supervisionado ou não-supervisionado?

**Supervisionado!** Tanto a amostra de treinamento (para construção da árvore) quanto a de teste (para poda da árvore) devem ser rotuladas.

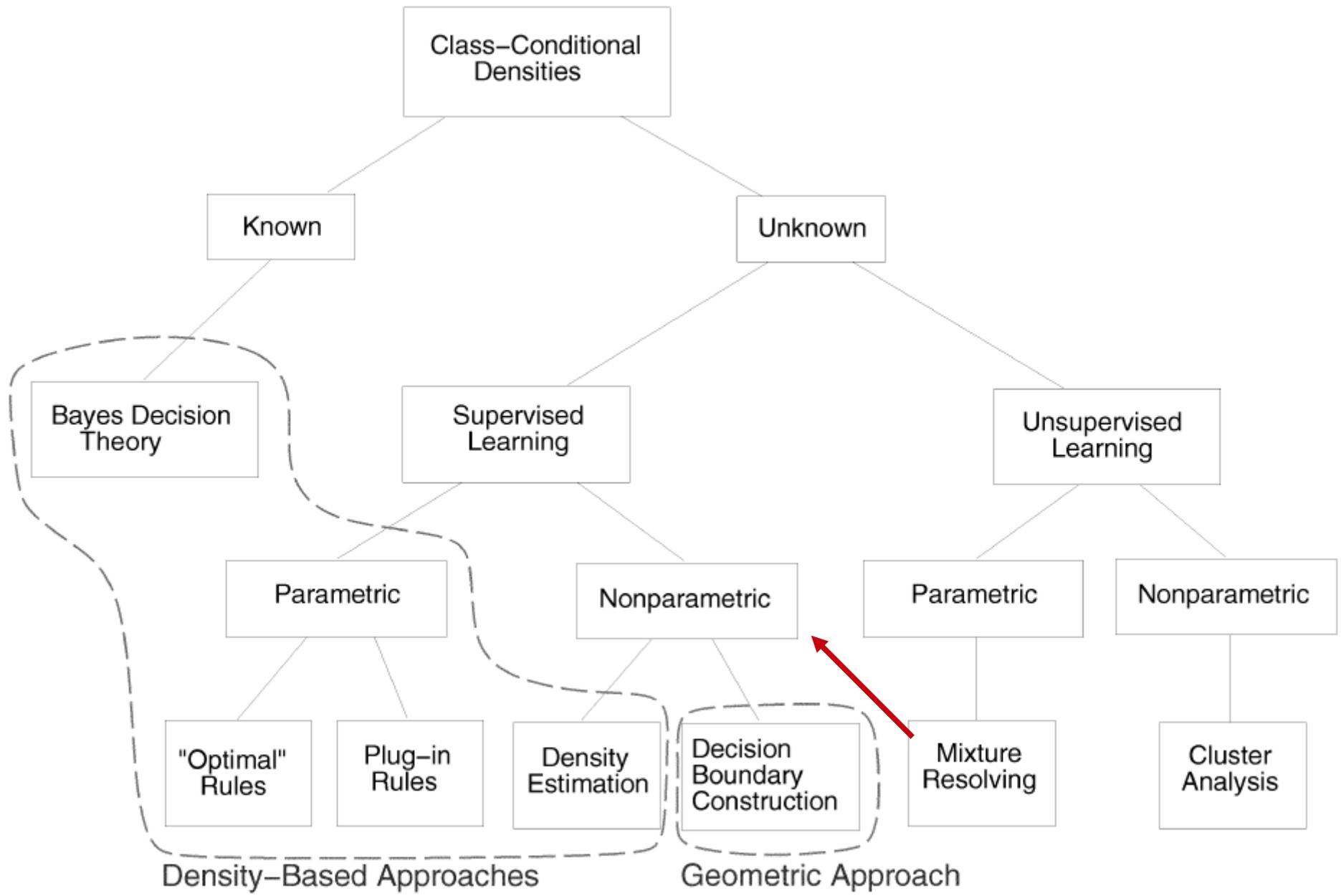
- Método paramétrico ou não-paramétrico?

**Não-paramétrico!** Não é assumida nenhuma família de distribuições para descrever o espaço nem a “cara” do classificador



# Características de Árvores de Decisão

- Método supervisionado ou não-supervisionado?  
**Supervisionado!** Tanto a amostra de treinamento (para construção da árvore) quanto a de teste (para poda da árvore) devem ser rotuladas.
- Método paramétrico ou não-paramétrico?  
**Não-paramétrico!** Não é assumida nenhuma família de distribuições para descrever o espaço nem a “cara” do classificador
- Também considerado **não-métrico**, por poder trabalhar com atributos nominais (categóricos)



# “Implementações” de árvores de decisão

- Variação das funções vistas hoje levam a diferentes algoritmos de árvores de decisão
- Exemplos:
  - ID3: bem simples - não lida com atributos numéricos nem missing values, não faz poda
  - C4.5: evolução do ID3
  - CART (Classification And Regression Trees): árvores binárias
  - ...

# Algumas das principais vantagens

- Interpretabilidade  
Pode ser convertida em um conjunto de regras
- Lida com atributos numéricos e categóricos
- Já efetua automaticamente seleção de características (selecionador **embutido**)
- Por ser não-paramétrica, não assume nenhuma família de distribuição
- Conseguem lidar com *missing values*

# Algumas das principais desvantagens

- Alguns algoritmos só trabalham com atributos discretos (ID3, C4.5)
  - REAL trabalha com números reais
- Funciona bem quando há poucos atributos altamente relevantes, mas pode ter mais dificuldades se houver uma grande complexidade de interações
- **Instável** - tende a ser sensível à amostra de treinamento devido à sua natureza gulosa (ruído, atributos irrelevantes)

# Alternativas para diminuir a instabilidade

- **Bagging**: várias árvores de decisão, cada uma treinada por uma amostra *bootstrap* de mesmo tamanho da amostra original (sorteio com reposição da amostra original)
  - classificação: classe mais “votada” pelas árvores
  - demora mais para treinar e classificar, mas o processo é facilmente paralelizável

# Alternativas para diminuir a instabilidade

- *Random Forests*: variação de *Bagging* (n tree árvores) na qual, no momento em que a árvore é construída, apenas um subconjunto aleatório (de  $m$  try elementos) dos atributos participa da subdivisão de um nó
  - Mais estável que Bagging

# Trabalho

- Semelhante aos anteriores, usando Random Forests
- R: Pacote randomForest
- Parâmetros a serem utilizados:
  - mtry (o parâmetro mais sensível): raiz (quadrada) de  $n$ ,  $n$  sendo o número de características (testar outros se puder)
  - ntree = 500 (quanto maior melhor, depende do tempo disponível)

# Referências

- Notas do Prof. Marcelo Lauretto (edisciplinas)
- **CLARK, A.; FOX, C.; LAPPIN, S. The Handbook of Computational Linguistics and Natural Language Processing. Ed. Wiley-Blackwell, 2010. Cap 7**