Árvore de Decisão (Continuação)

Prof. Clodoaldo A. M. Lima & Sarajane M. Peres

Como Incorporar atributos com Valores Continuos?

- A definição inicial do ID3 é restrita a atributos que assumem um conjunto discreto de valores inclusive o atributo objetivo cujo valor é predito pela árvore de decisão

- Altura, peso, temperatura são atributos contínuos. Eles têm muitos ou infinitos valores possíveis.

- **Para tratar este problema é preciso discretizar o atributo. Por exemplo, o atributo *Preço para o domínio do restaurante* foi discretizado nos valores $, $$, $$$. O atributo *Temperatura* foi discretizado nos valores *frio, médio, quente.***

- **Solução: algoritmos de discretização de variáveis contínuas**

Como Tratar Atributos Multivalorados

- Quando um atributo tem um grande número de valores possíveis, a medida de ganho dará uma indicação não apropriada da utilidade do atributo.

- Considere o caso extremo onde para todo os exemplos o valor do atributo seja diferente. Por exemplo, no domínio do restaurante se tivéssemos o atributo *NomeDoRestaurante.*

- Neste caso, cada exemplo é um caso único e portanto tem uma só classificação.

- Portanto a medida do ganho assume o valor máximo para este atributo. Entretanto, o atributo pode ser irrelevante ou inútil. Uma solução seria usar a *proporção de ganho.*
- **Ver exercício 18.12 de Russel e Norvig.**

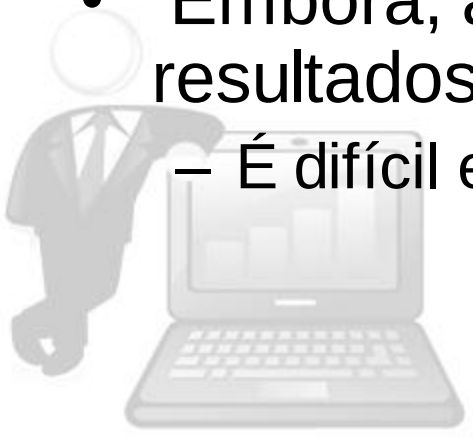
Exemplos com Mesma Descrição mas Diferentes Classificações

- Se existe dois ou mais exemplos com a mesma descrição (em termos dos atributos) mas diferentes classificações, então o algoritmo de aprendizagem deve falhar em encontrar uma árvore consistente com todos os exemplos de treino.

- **Solução:**
- **Fazer com que cada nó folha registre ou a classificação da maioria para seu conjunto de exemplos ou registre as estimativas de probabilidade de cada classificação usando as freqüências relativas.**

Como evitar overfitting?

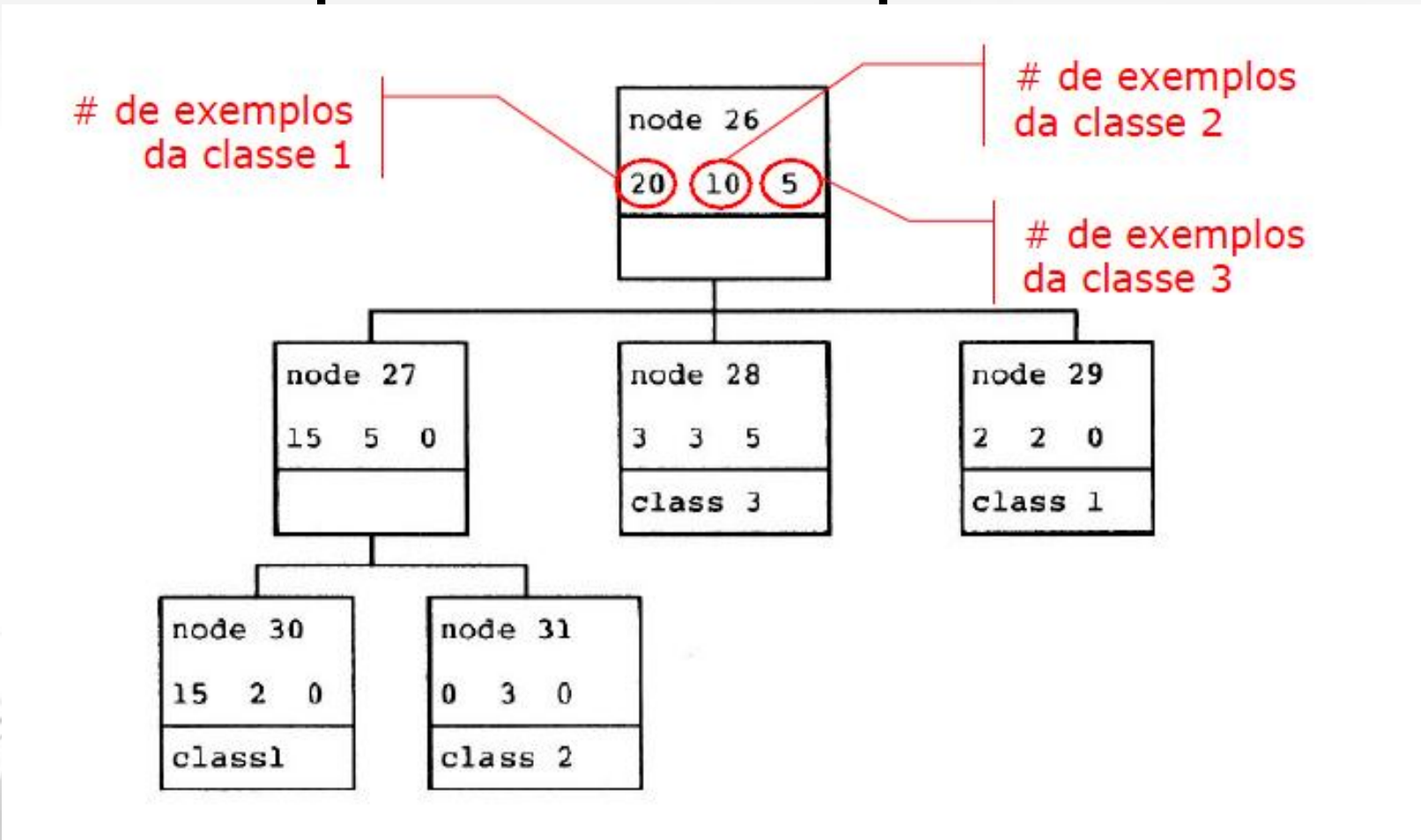
- Alternativas
  - Parar o crescimento antes que a árvore classifique os dados de treinamento perfeitamente
  - Permitir o completo crescimento da árvore e podá-la
-
- A primeira alternativa parece ser mais direta
- Embora, a segunda tem encontrado melhores resultados na prática
  - É difícil estimar precisamente o momento de parar



Abordagens para podar árvores

- Quatros estratégias
  - Error-Complexity Pruning
  - Critical Value Pruning
  - Minimum-Error Pruning
  - Reduced-Error Pruning

Exemplo de uma árvore parcialmente podada



Error-Complexity Pruning

- Breiman et al(1984) desenvolveu um método baseado em dois estágios.

- No primeiro estágio, uma seqüência de árvores T_0, T_1, ..., T_k é construída sobre o conjunto de treinamento onde T_0 é a árvore original antes da poda e T_k é um nó raiz.

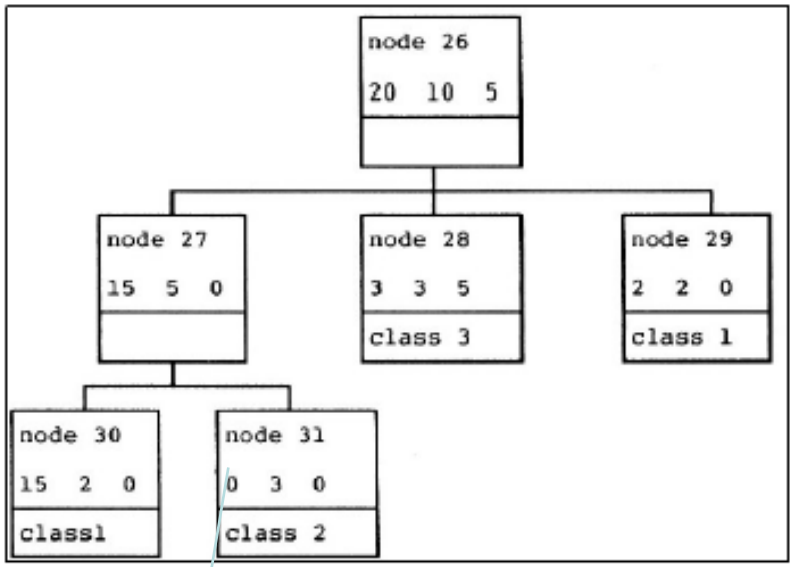
- No segundo estágio, uma destas arvores é escolhida, baseado na estimação do erro de generalização.

- Este método leva em conta ambos o número de erros e a complexidade (tamanho) da árvore.

Error-Complexity Pruning

- O método funciona da seguinte forma:
  - Cada nó é um ponto de partida para uma sub-árvore
  - Antes da poda, as folhas contém exemplos que pertencem a apenas uma classe
  - Após a poda, a folha conterá exemplos de diversas classes
  - Assim, a classe dessa folha é dada pela classe com maior freqüência dentre os exemplos
  - Isso gera erro
  - Dividindo esse erro pelo número de folhas obtém-se uma medida de redução do erro por folha
    - Essa é a medida error-complexity

Error-Complexity Pruning



Lima,

Número de exemplos da classe 1

- Assuma 200 instâncias no conjunto de treinamento
- Seja t um nó e Tt uma sub-arvore com raiz em t
- Observando o nó 26
  - Possui 4 folhas, N_T =4
  - Caso esse nó seja podado, ele será da classe 1
  - Assim, 15 dos 35 exemplos serão incorretamente classificados.
  - A sub-árvore com raiz no nó 26,tem 4 folhas, N_T = 4.
  - A taxa de erro é r(t) = 15/35
  - A proporção de dados em t é p(t) = 35/200

Error-Complexity Pruning

- O custo do nó t é:

$$R(t) = r(t)p(t) = \frac{15}{35} \times \frac{35}{200} = \frac{15}{200}$$

- Caso a árvore não fosse podada, o custo da sub-árvore seria:

$$R(T_t) = \sum R(i), \qquad para\ i = folhas\ da\ sub-árvore$$

$$R(T_t) = \frac{2}{17} \times \frac{17}{200} + 0 + \frac{6}{11} \times \frac{11}{200} + \frac{2}{4} \times \frac{4}{200} = \frac{10}{200}$$

- O algoritmo calcula α para cada sub-árvore e escolhe a que contém o menor valor para podar
- Assim, o custo total da sub-árvore é

$$R(T_t) + \alpha N_t \qquad R(t) + \alpha \quad (quando\ a\ sub-árvore\ for\ podada)$$

- Igualando as equações

$$\alpha = \frac{R(t) - R(T_t)}{N_T - 1} = \frac{15/200 - 10/200}{4 - 1} = \frac{5}{600}$$

Error-Complexity Pruning

- Este algoritmo calcula α para cada sub-árvore (exceto a primeira) e seleciona a sub-árvore com menor valor de α para a poda
- Repete-se este procedimento até que nenhum sub-árvore produza uma série de árvores a serem podadas.
- **O próximo passo é selecionar uma destas como a árvore final.**
- O critério de seleção da árvore final é o menor erro de classificação incorreta, entretanto, este critério não pode ser baseado no **conjunto de treinamento**, uma vez que sempre daria a árvore não podada como melhor. Ao invés disso, cada um das arvores podadas é usada para classificar um conjunto de dados de **teste independente**.

Error-Complexity Pruning

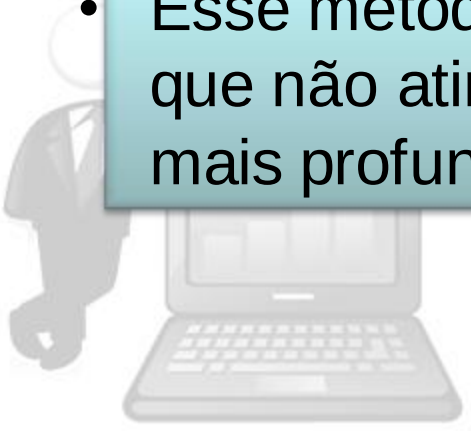
- O método de Breiman seleciona a menor arvore com uma taxa de erro de classificação dentro de um erro padrão mínimo.
- O desvio padrão da taxa de classificação incorreta é

$$SE = \sqrt{\frac{R*(100-R)}{N}}$$

  - Onde R = taxa de classificação incorreta e N numero de exemplo no dados de teste

Critical Value Pruning

- Esse método observa os valores que medem a importância de cada nó
- Essa medida é calculada na criação da árvore
- Esse valor mostra quão bem o atributo divide os dados
- Esse método especifica um valor crítico e poda os nós que não atingem o referido valor, a menos que um nó mais profundo não o atinja também



Critical Value Pruning

- Quanto maior o valor crítico, maior o grau de poda da árvore e menor a árvore resultante

- Na prática, uma série de árvores são geradas aumentando o valor crítico

Minimum-Error Pruning

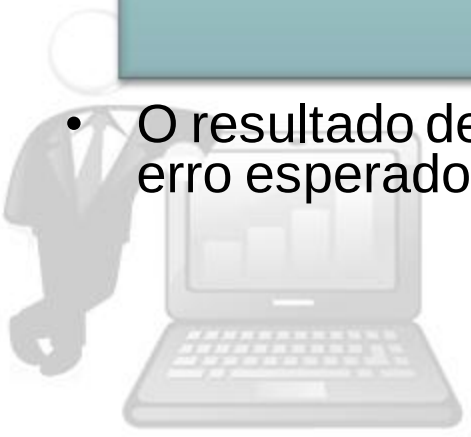
- Niblet & Bratko (1986) desenvolveram um método para encontrar um única arvore que deveria, teoricamente, produzir o menor erro ao classificar um conjunto de dados independente.
- Assuma um conjunto de dados com k classes, assuma também que temos observado n exemplos dos quais o maior número, nc, estão na classe c.
- Se nós predizermos que todos os exemplos estarão na classe c, a taxa de erro esperado, E_k, é dada por

$$E_k = \frac{(n - nc + k - 1)}{n + k}$$

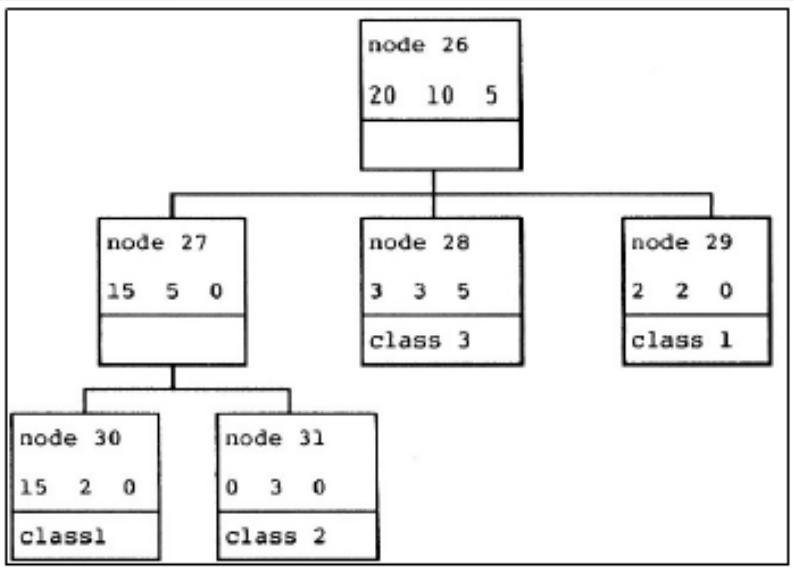
- Observe que este assume que todas as classes são igualmente prováveis.

Minimum-Error Pruning

- O método funciona da seguinte forma
  - Para cada nó na árvore, calcule a taxa de erro esperada se a sub-árvore é podada, de forma que o nó torna-se uma folha.
  - Então, calcule a taxa de erro esperada se a sub-árvore não é podada usando a taxa de erro de cada ramo, combinada com os pesos segundo a proporção de exemplos ao longo de cada ramo.
  - O procedimento é recursivo uma vez que a taxa de erro para um ramo não pode ser calculado até que você saiba se o ramo será podado
  - Se a poda da sub-árvore conduzir para uma taxa de erro esperado maior, então mantenha a sub-árvore
- O resultado deverá ser uma árvore podada que minimiza a taxa de erro esperado em dados de classificação independente.



Minimum-Error Pruning



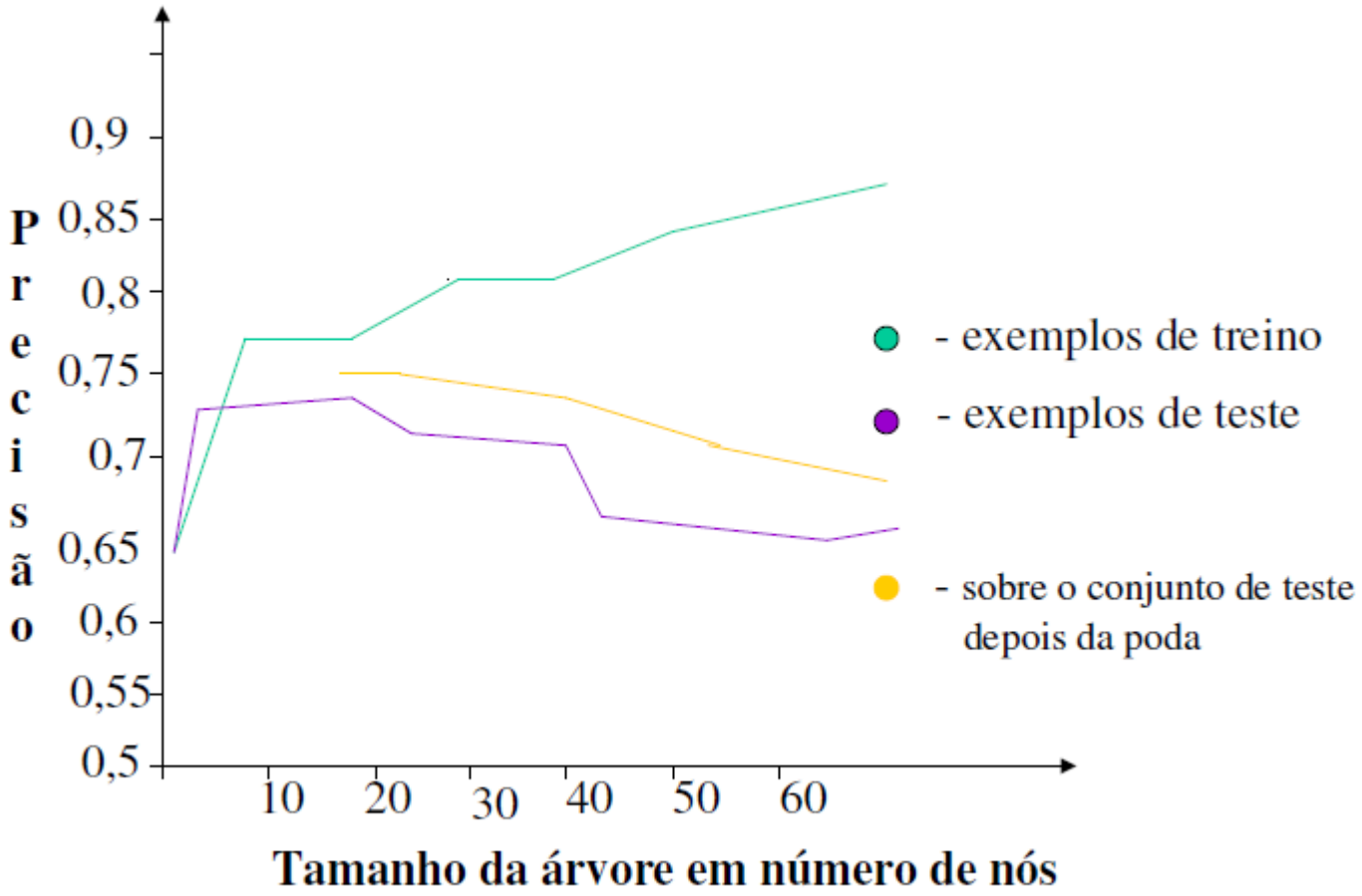
- Considere o nó 27
- Podando a sub-árvore
  - Ek = (20 – 15 +3 -1)/(20+3) = 0.304
- Não podando a sub-árvore
  - Ek = (17/20)* (17 – 15 + 3-1)/(17+3)
  - + (3/20)* (3-3+3-1)/(3+3) = 0.220

- Uma vez que o erro esperado para sub-árvore podada é maior, não realizado a poda
- Teoricamente, este é a solução ideal, uma vez que minimiza o erro esperado total e não requer um conjunto de teste separado.
- Entretanto, a suposição de classes igualmente prováveis é raramente satisfeita

Reduced-Error Pruning

- Proposto por Quinlan (1987)
- O método funciona da seguinte forma
  - Comece com uma arvore completa e aplique os dados de teste.
  - Para cada nó não folha, conte o número de erros se a sub-árvore é mantida e o numero de erros se o nó torna-se uma folha através da poda
  - A diferença entre o número de erros (isto é, número de erros se a sub-árvore é mantida menos o número de erro se a o nó torna-se folha), se positivo, é uma medida do ganho da poda da árvore
  - De todos os nós, escolha aquele com maior diferença como a sub-árvore a ser podada
  - Continue este processo, incluindo aqueles nós onde a redução é zero, até que a poda aumente a taxa de classificação incorreta
- Este deverá produzir a menor versão da arvore mais precisa com relação para o conjunto de dados de teste

Reduced-Error Pruning



Rule Post-Pruning

- Uma variante deste método é usado no C4.5
- Indicada quando os dados são limitados
- Este método envolve os seguintes passos:
  - Inferir a árvore de decisão como o conjunto de treinamento, fazendo crescer a árvore até que os dados de treino sejam ajustados tão bem quanto possível permitindo que o overfitting ocorra.
  - Converter a árvore aprendida em um conjunto equivalente de regras criando uma regra para cada caminho partindo da raíz para um nó folha.
  - Podar (generalizar) cada regra removendo qualquer pré-condição que resulte em aumento da precisão estimada
  - Ordenar as regras podadas conforme sua precisão, e considerá-las nesta seqüência ao classificar instâncias subseqüêntes.

Exemplo

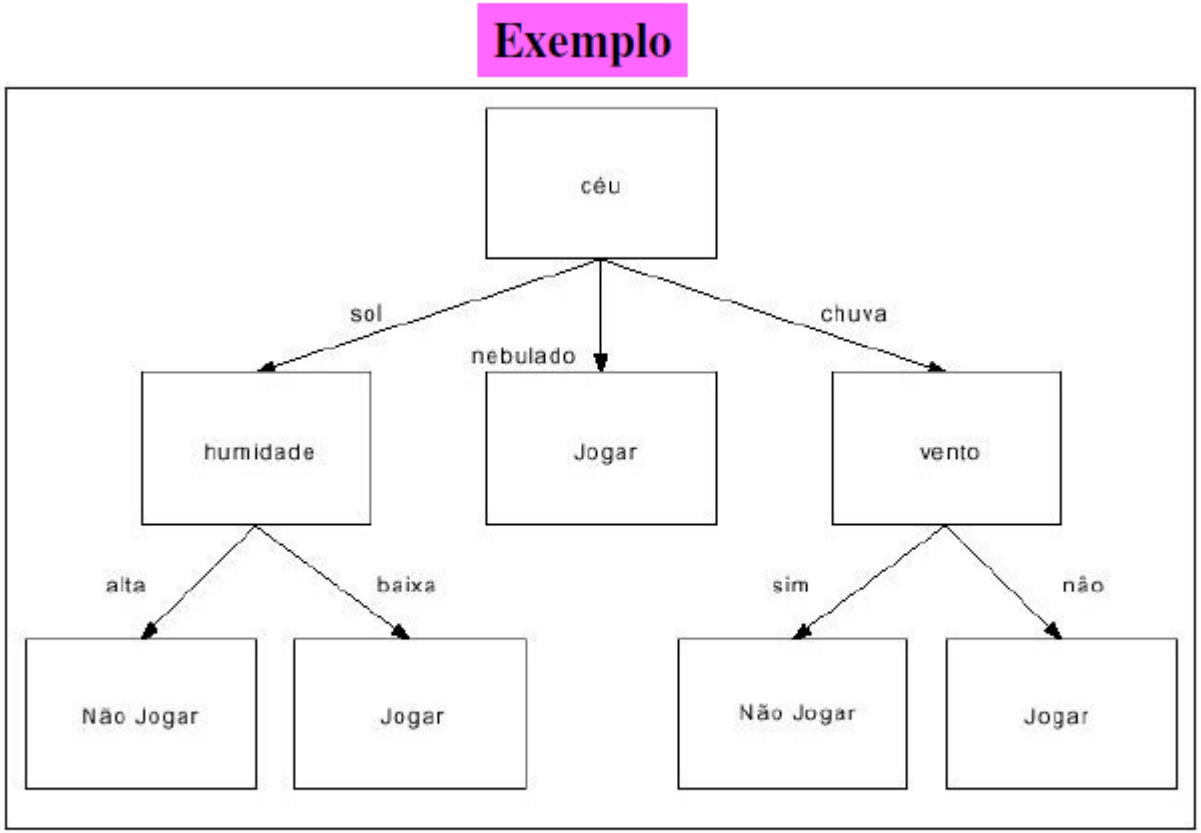


Figure 1: Árvore de decisão do exemplo - Jogar Golf -.

Rule Post-Pruning

- O ramo mais a esquerda da árvore
- SE Céu = sol E Humidade = Alta ENTÃO JogasGolf = Não

- Possíveis podas (generalização):
- SE Céu = Sol ENTÃO JogasGolf = Não
- SE Humidade =Alta ENTÃO JogarGolf = Não

- Intuitivamente, a poda gera definições mais fracas e portanto permite um conjunto maior de exemplos positivos

- Nenhuma poda deve ser feita se isto implicar em redução da precisão estimada da regra.

Precisão de uma regra

- Considere a representação da regra na forma:
- Corpo $\rightarrow$ Cabeça
- Abreviatura: B $\rightarrow$ H
- A precisão de uma regra R é uma medida do quanto uma regra é especifica para o problema

Definição de Precisão (Accuracy)

- È a probabilidade condicional de H ser verdade dado que B é verdade

$$Acc(R) = P(H|B) = \frac{P(HB)}{P(B)}$$

- A precisão de uma regra é uma medida do quanto uma regra é especifica para o problema (Lavrac et al, 1999)

- Mede a fração de exemplos predito positivos que

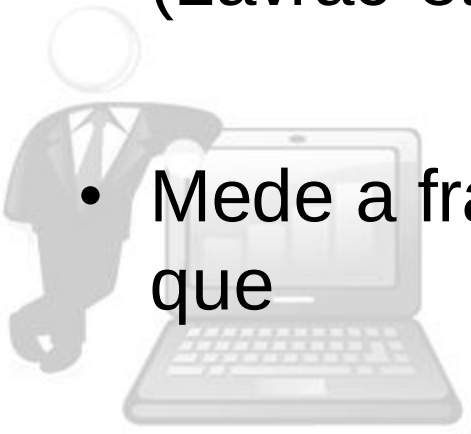


Tabela de Contingência para uma Regra

$\mathbf{B \rightarrow H}$

	H	$\bar{H}$	
B	hb	$\bar{h}b$	b
$\bar{B}$	$h\bar{b}$	$\bar{h}\bar{b}$	$\bar{b}$
	h	$\bar{h}$	n

- hb – número de exemplos para os quais H é verdade e B é verdade
- $\bar{h}b$ - número de exemplos para os quais H é falso e B é verdade
- b - numero de exemplos para os quais B é verdade
- n – numero total de exemplo de treino
- Precisão

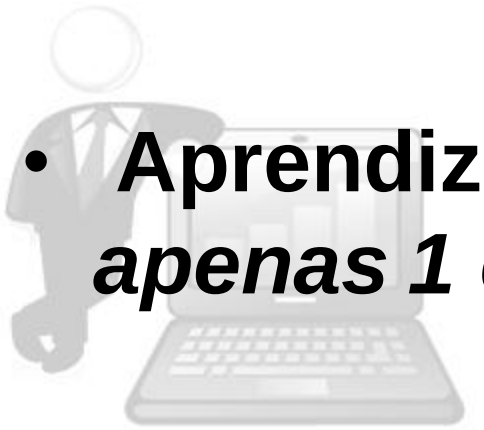
$$Acc(R) = \frac{hb}{b}$$

Algoritmos *IDT: Melhorias*

- **Atributos com valores desconhecidos: como calcular a entropia e ganho?**

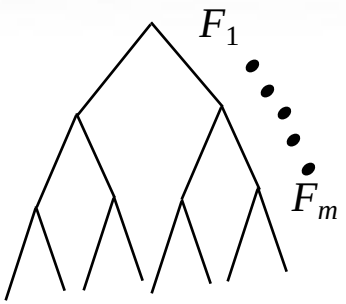
- **Atributos com importância diferente: chuva em pique-nique (só ocorre 1 vez no conj.)**

- **Aprendizado incremental: como aprender *apenas 1 exemplo a mais? (ITI)***



Complexidade Computacional

- No pior caso cria uma árvore completa onde todos os caminhos testam todos os atributos. Suponha *n* exemplos e *m* atributos.



Máximo de *n* exemplos espalhados por todos os nós em cada um dos *m* níveis

- Em cada nível, *i*, na árvore, temos que examinar os $m - i$ atributos restantes para cada instância do nível para calcular ganhos de informação.

$$\sum_{i=1}^{m} i \cdot n = O(nm^2)$$

- Porém, a árvore aprendida raramente é completa (número de folhas é $\leq n$). Na prática, a complexidade é linear em relação ao número de atributos (*m*) e o número de exemplos de treinamento (*n*).