Capítulo

5

Indução de Regras e Árvores de Decisão

Maria Carolina Monard
José Augusto Baranauskas

"Cavalheiros, o que pode ser mais fácil que colocar esse ovo em pé, algo que vocês disseram ser impossível? É a coisa mais simples do mundo. Qualquer um poderia fazer — depois de saber como."

– *Colombo*

Um ponto interessante sobre os seres humanos está relacionado à sua habilidade de fazer generalizações precisas a partir de fatos. O ser humano é capaz de encontrar estruturas ou padrões apenas observando um processo (aparentemente caótico) do mundo real. Em ciência de computação, essa habilidade pode ser obtida a partir de um conjunto de exemplos, fornecidos pelo usuário ou por um processo do mundo real, através da inferência indutiva, a qual mesmo sendo o recurso mais utilizado pelo cérebro na produção de conhecimento novo, deve ser utilizada cuidadosamente. Neste capítulo são descritas técnicas de Aprendizado de Máquina simbólico, enfatizando indutores cujas linguagens de representação de hipóteses consistem em árvores de decisão, regras ordenadas e regras não ordenadas.

5.1 Introdução

Como visto no capítulo anterior, sistemas de aprendizado podem ser classificados em *caixa-preta* e *orientado a conhecimento*. Nos sistemas orientados a conhecimento, o interesse principal consiste em obter descrições simbólicas que sejam de fácil compreensão e utilização por meio de modelos mentais. Segundo o *postulado da compreensibilidade* de Michalski (1983b):

> "Os resultados da indução por computador devem ser descrições simbólicas das entidades fornecidas, sendo semântica e estruturalmente similares àquelas que um especialista humano poderia produzir observando as mesmas entidades. Os componentes dessas descrições devem ser compreensíveis como simples 'pedaços' de informação, diretamente interpretáveis em linguagem natural, bem como reportar conceitos quantitativos e qualitativos de maneira integrada."

Como regra prática, Michalski assume que os componentes de descrição, tais como regras ou nós em uma árvore de decisão, devem ser expressões contendo menos de cinco condições em uma conjunção; poucas condições em uma disjunção; no máximo um nível de parênteses; no máximo

uma implicação; não mais que dois quantificadores e nenhuma recursão. Embora esses valores possam ser flexíveis, descrições geradas por indução dentro dos limites propostos são similares à representação do conhecimento humano e, portanto, fáceis de serem compreendidas. Embora tais medidas sejam simples de serem avaliadas, é importante salientar que elas são meramente sintáticas e que, muitas vezes, também devem ser consideradas medidas semânticas (Pazzani 2000a).

Existem muitos algoritmos de aprendizado que induzem classificadores. Neste capítulo são descritos indutores que contribuem para a compreensão dos dados em contraste com indutores que visam apenas uma grande precisão. Por exemplo, a indução de regras ou árvores de decisão pode auxiliar médicos a compreender melhor os dados, enquanto uma Rede Neural convencional, mesmo com precisão similar, pode ser muito difícil de ser compreendida por seres humanos[1]. Por exemplo, no desenvolvimento de sistemas especialistas é importante que especialistas humanos possam, fácil e confiavelmente, verificar o conhecimento extraído e relacioná-lo ao seu próprio domínio de conhecimento. Além disso, algoritmos de aprendizado que induzem estruturas compreensíveis, contribuindo para a compreensão do domínio considerado, podem produzir conhecimento novo (Dietterich 1986).

Em resumo, este capítulo concentra-se em aprendizado simbólico supervisionado para resolver problemas de classificação. O termo *simbólico* indica que os classificadores devem ser legíveis e interpretáveis por humanos.

5.2 Linguagens de Representação

A seguir são descritas algumas linguagens de representação freqüentemente utilizadas em AM simbólico em ordem crescente de complexidade e força expressiva. São fornecidas explicações intuitivas sobre essas linguagens evitando complexidade de notação. Uma vez que uma linguagem de representação pode descrever exemplos, hipóteses e conhecimento do domínio, por generalidade estes termos são referenciados como *itens* nas próximas seções.

5.2.1 Lógica de Ordem Zero ou Proposicional

Na lógica de ordem zero ou cálculo proposicional, o item a ser representado é descrito por conjunções, disjunções e negações de constantes booleanas que representam atributos individuais. Por exemplo:

$$\mathsf{fêmea} \wedge \mathsf{adulta} \rightarrow \mathsf{pode\_ter\_filhos}$$

Esta linguagem tem um baixo poder descritivo, não sendo capaz de descrever objetos sobre os quais relações são observadas.

5.2.2 Lógica de Atributos

De forma a representar itens, vários indutores proposicionais utilizam uma linguagem *baseada em atributos*. Formalmente, a lógica de atributos é equivalente ao cálculo proposicional, mas emprega uma notação mais poderosa e flexível. A melhoria é devido ao fato que os atributos são tratados como variáveis que podem assumir diversos valores. Por exemplo:

$$\mathsf{sexo{=}feminino} \wedge \mathsf{idade{=}adulta} \rightarrow \mathsf{classe{=}pode\_ter\_filhos}$$

ou equivalentemente,

[1] Existem, entretanto, várias pesquisas relacionadas com explicação de Redes Neurais.

$$\mathsf{sexo(feminino)} \wedge \mathsf{idade(adulta)} \rightarrow \mathsf{classe(pode\_ter\_filhos)}$$

Embora a maioria dos indutores utilize a lógica de atributos para descrever exemplos e hipóteses, sua baixa capacidade de expressão impede a representação de objetos estruturados, assim como as relações entre objetos ou entre seus componentes. Assim, aspectos relevantes dos exemplos que, de alguma forma poderiam caracterizar o conceito sendo aprendido, podem não ser representados.

5.2.3 Lógica de Primeira Ordem

De forma a superar as limitações de representação impostas por uma linguagem de atributos, o aprendizado utilizando representações que possuem maior poder, tais como algumas variações da lógica de primeira ordem, tem recebido maior atenção. A lógica de primeira ordem permite descrever e raciocinar sobre *objetos* e *predicados* que especificam *propriedades* de objetos ou *relacionamentos* entre objetos do domínio $\mathcal{D}$.

Um subconjunto importante da lógica de primeira ordem é composto pelas *cláusulas de Horn*. Uma cláusula de Horn consiste em uma regra cuja cabeça contém um único predicado e um corpo com zero, um ou mais predicados. O seguinte exemplo, na sintaxe proposta por Kowalsky (1979) para a linguagem de programação lógica PROLOG, descreve que uma pessoa X é irmão da pessoa Y se X é homem e ambos X e Y possuem o mesmo pai Z, onde X, Y, e Z são *variáveis* que representam objetos.

$$\mathsf{irmão}(X,Y) \ \texttt{:-}\ \mathsf{homem}(X),\ \mathsf{pai}(Z,X),\ \mathsf{pai}(Z,Y).$$

A parte à esquerda do símbolo **:-** é a cabeça e a parte à direita do símbolo é o corpo (ou cauda) da cláusula. O símbolo **:-** é equivalente à implicação lógica $\leftarrow$ e é denominado *neck* [2]. As vírgulas separando cada predicado significam conjunções lógicas. Além disso, todas as variáveis estão sempre universalmente quantificadas, ou seja, no exemplo acima, a cláusula é verdadeira para todo $X, Y, Z \in \mathcal{D}$. As variáveis entre parênteses são chamadas de *argumentos*.

Nota-se que se todos os predicados não possuem argumentos, a linguagem se reduz à lógica de ordem zero e se todos os predicados possuem um único argumento constante (sem variáveis envolvidas), a linguagem se reduz à lógica de atributos.

5.2.4 Lógica de Segunda Ordem

A lógica de segunda ordem é uma extensão da lógica de primeira ordem, permitindo que os predicados possam ser considerados como variáveis. Por exemplo, suponha o *esquema*:

$$P_1(X,Y) \ \texttt{:-}\ P_2(X),\ P_3(X,Z),\ P_4(Y,Z).$$

onde P_1, P_2, P_3, P_4 são variáveis que representam predicados e X, Y, Z são variáveis que representam objetos. Uma possível instanciação poderia ser

$$\mathsf{irmão}(X,Y) \ \texttt{:-}\ \mathsf{homem}(X),\ \mathsf{pai}(Z,X),\ \mathsf{pai}(Z,Y).$$

Com a instanciação, o esquema permanece intacto e apenas os nomes dos predicados podem variar. É conveniente salientar que esta linguagem de representação é tão rica e flexível que seu uso é, em muitos casos, computacionalmente inviável. Algumas vezes, na prática, se introduz restrições, tais como limitar o número de predicados na cláusula, excluir definições recursivas ou mesmo limitar o número de argumentos do predicado (Morik, Wrobel, Jörg-Uwe, & Emde 1993).

[2] q **:-** $p \equiv q \leftarrow p \equiv p \rightarrow q$

5.3 Indução de Árvores de Decisão

Algoritmos que induzem árvores de decisão pertencem à família de algoritmos *Top Down Induction of Decision Trees* — TDIDT. Uma árvore de decisão (ou AD) é uma estrutura de dados definida recursivamente como:

- um *nó folha* que corresponde a uma classe ou
- um *nó de decisão* que contém um teste sobre algum atributo. Para cada resultado do teste existe uma aresta para uma subárvore. Cada subárvore tem a mesma estrutura que a árvore.

Na Figura 5.1 é mostrado um exemplo ilustrativo de uma árvore de decisão para o diagnóstico de um paciente. Na figura, cada elipse é um teste em um atributo para um dado conjunto de dados de pacientes. Cada retângulo representa uma classe, ou seja, o diagnóstico. Para diagnosticar (classificar) um paciente, basta começar pela raíz, seguindo cada teste até que uma folha seja alcançada.

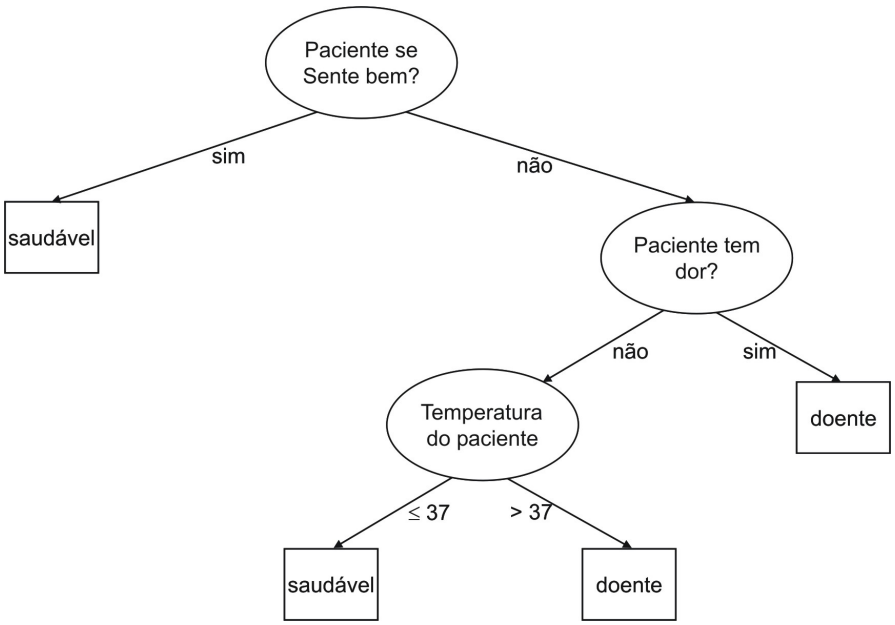

Figura 5.1: Uma árvore de decisão simples para o diagnóstico de um paciente

É fácil perceber que a árvore pode ser representada como um conjunto de regras. Cada regra tem seu início na raíz da árvore e caminha até uma de suas folhas. Por exemplo, a árvore mostrada na Figura 5.1 pode ser lida como:

if paciente se sente bem = sim **then**
  classe = saudável
else
  if paciente tem dor = não **then**
    if temperatura do paciente $\leq$ 37 **then**
      classe = saudável
    else {temperatura do paciente $>$ 37}
      classe = doente
    end if
  else {paciente tem dor = sim}

classe = doente
end if
end if

Como as regras que representam uma árvore de decisão são disjuntas, isto é, apenas uma única regra dispara quando um novo exemplo é classificado, uma forma alternativa de representar tais regras consiste em escrever uma regra separadamente para cada nó folha, iniciando pela raíz; conseqüentemente, nenhum **else** é realmente necessário:

if paciente se sente bem = sim **then**
classe = saudável
end if
if paciente se sente bem = não **and** paciente tem dor = não
and temperatura do paciente ≤ 37 **then**
classe = saudável
end if
if paciente se sente bem = não **and** paciente tem dor = não
and temperatura do paciente > 37 **then**
classe = doente
end if
if paciente se sente bem = não **and** paciente tem dor = sim **then**
classe = doente
end if

5.3.1 Construindo uma Árvore de Decisão

O método para a construção de uma árvore de decisão a partir de um conjunto de treinamento T é surpreendemente simples. Assumindo que as classes sejam denotadas por $\{C_1, C_2, \ldots, C_k\}$, os seguintes passos devem ser seguidos:

1. T contém um ou mais exemplos, todos pertencentes à mesma classe C_j. Nesse caso, a árvore de decisão para T é um nó folha identificando a classe C_j;

2. T não contém exemplos. Novamente, nessa situação, a árvore é uma folha mas a classe associada à folha deve ser determinada a partir de informação além de T. Por exemplo, a classe mais freqüente para o nó pai desse nó pode ser utilizada;

3. T contém exemplos que pertencem a várias classes. Nesse caso á idéia é refinar T em subconjuntos de exemplos que são (ou aparentam ser) conjuntos de exemplos pertencentes a uma única classe.

   Normalmente, um teste é escolhido, baseado em um único atributo que possui resultados mutuamente exclusivos (na realidade, cada indutor tem sua própria forma de escolher o atributo que será utilizado no teste). Sejam os possíveis resultados do teste denotados por $\{O_1, O_2, ..., O_r\}$. T é então particionado em subconjuntos T_1, T_2, ..., T_r, nos quais cada T_i contém todos os exemplos em T que possuem como resultado daquele teste o valor O_i. A AD para T consiste em um nó interno identificado pelo teste escolhido e uma aresta para cada um dos resultados possíveis;

4. Os passos 1, 2 e 3 são aplicados recursivamente para cada subconjunto de exemplos de treinamento de forma que, em cada nó, as arestas levam para as subárvores construídas a partir do subconjunto de exemplos T_i;

5. Após a construção da AD, a poda pode ser realizada para melhorar a capacidade de generalização da AD (vide Seção 5.3.3).

5.3.2 Escolha do Melhor Atributo para Particionar

A chave para o sucesso de um algoritmo de aprendizado por AD depende do critério utilizado para escolher o atributo que particiona o conjunto de exemplos em cada iteração. Algumas possibilidades para escolher esse atributo são:

- aleatória: seleciona qualquer atributo aleatoriamente;
- menos valores: seleciona o atributo com a menor quantidade de valores possíveis;
- mais valores: seleciona o atributo com a maior quantidade de valores possíveis;
- ganho máximo: seleciona o atributo que possui o maior ganho de informação esperado, isto é, seleciona o atributo que resultará no menor tamanho esperado das subárvores, assumindo que a raíz é o nó atual;
- índice Gini (Breiman, Friedman, Olshen, & Stone 1984);
- razão de ganho (Quinlan 1988).

5.3.3 Poda

Após a construção da árvore de decisão, é possível que o classificador induzido seja muito específico para o conjunto de treinamento. Nesse caso, diz-se que o classificador super-ajustou os dados de treinamento, ou seja, ocorreu um *overfitting* (Seção 4.4). Como os exemplos de treinamento são apenas uma amostra de todos os exemplos possíveis (Seção 4.5), é possível adicionar arestas na árvore que melhoram seu desempenho nos dados de treinamento mas que piora seu desempenho em um conjunto de teste.

Na Figura 5.2 é ilustrado o impacto do super-ajuste no aprendizado por árvores de decisão (Mitchell 1998b). O eixo horizontal dessa figura indica o número total de nós na AD à medida que a árvore é construída. O eixo vertical indica o erro nas previsões feitas pela árvore. Na linha pontinhada é mostrada a taxa de erro da AD calculada sobre o conjunto de treinamento enquanto que na linha sólida é mostrada a taxa de erro calculada sobre um conjunto de teste. Como esperado, o erro sobre o conjunto de treinamento (erro aparente) decresce monotonicamente à medida que a árvore é construída. Entretanto, o erro medido sobre o conjunto de teste (erro verdadeiro) primeiramente decresce até $N2$ nós na árvore e então aumenta.

Para tentar solucionar o problema de super-ajuste dos dados, alguns indutores *podam* a AD depois de induzi-la. Esse processo, mostrado na Figura 5.3, reduz o número de nós (testes) internos, reduzindo a complexidade da árvore enquanto produz um desempenho melhor que a árvore original. Em geral, os indutores de AD separam por si próprios o conjunto de exemplos em um conjunto de exemplos de treinamento (que é utilizado para construir a AD) e um conjunto de exemplos de poda, o qual é efetivamente utilizado para realizar o processo de poda.

Esse tipo de poda é chamado de *pós-poda*, como descrito na Seção 4.4, uma vez que ele ocorre após a indução da AD. Existem vários métodos de pós-poda, incluindo complexidade do erro (Breiman, Friedman, Olshen, & Stone 1984) e erro pessimista (Quinlan 1988).

É também possível utilizar *pré-poda* na AD. Esse processo é efetuado enquanto a AD é induzida. Entretanto, a pré-poda sofre de um efeito colateral: conjunções de teste podem ser a melhor forma de particionar os exemplos, mas seus atributos individuais podem não distinguir muito bem os exemplos. Assim, a pré-poda pode evitar que determinados tipos de conjunções apareçam na árvore.

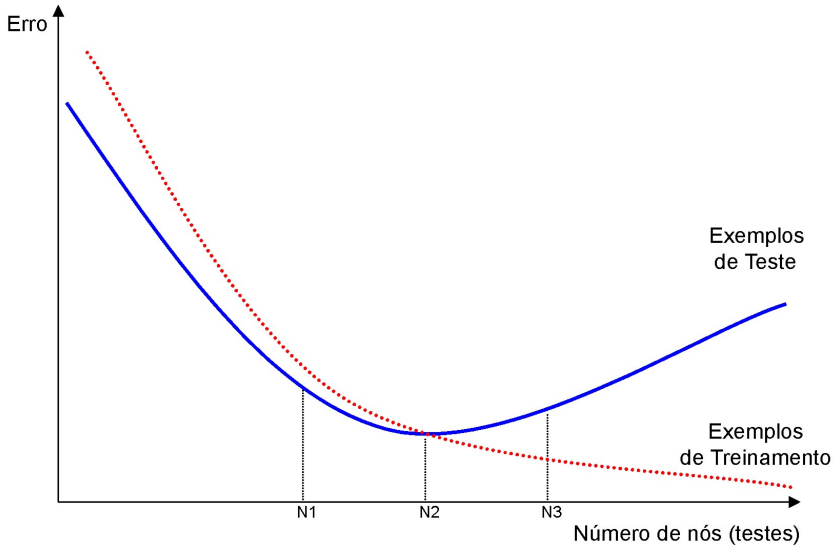

Figura 5.2: Relacionamento entre tamanho da árvore de decisão e a taxa de erro

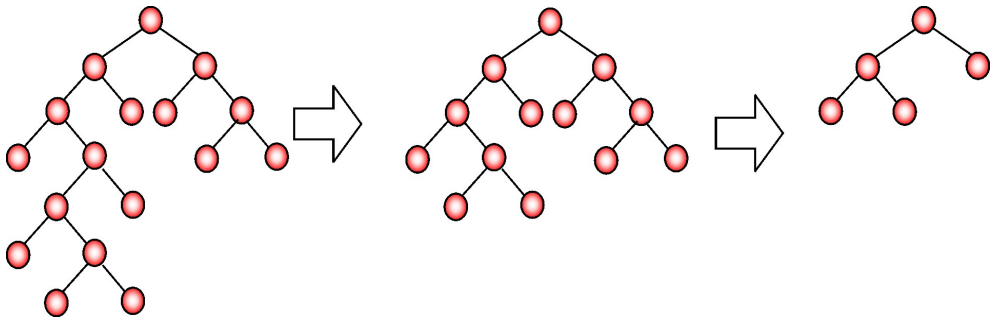

Figura 5.3: Uma árvore grande é induzida de forma a super-ajustar os exemplos e então ela é podada até obter uma árvore menor (mais simples)

5.3.4 Classificando Novos Exemplos

A AD, após construída, pode ser utilizada para classificar novos exemplos iniciando-se pela raíz da árvore e caminhando através de cada nó de decisão até que uma folha seja encontrada. Quando uma folha é encontrada, a classe do novo exemplo é dada pela classe daquela folha.

5.3.5 Interpretação Geométrica

Considerando os exemplos como um vetor de m atributos, tal vetor corresponde a um ponto no espaço m-dimensional dos atributos. Sob esse ponto de vista, a AD corresponde a uma divisão (para cada teste) deste espaço em regiões, sendo cada região rotulada com uma classe.

Atributo-Valor

A fim de ilustrar essa divisão do espaço, considere apenas dois atributos reais e duas classes (o e +). Na Figura 5.4 é mostrado um exemplo para testes do tipo:

$$X_i \ op \ Valor$$

onde $X_i \in \{X_1, X_2\}$, op é um operador no conjunto $\{\leq, >\}$ e $Valor$ é um valor válido para os atributos X_1 ou X_2.

Para esse tipo de teste, o espaço de descrição é particionado em regiões retangulares, ou seja, no caso geral, em hiperplanos que são ortogonais aos eixos do atributo testado e paralelo a todos os demais eixos. Essa é uma observação importante uma vez que regiões produzidas por árvores de decisão que utilizam tais testes são todos hiperretângulos. À medida que a árvore é construída, mais e mais regiões são adicionadas ao espaço (linhas sólidas).

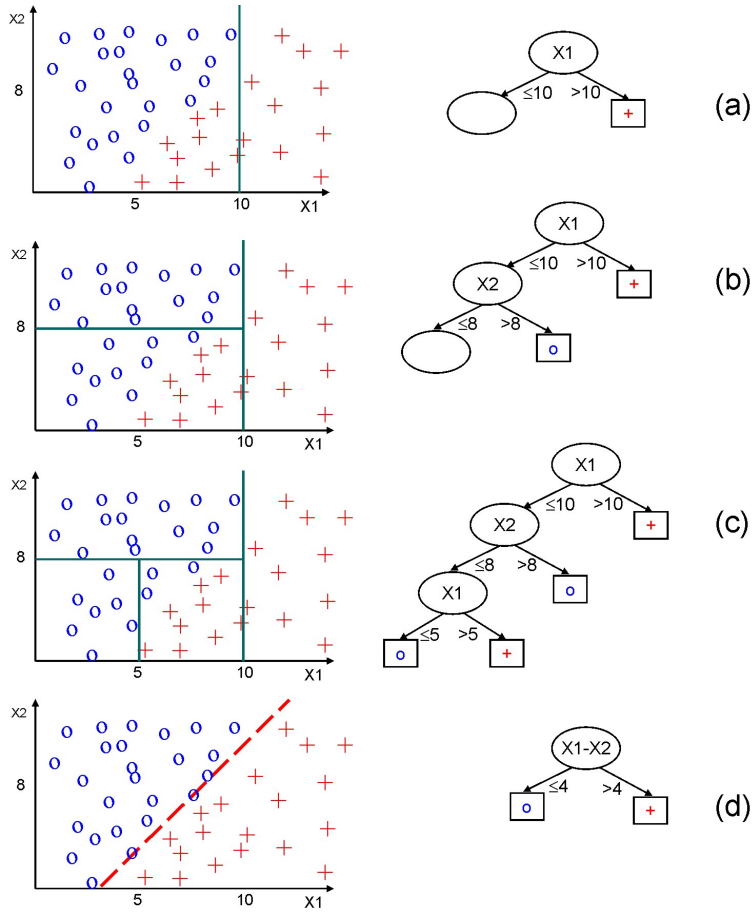

Figura 5.4: Regiões que não se sobrepõem são formadas por uma árvore de decisão no espaço de descrição

Combinação Linear de Atributos

Ainda considerando a Figura 5.4, deve ser observado que em (d), uma hipótese mais simples (linha tracejada) pode classificar melhor os exemplos. Isso motivou a construção de árvores de decisão *oblíquas* que produzem hiperplanos não ortogonais (Breiman, Friedman, Olshen, & Stone 1984, Capítulo 5), (Murthy, Kasif, & Salzberg 1994). Nesse caso, os testes assumem a seguinte forma:

$$c_1 \times X_1 + c_2 \times X_2 + \ldots + c_m \times X_m \ \ op \ \ Valor$$

onde c_i é uma constante, X_i é um atributo contínuo (inteiro ou real), *op* é um operador no conjunto $\{<, \leq, >, \geq\}$ e *Valor* é uma constante. Para esse tipo de teste, o espaço de descrição é dividido em regiões não retangulares, ou seja, hiperplanos que não são necessariamente ortogonais aos eixos.

Resumindo, a indução de árvores de decisão é um dos métodos de aprendizado mais utilizados na prática. É um método rápido para aprendizado de conceitos, simples de implementar,

Exemplo No.	Aparência	Temperatura	Umidade	Ventando	Viajar?
T_1	sol	25	72	sim	vá
T_2	sol	28	91	sim	não_vá
T_3	sol	22	70	não	vá
T_4	sol	23	95	não	não_vá
T_5	sol	30	85	não	não_vá
T_6	nublado	23	90	sim	vá
T_7	nublado	29	78	não	vá
T_8	nublado	19	65	sim	não_vá
T_9	nublado	26	75	não	vá
T_{10}	nublado	20	87	sim	vá
T_{11}	chuva	22	95	não	vá
T_{12}	chuva	19	70	sim	não_vá
T_{13}	chuva	23	80	sim	não_vá
T_{14}	chuva	25	81	não	vá
T_{15}	chuva	21	80	não	vá

Tabela 5.1: Conjunto de exemplos viagem

permite transformar seus resultados em forma de regras interpretáveis, pode tratar exemplos com ruído e é uma tecnologia madura utilizada em vários produtos comerciais.

Entretanto, árvores muito grandes são geralmente difícies de serem lidas. Além disso, árvores univariadas nas quais apenas um atributo é utilizado em cada nó interno de teste são limitadas a partições paralelas aos eixos no espaço de descrição, limitando o conceito que pode ser aprendido. Por outro lado, árvores multivariadas (oblíquas) podem utilizar mais de um atributo em cada nó interno, mas requerem maiores recursos computacionais para serem induzidas.

5.3.6 Um Exemplo

Nesta seção é dado um exemplo da construção de um AD, adaptado de (Quinlan 1988). Suponha um conjunto de exemplo consistindo de medidas diária sobre as condições do tempo, na qual cada exemplo é composto pelos seguinte atributos:

- aparência: assume os valores discretos “sol”, “nublado” ou “chuva”;
- temperatura: um valor numérico indicando a temperatura em graus Celsius;
- umidade: também um valor numérico indicando a portentagem de humidade;
- ventando: assume valores discretos “sim” ou “não” indicando se é um dia com vento.

Além disso, para cada dia (exemplo), alguém rotulou cada medida diária como “vá” se o tempo estava bom o suficiente para uma viagem ao campo ou “não_vá” caso contrário. Os exemplos seriam similares aos mostrados na Tabela 5.1. Embora esse exemplo possua apenas duas classe, é importante lembrar que uma árvore de decisão pode trabalhar com qualquer número k de classes $\{C_1, C_2, \ldots, C_k\}$.

A indução da AD inicia considerando o fato que o conjunto de treinamento T contém exemplos pertencentes a mais de uma classe. Assim, é necessário escolher um teste baseado em um único atributo. Como descrito na Seção 5.3.2, a escolha do atributo para particionar os exemplos depende da implementação de cada indutor. Para este exemplo, vamos escolher *aparência* como teste, tendo três possíveis resultados $\{O_1, O_2, O_3\} = \{\text{sol, nublado, chuva}\}$.

Teste	Exemplo	Aparência	Temperatura	Umidade	Ventando	Viajar?
if aparência = sol	T_1	sol	25	72	sim	vá
	T_2	sol	28	91	sim	não_vá
	T_3	sol	22	70	não	vá
	T_4	sol	23	95	não	não_vá
	T_5	sol	30	85	não	não_vá
if aparência = nublado	T_6	nublado	23	90	sim	vá
	T_7	nublado	29	78	não	vá
	T_8	nublado	19	65	sim	não_vá
	T_9	nublado	26	75	não	vá
	T_{10}	nublado	20	87	sim	vá
if aparência = chuva	T_{11}	chuva	22	95	não	vá
	T_{12}	chuva	19	70	sim	não_vá
	T_{13}	chuva	23	80	sim	não_vá
	T_{14}	chuva	25	81	não	vá
	T_{15}	chuva	21	80	não	vá

Tabela 5.2: Construindo uma AD a partir dos exemplos de viagem (passo 1)

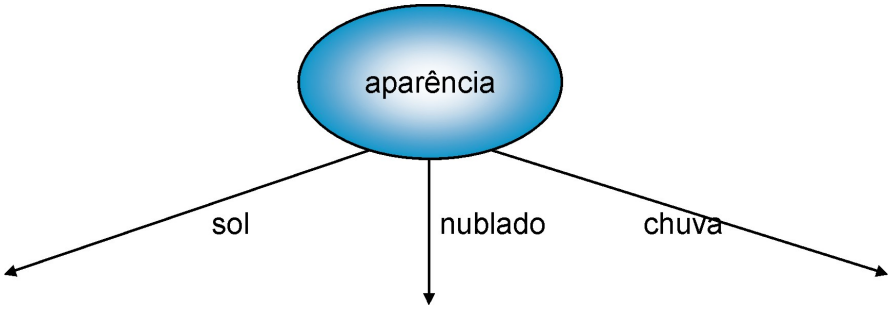

Figura 5.5: Construindo uma AD a partir dos exemplos de viagem (passo 1)

Então T é particionado em 3 subconjuntos como é mostrado na Tabela 5.2 e na Figura 5.5 correspondente.

Como pode ser notado, cada subconjunto ainda contém exemplos pertencentes a várias classes, portanto é necessário escolher um outro teste baseado em um único atributo. Assuma que foi selecionado o atributo *umidade* para as subárvores "sol" e "nublado" e *ventando* para a subárvore "chuva". Cada subconjunto é agora particionado como é mostrado na Tabela 5.3 e a Figura 5.6 correspondente.

Após a construção da AD completa, considere a seguinte subárvore:

if aparência = nublado **then**
  if umidade > 70 **then**
    classe = vá {Exemplos cobertos: T_6, T_7, T_9, T_{10}}
  else {umidade ≤ 70}
    classe = não_vá {Exemplo coberto: T_8}
  end if
end if

É possível notar que apenas um exemplo (T_8) satisfaz o teste "umidade ≤ 70"; todos os outros exemplos para a subárvore nublado pertencem à "classe = vá". Isso pode indicar um *overfitting* do dados e o indutor pode podar essa subárvore, como pode ser visto na Tabela 5.4 e a Figura 5.7 correspondente.

A poda da AD pode, em geral, melhorar o desempenho para exemplos não vistos. Isso pode parecer contra-intuitivo, uma vez a poda descarta alguma informação (o exemplo T_8 nesse caso).

Teste	Exemplo	Aparência	Temperatura	Umidade	Ventando	Viajar?
if aparência = sol	T_1	sol	25	72	sim	vá
e umidade $\leq$ 78	T_3	sol	22	70	não	vá
if aparência = sol	T_2	sol	28	91	sim	não_vá
e umidade > 78	T_4	sol	23	95	não	não_vá
	T_5	sol	30	85	não	não_vá
if aparência = nublado	T_6	nublado	23	90	sim	vá
umidade > 70	T_7	nublado	29	78	não	vá
	T_9	nublado	26	75	não	vá
	T_{10}	nublado	20	87	sim	vá
if aparência = nublado						
e umidade $\leq$ 70	T_8	nublado	19	65	sim	não_vá
if aparência = chuva	T_{11}	chuva	22	95	não	vá
e ventando = não	T_{14}	chuva	25	81	não	vá
	T_{15}	chuva	21	80	não	vá
if aparência = chuva	T_{12}	chuva	19	70	sim	não_vá
e ventando = sim	T_{13}	chuva	23	80	sim	não_vá

Tabela 5.3: Construindo uma AD a partir dos exemplos de viagem (passo 2)

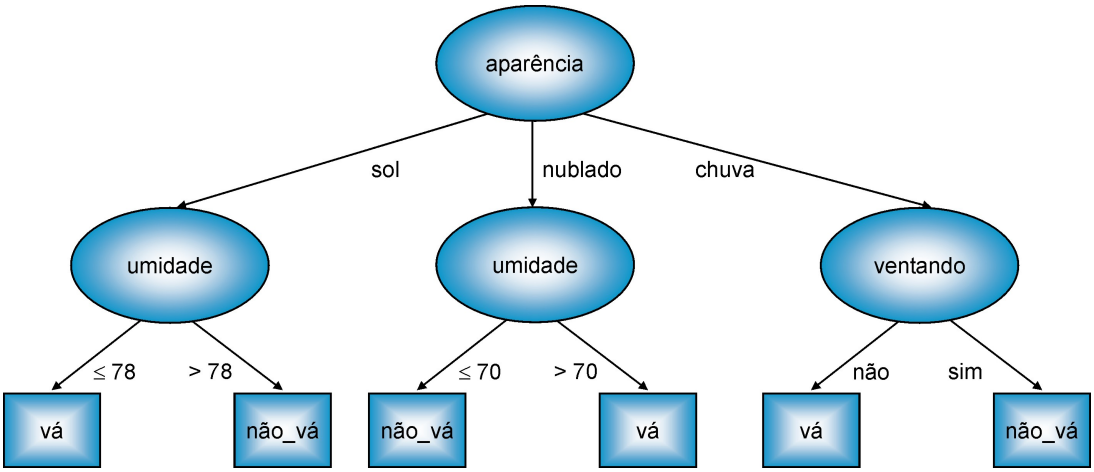

Figura 5.6: Construindo uma AD a partir dos exemplos de viagem (passo 2)

Entretanto, quando o aprendizado ocorre em exemplos contendo ruído, um grau adequado de poda pode melhorar o desempenho em exemplos não vistos. De fato, a poda, em geral, elimina erros provenientes de ruídos ao invés de descartar informação relevante (Bratko 1990).

5.4 Indução de Regras Ordenadas

Como descrito na seção anterior, a indução de árvores de decisão recursivamente divide os exemplos em subconjuntos menores, tentando separar cada classe das demais. A indução de regras, por outro lado, o faz diretamente. Nesse processo, cada regra cobre um subconjunto de exemplos que pertencem a uma classe específica. Basicamente, há duas formas de indução de regras: regras ordenadas e não ordenadas.

A indução de regras ordenada trabalha de forma iterativa, cada iteração procura por um <complexo> que cobre um grande número de exemplos de uma mesma classe C_i e poucos de outras classes $C_j, j \neq i$.

Ao encontrar um <complexo>, os exemplos cobertos que pertencem à classe C_i sendo aprendida (assim como, eventualmente, alguns poucos exemplos de outras classes C_j, $j \neq i$

Teste	Exemplo	Aparência	Temperatura	Umidade	Ventando	Viajar?
if aparência = sol	T_1	sol	25	72	sim	vá
e umidade $\leq$ 78	T_3	sol	22	70	não	vá
if aparência = sol	T_2	sol	28	91	sim	não_vá
e umidade > 78	T_4	sol	23	95	não	não_vá
	T_5	sol	30	85	não	não_vá
if aparência = nublado	T_6	nublado	23	90	sim	vá
	T_7	nublado	29	78	não	vá
	T_8	nublado	19	65	sim	não_vá
	T_9	nublado	26	75	não	vá
	T_{10}	nublado	20	87	sim	vá
if aparência = chuva	T_{11}	chuva	22	95	não	vá
e ventando = não	T_{14}	chuva	25	81	não	vá
	T_{15}	chuva	21	80	não	vá
if aparência = chuva	T_{12}	chuva	19	70	sim	não_vá
e ventando = sim	T_{13}	chuva	23	80	sim	não_vá

Tabela 5.4: Podando a AD dos exemplos de viagem

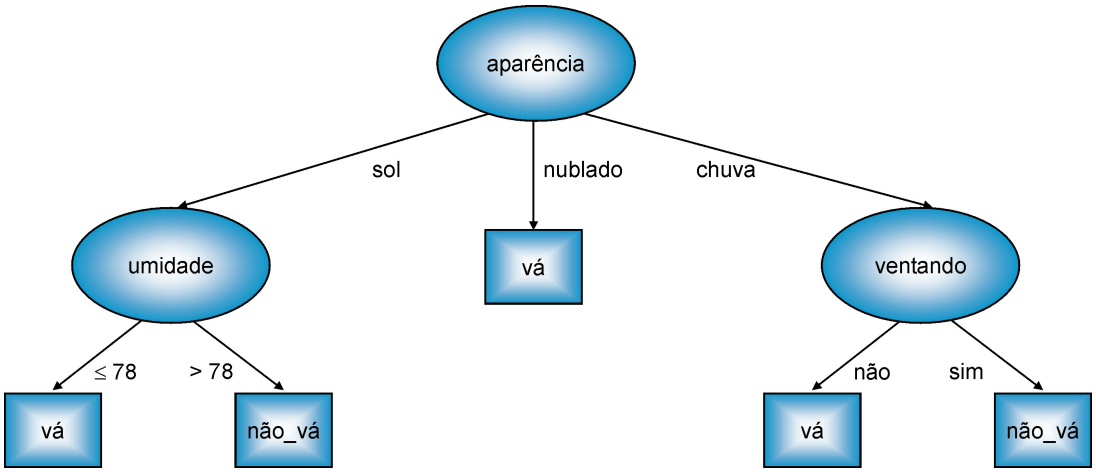

Figura 5.7: Construindo uma AD a partir dos exemplos de viagem (passo 3)

também cobertos pelo mesmo <complexo>) são removidos do conjunto de treinamento e a regra "**if** <complexo> **then** class = C_i" é adicionada no final da lista de regras. Esse processo se repete até que nenhum complexo possa ser encontrado.

5.4.1 Classificando Novos Exemplos

Para classificar novos exemplos, o classificador de regras ordenadas tenta cada regra em ordem até encontrar uma cujas condições sejam satisfeitas pelo novo exemplo. A classe do novo exemplo é aquela associada à classe predita pela regra. Assim, a *ordem* das regras é fundamental. Isto significa que uma regra isoladamente, exceto a primeira, não tem validade por si própria. Este é um aspecto muito importante da indução de regras que, muitas vezes, é negligenciado pelos usuários. Se nenhuma regra é satisfeita, existe uma regra default que, em geral, atribui ao novo exemplo a classe mais comum no conjunto de treinamento.

5.4.2 Interpretação Geométrica

As regras ordenadas podem ser vistas como uma árvore binária degenerada, uma vez que para classificar um novo exemplo, cada regra é testada até que uma dispare. Essa situação

é equivalente a um comando *if-then-elsif*, e o espaço de descrição pode ser considerado como particionado em regiões que não se sobrepõem, assim como ocorre para árvores de decisão.

5.4.3 Um Exemplo

O mesmo exemplo utilizado na Seção 5.3.6 será utilizado aqui. Na Tabela 5.5 é mostrada a lista de regras, onde R_1 é a primeira regra induzida, R_2 é a segunda e assim por diante. Os exemplos cobertos corretamente (CC) e cobertos incorretamente (CI) são mostrados para cada regra.

Considerando a indução da regra inicial R_1, todos os exemplos que satisfazem suas condições e estão na mesma classe predita pela conclusão da regra — exemplos T_1, T_3, T_7, T_8, T_9, T_{14}, T_{15} — assim como os exemplo que satisfazem as condições da regra mas não pertencem à classe predita vá — exemplos T_{12}, T_{13} — são removidos pelo algoritmo de indução de regras ordenadas e um **else** é introduzido na lista de regras antes de induzir a próxima regra. Após isso, a próxima regra é induzida pelo algoritmo e o processo continua. Observe que a última regra R_4 é a regra default, que somente dispara quando nenhum das regras anteriores R_1, R_2, R_3 disparam.

	Regra	CC	CI
R_1	if umidade $<$ 83 then classe = vá	$T_1, T_3, T_7, T_8, T_9, T_{14}, T_{15}$	T_{12}, T_{13}
R_2	else if temperatura $\geq$ 23 then classe = não_vá	T_2, T_4, T_5	T_6
R_3	else if aparência = chuva then classe = vá	T_{11}	
R_4	else classe = não_vá	T_{10}	

Tabela 5.5: Regras ordenadas para os exemplos de viagem

5.5 Indução de Regras Não Ordenadas

O algoritmo de regras ordenadas pode ser alterado convenientemente para induzir regras não ordenadas. A alteração principal consiste em iterar para cada classe C_i removendo apenas os exemplos cobertos e que são da classe C_i quando uma regra é encontrada. Assim, diferentemente de regras ordenadas, os exemplos das classes C_j, $j \neq i$ incorretamente cobertos pelo <complexo> encontrado devem permanecer porque agora cada nova regra deve ser comparada com todos os exemplos cobertos incorretamente. Exemplos cobertos que possuem a classe C_i sendo aprendida devem ser removidos para evitar que o algoritmo encontre a mesma regra novamente.

5.5.1 Classificando Novos Exemplos

Para classificar um novo exemplo, todas as regras são testadas e aquelas que disparam são coletadas. Se mais de uma classe é prevista pelas regras disparadas, o método usual de resolver qual classe deve ser associada ao novo exemplo consiste em associar a cada regra com a distribuição de exemplos cobertos entre classes e então somar essas distribuições para encontrar a classe mais provável. Por exemplo, considere as três regras:

if	cabeça=quadrada	**and**	segura=arma	**then**	classe=inimigo	cobre	[15,1]
if	tamanho=alto	**and**	voa=não	**then**	classe=amigo	cobre	[1,10]
if	aparência=bravo			**then**	classe=inimigo	cobre	[20,0]

Aqui, as duas classes são {inimigo,amigo} e [15,1] denota que a regra cobre 15 exemplos de treinamento de 'inimigo' e 1 de 'amigo'. Dado um novo exemplo com os seguinte atributos

cabeça quadrada, segura uma arma, alto, não voador e bravo, todas as três regras disparam. Somando os exemplos cobertos [36,11], a classe majoritária é então utilizada para prever a classe do novo exemplo — inimigo.

5.5.2 Interpretação Geométrica

A indução de regras não ordenada divide o espaço de descrição em regiões que podem se sobrepor, uma vez que cada exemplo pode ser coberto por mais de uma regra.

A fim de ilustrar, considere apenas dois atributos (X_1 e X_2) e duas classes (o e +). Na Figura 5.8 são mostradas quatro regras (uma para cada região retangular): dois retângulos com linhas sólidas que representam as duas regras tendo <classe = o> e dois retângulos com linhas pontilhadas representando as duas regras tendo <classe = +> . Logicamente, como em uma árvore de decisão, essa divisão do espaço de descrição corresponde a testes atributo-valor. Considerando uma combinação linear de atributos, o espaço de descrição seria dividido em retângulos não ortoginais.

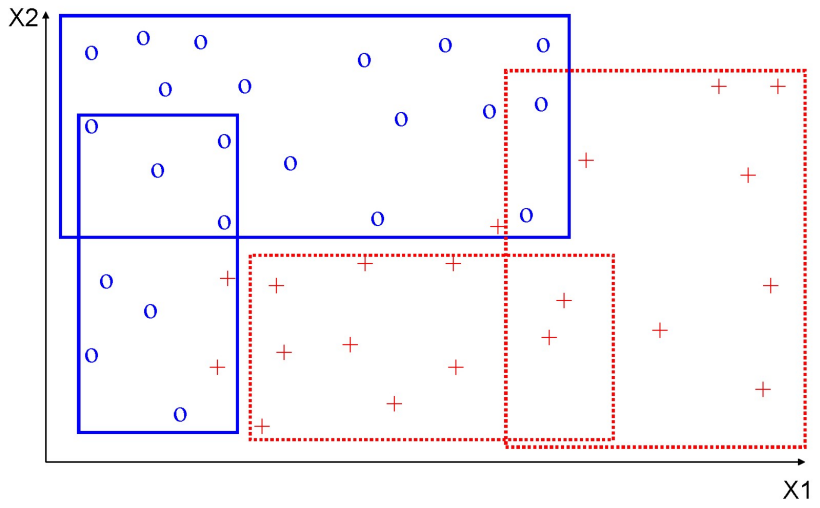

Figura 5.8: Regiões que se sobreprõem são geralmente formadas pela indução de regras não ordenadas no espaço de descrição

5.5.3 Um Exemplo

O mesmo exemplo utilizado na seção anterior será utilizado aqui. Na Tabela 5.6 é mostrado o conjunto de regras, onde R_1 é a primeira regra induzida, R_2 é a segunda e assim por diante. Novamente, os exemplos cobertos corretamente (CC) e cobertos incorretamente (CI) são mostrados para cada regra.

Considerando a indução da regra inicial R_1, todos os exemplos que satisfazem suas condições e estão na mesma classe predita pela conclusão da regra — exemplos T_6, T_7, T_8, T_9. Portanto, apenas esses exemplos são removidos do conjunto de treinamento e a próxima regra é induzida e o processo continua dessa forma. Assim como na indução de regras ordenadas, observe que a última regra R_6 é a regra default, que dispara somente se nenhum das regras anteriores $R_1, R_2, \ldots, R_5$ disparam.

	Regra	CC	CI
R_1	if aparência = nublado then classe = vá	T_6, T_7, T_8, T_9	T_{10}
R_2	if aparência = sol and umidade > 77 then classe = não_vá	T_2, T_4, T_5	
R_3	if temperatura > 24 then classe = vá	T_1, T_7, T_9, T_{14}	
R_4	if aparência = chuva and ventando = não then classe = vá	T_{11}, T_{14}, T_{15}	
R_5	if aparência = chuva and ventando = sim then classe = não_vá	T_{12}, T_{13}	
R_6	classe = vá	$T_1, T_3, T_6, T_7, T_9, T_{10}, T_{11}, T_{14}, T_{15}$	$T_2, T_4, T_5, T_8, T_{12}, T_{13}$

Tabela 5.6: Regras não ordenadas para os exemplos de viagem

5.6 Árvores *versus* Regras

O ponto forte da indução de regras é sua compreensibilidade e pouco espaço de armazenamento. Entretanto, o processo para induzir regras é mais lento que o para induzir árvore de decisão. Além disso, há muitos parâmetros a serem ajustados.

Existem duas variações básicas na indução de regras. Na primeira a estratégia, a lista de regras induzidas por uma árvore de decisão é reescrita como um conjunto parcialmente ordenado de regras (Quinlan 1987; Quinlan 1993), ou seja, as regras são agrupadas por classes e não são ordenadas dentro de uma mesma classe (ou seja, é possível trocar a ordem entre regras prevendo uma mesma classe), mas são ordenadas intra-classes. É importante notar que tal estratégia não consiste apenas na simples reescrita da árvore de decisão para um conjunto de regras. Na verdade, ela generaliza as regras desconsiderando condições supérfluas, ou seja, condições irrelevantes que não afetam a conclusão, sem afetar a precisão e mantendo as regras mais importantes.

Na segunda estratégia, as regras são induzidas diretamente a partir dos exemplos (Clark & Niblett 1989). Assim que uma regra é encontrada, os exemplos de treinamento são removidos e o processo se repete. O conjunto de regras resultante pode ainda ser refinado através da remoção de condições supérfluas aplicando-se algum teste estatístico.

Até o momento, para uma vasta gama de aplicações, os resultados utilizando indução de regras não têm sido consistentemente melhores que aqueles obtidos com árvores de decisão. É pouco provável que seja desenvolvido um indutor de regras que apresente desempenho e velocidade equivalentes a um indutor de árvore de decisão.

5.7 Avaliação de Regras

Considerando cada regra no formato $L \rightarrow R$, sua correspondente matriz de contingência é mostrada na Tabela 5.7 na próxima página (Lavrač, Flach, & Zupan 1999). Nesta tabela, L denota o conjunto de exemplos para os quais a condição da regra é verdadeira e seu complemento $\bar{L}$ denota o conjunto de exemplos para os quais a condição da regra é falsa e analogamente para R e $\bar{R}$. LR denota o conjunto de exemplos $L \cap R$ no qual ambos L e R são verdadeiros, $L\bar{R}$ denota o conjunto de exemplos $L \cap \bar{R}$ no qual L é verdadeiro e R é falso e assim por diante.

Por generalidade, denota-se a cardinalidade de um conjunto A por a, ou seja, $a = |A|$. Assim, l denota o número de exemplos no conjunto L, ou seja, $l = |L|$, r denota o número de exemplos no conjunto R, ou seja $r = |R|$, lr denota o número de exemplos no conjunto LR com $lr = |LR|$ e assim por diante. Como anteriormente, n indica o número total de exemplos.

A freqüência relativa $|A|/n = a/n$ associada ao subconjunto A é denotada por $p(A)$, onde A é um subconjunto dos n exemplos. Dessa forma, a freqüência relativa é usada como uma estimativa de probabilidade. A notação $p(A|B)$ segue sua definição habitual em probabilidade,

	L	$\bar{L}$	
R	lr	$\bar{l}r$	r
$\bar{R}$	$l\bar{r}$	$\bar{l}\bar{r}$	$\bar{r}$
	l	$\bar{l}$	n

Tabela 5.7: Matriz de contingência para a regra $L \rightarrow R$

dada por (5.1), onde A e B são ambos subconjuntos dos n exemplos.

$$p(A|B) = \frac{p(A \cap B)}{p(B)} = \frac{p(AB)}{p(B)} = \frac{\frac{|AB|}{n}}{\frac{|B|}{n}} = \frac{\frac{ab}{n}}{\frac{b}{n}} = \frac{ab}{b} \tag{5.1}$$

Várias medidas podem ser usadas para avaliar o desempenho de um classificador, sendo a precisão a mais comum. Entretanto, com novos problemas a serem tratados, novas medidas considerando novidade, simplicidade e facilidade de compreensão humana são necessárias (Freitas 1998a; Freitas 1998c; Freitas 1999b; Todorovski, Flach, & Lavrač 2000).

Utilizando como base a matriz de contingência, é possível definir a maioria das medidas sobre regras, por exemplo, a confiabilidade positiva prel, confiabilidade negativa nrel, suporte sup, sensitividade sens, especificidade spec, precisão total tacc e cobertura cov definidas pelas equações (5.2) até (5.8), respectivamente.

$$\mathsf{prel}(L \rightarrow R) = p(R|L) = \frac{lr}{l} \tag{5.2}$$

$$\mathsf{nrel}(L \rightarrow R) = p(\bar{R}|\bar{L}) = \frac{\bar{l}\bar{r}}{\bar{l}} \tag{5.3}$$

$$\mathsf{sup}(L \rightarrow R) = p(LR) = \frac{lr}{n} \tag{5.4}$$

$$\mathsf{sens}(L \rightarrow R) = p(L|R) = \frac{lr}{r} \tag{5.5}$$

$$\mathsf{spec}(L \rightarrow R) = p(\bar{L}|\bar{R}) = \frac{\bar{l}\bar{r}}{\bar{r}} \tag{5.6}$$

$$\mathsf{tacc}(L \rightarrow R) = p(LR) + p(\bar{L}\bar{R}) = \frac{lr + \bar{l}\bar{r}}{n} \tag{5.7}$$

$$\mathsf{cov}(L \rightarrow R) = p(L) = \frac{l}{n} \tag{5.8}$$

Além dessas medidas, Piatetsky-Shapiro (1991) propõe a novidade nov (*novelty*), também definida em Lavrač, Flach, & Zupan (1999) juntamente com a satisfação sat (*satisfaction*), conforme (5.9) e (5.10), respectivamente.

$$\mathsf{nov}(L \rightarrow R) = p(LR) - p(L)p(R) = \frac{lr}{n} - \frac{l \cdot r}{n^2} \tag{5.9}$$

$$\mathsf{sat}(L \rightarrow R) = \frac{p(\bar{R}) - p(\bar{R}|L)}{p(\bar{R})} = 1 - \frac{n \cdot l\bar{r}}{l \cdot \bar{r}} \tag{5.10}$$

Considerando L e R, a novidade é definida verificando se LR é independente deles. Isto pode ser obtido comparando o resultado observado lr contra o valor esperado sob a consideração de independência $\frac{l \cdot r}{n}$. Quanto mais o valor observado diferir do valor esperado, maior a probabilidade de que exista uma associação verdadeira e inesperada entre L e R. Pode ser demonstrado que $-0,25 \leq \mathsf{nov} \leq 0,25$: quanto maior um valor positivo (perto de 0,25), mais forte é a associação entre L e R, enquanto que quanto menor um valor negativo (perto de $-0,25$), mais forte é a associação entre L e $\bar{R}$.

Já a satisfação é o aumento relativo na precisão entre a regra $L \rightarrow \mathsf{verdade}$ e a regra $L \rightarrow R$. Segundo Lavrač, Flach, & Zupan (1999) esta medida é indicada para tarefas voltadas à descoberta de conhecimento, sendo capaz de promover um equilíbrio entre regras com diferentes condições e conclusões.

5.8 Perspectivas Futuras

Existem várias propostas para construir classificadores simbólicos, ou seja, para descrever de uma forma compreensível pelo usuário o conceito induzido. Duas dessas formas, a indução de árvores de decisão e de regras foram descritas neste capítulo.

Há também um aspecto do aprendizado humano que não tem sido ainda bem explorado pelas técnicas de AM, que está relacionado ao fato de que seres humanos aprendem muitos conceitos em paralelo, refinando e melhorando constantemente o conhecimento adquirido.

A investigação de estruturas diferentes, que podem ser apropriadas para diferentes contextos, bem como o entendimento do seu poder e limitação são necessários para o uso com êxito de Aprendizado de Máquina. Quanto maior a compreensão sobre as estruturas fundamentais utilizadas pelos classificadores, mais adequadamente pode-se aplicar ou alterá-las com base no conhecimento do domínio. Assim, um outro aspecto de técnicas e algoritmos de AM simbólico que tem despertado interesse e necessita ser melhor investigado está relacionado com a compreensibilidade/qualidade do conhecimento induzido e não somente com a precisão do classificador.

5.9 Referências Comentadas / Leitura Adicional

O indutor CART foi desenvolvido por estatísticos durante aproximadamente o mesmo período que ID3, no final dos anos 1970s. Esses indutores constroem árvores de decisão, sendo similares e muito eficientes. Breiman, Friedman, Olshen, & Stone (1984) escreveram um excelente livro sobre árvores de decisão.

Existem inúmeras ferramentas que implementam técnicas de AM. Entre elas destacam-se $\mathcal{MLC}$++ que é uma biblioteca em C++, desenvolvida em 1993 na Universidade de Stanford (**?**; **?**) e em 1995 passou a estar sob a responsabilidade da Silicon Graphics. O projeto tem como objetivo facilitar o uso dos algoritmos de AM bem como auxiliar pesquisadores da área em experimentos com novos algoritmos ou com modificações nos algoritmos existentes.

A biblioteca contém implementados os principais algoritmos de indução de diferentes paradigmas, como visto no capítulo anterior. Dentre algumas facilidades, $\mathcal{MLC}$++ fornece um formato padrão de entrada de dados, similar a Tabela 4.1 na página 44, para todos os algoritmos; obtenção de estatísticas de desempenho, tais como precisão, taxa de aprendizado e matriz de confusão; e visualização gráfica das estruturas aprendidas, por exemplo, árvores de decisão. Alguns dos algoritmos suportam visualização dos classificadores e podem gerar saídas para o software MineSet™. Entretanto, o código disponível data de 1997 e sua compilação não é trivial de ser efetuada.

MineSet ™ é um produto da Silicon Graphics para análise exploratória de dados http://www.sgi.com/. Combina várias ferramentas integradas e interativas para acesso e transformação de dados, Mineração de Dados e visualização. Este software usa $\mathcal{MLC}$++ como base para os algoritmos de indução. Entretanto, por se tratar de um produto comercial, alguns detalhes, sobre a implementação e funcionamento interno de suas ferramentas são omitidos, o que dificulta sua análise por pesquisadores da área de AM.

Weka é um conjunto de algoritmos de AM, escritos em Java e de domínio público, podendo ser obtido em http://www.cs.waikato.ac.nz/~ml/weka/. Diferentemente da biblioteca $\mathcal{MLC}$++, que possui interfaces para indutores já existentes, Weka adota uma outra abordagem. Todos os algoritmos são implementados em Java, tanto novos como aqueles pré-existentes.

Ainda que esse processo de recodificar algoritmos padroniza as interfaces e produza código uniforme, as novas versões dos algoritmos originais podem não ser disponibilizadas na Weka, pois exigem a conversão em código Java. Além disso, a recodificação de algoritmos sempre está sujeita a falhas, as quais podem causar um comportamento anômalo do algoritmo em Java que não ocorre no código fonte original.