

Katti Faceli

Ana Carolina Lorena

João Gama

André C. P. L. F. de Carvalho

INTELIGÊNCIA ARTIFICIAL

**Uma Abordagem de
Aprendizado de Máquina**

Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina



O GEN | Grupo Editorial Nacional reúne as editoras Guanabara Koogan, Santos, Roca, AC Farmacêutica, Forense, Método, LTC, E.P.U. e Forense Universitária, que publicam nas áreas científica, técnica e profissional.

Essas empresas, respeitadas no mercado editorial, construíram catálogos inigualáveis, com obras que têm sido decisivas na formação acadêmica e no aperfeiçoamento de várias gerações de profissionais e de estudantes de Administração, Direito, Enfermagem, Engenharia, Fisioterapia, Medicina, Odontologia, Educação Física e muitas outras ciências, tendo se tornado sinônimo de seriedade e respeito.

Nossa missão é prover o melhor conteúdo científico e distribuí-lo de maneira flexível e conveniente, a preços justos, gerando benefícios e servindo a autores, docentes, livreiros, funcionários, colaboradores e acionistas.

Nosso comportamento ético incondicional e nossa responsabilidade social e ambiental são reforçados pela natureza educacional de nossa atividade, sem comprometer o crescimento contínuo e a rentabilidade do grupo.

Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina

Katti Faceli

Professora Adjunta da Universidade Federal de São Carlos

Ana Carolina Lorena

Professora Adjunta da Universidade Federal do ABC

João Gama

Professor Associado com Agregação da Universidade do Porto

André Carlos Ponce de Leon Ferreira de Carvalho

Professor Titular da Universidade de São Paulo



Os autores e a editora empenharam-se para citar adequadamente e dar o devido crédito a todos os detentores dos direitos autorais de qualquer material utilizado neste livro, dispondo-se a possíveis acertos caso, inadvertidamente, a identificação de algum deles tenha sido omitida.

Não é responsabilidade da editora nem dos autores a ocorrência de eventuais perdas ou danos a pessoas ou bens que tenham origem no uso desta publicação.

Apesar dos melhores esforços dos autores, do editor e dos revisores, é inevitável que surjam erros no texto. Assim, são bem-vindas as comunicações de usuários sobre correções ou sugestões referentes ao conteúdo ou ao nível pedagógico que auxiliem o aprimoramento de edições futuras. Os comentários dos leitores podem ser encaminhados à LTC – Livros Técnicos e Científicos Editora Ltda.

Direitos exclusivos para a língua portuguesa
Copyright © 2011 by
LTC – Livros Técnicos e Científicos Editora Ltda.
Uma editora integrante do GEN | Grupo Editorial Nacional

Reservados todos os direitos. É proibida a duplicação ou reprodução deste volume, no todo ou em parte, sob quaisquer formas ou por quaisquer meios (eletrônico, mecânico, gravação, fotocópia, distribuição na internet ou outros), sem permissão expressa da editora.

Travessa do Ouvidor, 11
Rio de Janeiro, RJ — CEP 20040-040
Tels.: 21-3543-0770 / 11-5080-0770
Fax: 21-3543-0896
ltc@grupogen.com.br
www.ltceditora.com.br

Capa: Máquina Voadora DG

CIP-BRASIL. CATALOGAÇÃO-NA-FONTE
SINDICATO NACIONAL DOS EDITORES DE LIVROS, RJ

Inteligência Artificial : Uma Abordagem de Aprendizagem de Máquina / Katti Faceli... [et al.]. - Rio de Janeiro : LTC, 2011.
28cm

Inclui bibliografia e índice
ISBN 978-85-216-1880-5

1. Inteligência artificial. 2. Aprendizado por máquina. 3. Redes neurais. 4. Programação genética (Ciência da computação). 4. Algoritmos genéticos. I. Faceli, Katti.

11-2889.

CDD: 006.3
CDU: 004.81

Prefácio

Este livro trata do tema Inteligência Artificial com o foco em Aprendizado de Máquina (AM) e pode ser utilizado como livro-texto ou material de apoio em cursos de graduação e de pós-graduação nos tópicos Inteligência Artificial, Aprendizado de Máquina, Mineração de Dados, Análise de Dados e Sistemas Inteligentes.

Esperamos que este livro, ao mesmo tempo que introduza o leitor aos principais aspectos de AM e a temas de pesquisa recentes, sirva de alicerce para a realização de pesquisas que levem ao crescimento e fortalecimento da área. Esperamos ainda que o livro estimule o leitor a utilizar as várias técnicas nele abordadas na solução de problemas reais.

O texto foi concebido em largura: foca grande parte dos temas clássicos de aprendizado automático, desde o aprendizado supervisionado ao aprendizado não supervisionado, os métodos simbólicos, probabilísticos e subsimbólicos, às tendências atuais e aplicações de aprendizado a problemas de interesses econômico e social. Embora existam outros livros de Inteligência Artificial escritos na língua portuguesa ou traduzidos de livros publicados no exterior, os livros escritos na língua portuguesa em geral oferecem uma cobertura de temas mais tradicionais de Inteligência Artificial, como lógica, algoritmos de busca e representação de conhecimento. Livros nacionais que abordam temas relacionados com Aprendizado de Máquina se dedicam a tópicos específicos, como Redes Neurais e Algoritmos Genéticos. Livros nacionais que abordam uma maior parcela do conteúdo coberto por este livro cobrem apenas parte dos temas de Aprendizado de Máquina e são coletâneas de capítulos escritos por autores de forma independente. Os livros traduzidos de obras publicadas no exterior podem apresentar problemas inerentes a traduções, como propagação de erros encontrados no original, traduções baseadas em interpretações do tradutor e possível defasagem de conteúdo por causa do período decorrido entre a concepção do material original e a publicação da versão traduzida.

Agradecimentos

Inicialmente gostaríamos de agradecer às nossas famílias: João, Simoni, Amanda e Roberto; Dimas e Amanda; Julia, Rita e Luis; Valéria, Beatriz, Gabriela e Mariana, pelo amor, incentivo, paciência e compreensão pelas horas de convivência a que tivemos que abdicar para a produção deste livro.

Agradecemos aos amigos e colegas pelo apoio e incentivo à confecção deste livro. Em especial, à professora Tiemi C. Sakata, pela leitura e valioso auxílio com as correções de todo o texto, a Bruno Feres de Souza e a João Mendes Moreira pela leitura e valiosos comentários.

Gostaríamos de agradecer também aos nossos ex-alunos e atuais alunos de graduação e pós-graduação pela ajuda em partes deste livro, em especial a (em ordem alfabética) André Rossi, Carlos Ferreira, Cláudia Milaré, Cláudio Capoli, Márcia Oliveira, Ricardo Cerri e Faimison Rodrigues Porto pelas correções enviadas.

Finalmente agradecemos às nossas universidades, Universidade Federal do ABC, Universidade Federal de São Carlos, Universidade de São Paulo e Universidade do Porto, e às agências de fomento à pesquisa do Brasil e de Portugal, CAPES, CNPq, FAPESP, FCT e FINEP pelo apoio recebido para a preparação de material didático e realização das pesquisas que deram origem a este livro. Em especial gostaríamos de agradecer o financiamento ao projeto FCT-CAPES (224/09).

Sobre os Autores

Katti Faceli é Professora Adjunta da Universidade Federal de São Carlos, *campus* de Sorocaba. Possui graduação em bacharelado em Ciência da Computação pela Universidade de São Paulo (1998), mestrado (2001) e doutorado (2006) em Ciência da Computação e Matemática Computacional pela Universidade de São Paulo e pós-doutorado em Ciência da Computação pela Universidade de São Paulo - São Carlos (2008). Tem experiência na área de Ciência da Computação, com ênfase em Inteligência Artificial, atuando principalmente nos seguintes temas: aprendizado de máquina, bioinformática, análise de agrupamento e sistemas inteligentes híbridos. Possui vários trabalhos publicados em periódicos e conferências nacionais e internacionais nesses temas.

Ana Carolina Lorena é Professora Adjunta do Centro de Matemática, Computação e Cognição (CMCC) da Universidade Federal do ABC (UFABC). Possui graduação em bacharelado em Ciência da Computação pelo Instituto de Ciências Matemáticas e de Computação (ICMC), da Universidade de São Paulo (USP), *campus* de São Carlos, concluída em 2001. Possui doutorado e pós-doutorado em Ciência da Computação e Matemática Computacional também pelo ICMC-USP-São Carlos, concluídos em 2006 e 2007. É líder do Grupo Interdisciplinar de Mineração de Dados e Aplicações (GIMDA) na UFABC. Tem experiência em Ciência da Computação, atuando principalmente na área de Inteligência Artificial, nos seguintes temas: mineração de dados, aprendizado de máquina supervisionado, classificação de dados, *support vector machines* e sistemas inteligentes híbridos.

João Gama é Professor e pesquisador no LIAAD, o Laboratório de Inteligência Artificial e Análise de Dados da Universidade do Porto (Portugal). Possui doutorado em Ciência da Computação pela mesma instituição. A sua área de pesquisa é a aprendizagem de máquina, principalmente em fluxos contínuos de dados. Publicou mais de 100 artigos científicos em conferências e revistas internacionais. Organizou várias conferências internacionais (ECML-PKDD 2005, DS09, ADMA09, IDA 11) e séries de workshops em *Data Streams Analysis*. É autor de um livro recente em *Knowledge Discovery from Data Streams*.

André C. Ponce de Leon Ferreira de Carvalho é Professor Titular da Universidade de São Paulo. Possui graduação e mestrado em Ciência da Computação pela Universidade Federal de Pernambuco (1987), Ph.D. em Electronics pela University of Kent (1994), Reino Unido. É pós-doutorado pela Universidade do Porto (Portugal). Foi Associate Professor na University of Guelph, Canadá, Pesquisador Visitante na Universidade do Porto (2005-2006) e Professor Visitante na University of Kent (2010-2011). Atua na área de Aprendizado de Máquina, principalmente em mineração de dados, computação bioinspirada e sistemas inteligentes híbridos, com a orientação de vários mestrados e doutorados. Possui vários trabalhos publicados em diversos periódicos e conferências, além de ter organizado conferências e números especiais de periódicos. É membro do Centro de Matemática e Estatística Aplicada à Indústria do ICMC-USP e realiza atividades de consultoria a empresas sobre a utilização de Aprendizado de Máquina em problemas reais.

Material Suplementar

Este livro conta com materiais suplementares.

O acesso é gratuito, bastando que o leitor se cadastre em
<http://gen-io.grupogen.com.br>.



GEN-IO (GEN | Informação Online) é o repositório de material suplementar e de serviços relacionados com livros publicados pelo GEN | Grupo Editorial Nacional, o maior conglomerado brasileiro de editoras do ramo científico-técnico-profissional, composto por Guanabara Koogan, Santos, Roca, AC Farmacêutica, Forense, Método, LTC, E.P.U. e Forense Universitária.

Notação

n	Número de objetos de um conjunto de dados
d	Número de dimensões (atributos) dos objetos de um conjunto de dados
X	Conjunto de dados
x_i	i -ésimo objeto do conjunto de dados X
\mathbf{x}	Vetor de atributos de um objeto qualquer
\mathbf{x}^j	j -ésima coluna de atributos do conjunto de dados X
$y_{i(k)}$	Rótulo do objeto x_i
\mathbf{x}_i	i -ésimo objeto do cluster C_k
\mathbf{x}_i^j ou x_{ij}	Valor do j -ésimo atributo do i -ésimo objeto do conjunto de dados X
\bar{x}_i	Média dos atributos do i -ésimo objeto do conjunto de dados X
\bar{x}_j	Média dos valores da j -ésima coluna de atributos do conjunto de dados X
$d(x_i, x_j)$	Distância entre os objetos x_i e x_j
$s(x_i, x_j)$	Similaridade entre os objetos x_i e x_j
S_{nxn}	Matriz de similaridade/dissimilaridade entre pares de objetos
S_{ij}	Similaridade/dissimilaridade entre os objetos x_i e x_j na matriz S
$ A $	Número de elementos do conjunto A
π^i	i -ésima partição
C_k^i	k -ésimo cluster da i -ésima partição
C_k	Conjunto de objetos da classe C_k ou k -ésimo cluster de uma partição qualquer
$\bar{x}^{(k)}$	Centroide do cluster C_k
k	Número de classes ou clusters
k^i	Número de clusters da i -ésima partição (π^i)
$\hat{f}(x_i)$	Hipótese aplicada ao objeto x_i
$f(x_i)$	Função objetivo aplicada ao objeto x_i
M_c	Matriz de confusão
Y	Domínio da variável aleatória y

Sumário

Notação

1 Introdução 1

- 1.1 Inteligência Artificial e Aprendizado de Máquina 2
- 1.2 Indução de Hipóteses 4
- 1.3 Viés Indutivo 5
- 1.4 Tarefas de Aprendizado 5
- 1.5 Estrutura do Livro 7

Parte I Preparação de Dados 9

2 Análise de Dados 12

- 2.1 Caracterização de Dados 12
 - 2.1.1 Tipo 14
 - 2.1.2 Escala 15
- 2.2 Exploração de Dados 16
 - 2.2.1 Dados univariados 17
 - 2.2.2 Dados multivariados 23
- 2.3 Considerações Finais 28

3 Pré-processamento de Dados 29

- 3.1 Eliminação Manual de Atributos 30
- 3.2 Integração de Dados 30
- 3.3 Amostragem de Dados 31
- 3.4 Dados Desbalanceados 33
- 3.5 Limpeza de Dados 34
 - 3.5.1 Dados incompletos 34
 - 3.5.2 Dados inconsistentes 37
 - 3.5.3 Dados redundantes 37
 - 3.5.4 Dados com ruídos 39
- 3.6 Transformação de Dados 41
 - 3.6.1 Conversão simbólico-numérico 41
 - 3.6.2 Conversão numérico-simbólico 44
 - 3.6.3 Transformação de atributos numéricos 44
- 3.7 Redução de Dimensionalidade 46
 - 3.7.1 Agregação 47
 - 3.7.2 Seleção de atributos 47
 - 3.7.3 Técnicas de ordenação 49
 - 3.7.4 Técnicas de seleção de subconjunto 50
- 3.8 Considerações Finais 51

Parte II Modelos Preditivos 53

4 Métodos Baseados em Distâncias 58

- 4.1 Introdução 58

4.2	O Algoritmo do 1 - Vizinho Mais Próximo	59
4.2.1	Superfícies de decisão	60
4.2.2	Distâncias	61
4.3	O Algoritmo k -NN	61
4.4	Análise do Algoritmo	62
4.4.1	Aspectos positivos	63
4.4.2	Aspectos negativos	63
4.5	Desenvolvimentos	64
4.6	Raciocínio Baseado em Casos	66
4.6.1	Representação de casos	66
4.6.2	O ciclo de raciocínio baseado em casos	67
4.7	Considerações Finais	68

5 Métodos Probabilísticos 70

5.1	Aprendizado Bayesiano	71
5.1.1	O problema de inferência e o teorema de Bayes	73
5.2	Classificador <i>Naive Bayes</i>	73
5.2.1	Detalhes de implementação	74
5.2.2	Um exemplo ilustrativo	75
5.2.3	Análise do algoritmo	76
5.2.4	Desenvolvimentos	77
5.3	Redes Bayesianas para Classificação	78
5.3.1	Classificadores bayesianos com k -dependências	80
5.4	Considerações Finais	82

6 Métodos Baseados em Procura 83

6.1	Árvores de Decisão e Regressão	83
6.1.1	Indução de árvores de decisão e regressão	84
6.1.2	Estratégias de poda	91
6.1.3	Valores desconhecidos	94
6.1.4	Discussão: vantagens e desvantagens	95
6.2	Regras de Decisão	97
6.2.1	Por que regras de decisão?	97
6.2.2	De árvores de decisão às regras de decisão	98
6.2.3	O algoritmo da cobertura	99
6.3	Outros Modelos para Árvores de Previsão	104
6.3.1	Árvores de modelos	104
6.3.2	Árvores de opção	105
6.4	Considerações Finais	106

7 Métodos Baseados em Otimização 107

7.1	Redes Neurais Artificiais	107
7.1.1	Sistema nervoso	108
7.1.2	Componentes básicos das RNAs	109
7.1.3	Redes Perceptron e Adaline	113
7.1.4	Perceptron multicamadas	115
7.1.5	Algoritmo <i>back-propagation</i>	117
7.1.6	Projeto da arquitetura de uma RNA	120
7.1.7	Discussão: vantagens e desvantagens	121
7.2	Máquinas de Vetores de Suporte	122
7.2.1	Teoria de aprendizado estatístico	122

7.2.2	SVMs lineares	125
7.2.3	SVMs não lineares	130
7.2.4	SVMs em problemas de regressão	133
7.2.5	Discussão: vantagens e desvantagens	135
7.3	Considerações Finais	136
8	Modelos Múltiplos Preditivos	137
8.1	Combinando Previsões de Classificadores	139
8.1.1	Métodos de votação <i>versus</i> métodos de seriação	140
8.1.2	Métodos dinâmicos <i>versus</i> métodos estáticos	141
8.2	Combinando Classificadores Homogêneos	144
8.2.1	Métodos baseados em amostragem dos exemplos de treinamento	144
8.2.2	Métodos baseados na amostragem do conjunto de atributos	149
8.2.3	Métodos baseados na injeção de aleatoriedade	149
8.2.4	Métodos baseados na perturbação dos exemplos de teste	150
8.3	Combinando Classificadores Heterogêneos	150
8.3.1	Generalização em pilha	151
8.3.2	Generalização em cascata	152
8.3.3	Meta-aprendizado	155
8.3.4	Sistemas híbridos	156
8.4	Considerações Finais	157
9	Avaliação de Modelos Preditivos	158
9.1	Métricas de Erro	159
9.1.1	Métricas para classificação	159
9.1.2	Métricas para regressão	160
9.2	Amostragem	160
9.2.1	Holdout e amostragem aleatória	162
9.2.2	Validação cruzada	162
9.2.3	Bootstrap	163
9.3	Problemas de Duas Classes e o Espaço ROC	163
9.3.1	Medidas de desempenho	164
9.3.2	Análise ROC	165
9.4	Testes de Hipóteses	168
9.4.1	Comparando dois modelos	169
9.4.2	Comparando mais modelos	170
9.5	Decomposição Viés-Variância da Taxa de Erro	172
9.5.1	Definição de viés e variância	173
9.5.2	Medindo os componentes viés-variancia	174
9.6	Considerações Finais	175
Parte III	Modelos Descritivos	177
10	Mineração de Padrões Frequentes	180
10.1	Mineração de Conjuntos de Itens Frequentes	180
10.1.1	O espaço de busca	182
10.2	O Algoritmo Apriori	182
10.2.1	Regras de associação	183
10.2.2	Discussão	184
10.3	O Algoritmo FP-growth	185
10.4	Sumarização de Itemsets	187

10.4.1 Heurísticas para seleção de regras de associação	188
10.5 Considerações Finais	190

11 Análise de Agrupamentos 191

11.1 Definições Básicas	192
11.2 Etapas da Análise de Agrupamento	196
11.2.1 Preparação dos dados	196
11.2.2 Proximidade	198
11.2.3 Agrupamento	202
11.2.4 Validação	203
11.2.5 Interpretação	207
11.3 Considerações Finais	207

12 Algoritmos de Agrupamentos 208

12.1 Algoritmos Hierárquicos	208
12.2 Algoritmos Particionais Baseados em Erro Quadrático	212
12.3 Algoritmos Baseados em Densidade	214
12.4 Algoritmos Baseados em Grafo	215
12.5 Algoritmos Baseados em Redes Neurais	216
12.6 Algoritmos Baseados em Grid	217
12.7 Considerações Finais	218

13 Modelos Múltiplos Descritivos 219

13.1 Ensembles de Agrupamentos	220
13.1.1 Geração dos agrupamentos iniciais	221
13.1.2 Determinação da função consenso	222
13.1.3 Algumas técnicas ilustrativas	225
13.2 Agrupamento Multiobjetivo	231
13.3 Ensemble Multiobjetivo	233
13.4 Considerações Finais	235

14 Avaliação de Modelos Descritivos 236

14.1 Critérios de Validação	237
14.2 Critérios Relativos	242
14.2.1 Índices empregados em critérios relativos	242
14.2.2 Outras abordagens de validação relativa	245
14.3 Critérios Internos	248
14.3.1 Estatística Gap	249
14.4 Critérios Externos	251
14.4.1 Índice Rand	252
14.4.2 Índice Jaccard	252
14.4.3 Índice Fowlkes e Mallows	253
14.4.4 Índice Hubert normalizado	253
14.4.5 Índice Rand corrigido	253
14.4.6 Índice variação de informação	254
14.4.7 Comparação dos índices tradicionais para validação externa	254
14.5 Considerações Finais	255

Parte IV Tópicos Avançados 257

15 Aprendizado em Fluxos Contínuos de Dados	260
15.1 Desafios no Aprendizado em Fluxos Contínuos de Dados	261

15.2 Algoritmos Ilustrativos para Aprendizado em Fluxos de Dados	262
15.2.1 O algoritmo <i>árvore de decisão muito rápida</i>	262
15.2.2 Análise de agrupamentos em séries temporais contínuas	264
15.2.3 Redes neurais	266
15.3 Detecção de Mudança	266
15.4 Considerações Finais	269

16 Meta-aprendizado 270

16.1 Caracterização de Conjuntos de Dados	271
16.1.1 Caracterização direta	272
16.1.2 Caracterização por propriedades de modelos	273
16.1.3 Landmarking	273
16.2 Medidas de Avaliação dos Algoritmos	274
16.3 Formas de Apresentação de Sugestões	274
16.4 Recomendações a partir da Caracterização	275
16.5 Estudo de Casos	275
16.6 Considerações Finais	276

17 Decomposição de Problemas Multiclasse 278

17.1 Fase de Decomposição	279
17.1.1 Um-contra-todos	280
17.1.2 Todos-contra-todos	280
17.1.3 Códigos de correção de erros de saída	281
17.1.4 Decomposições hierárquicas	282
17.2 Fase de Reconstrução	285
17.3 Considerações Finais	287

18 Classificação Multirrótulo 288

18.1 Principais Abordagens	289
18.1.1 Abordagem independente de algoritmo	290
18.1.2 Abordagem dependente de algoritmo	293
18.2 Densidade e Cardinalidade de Rótulo	293
18.3 Medidas de Avaliação	294
18.4 Considerações Finais	296

19 Classificação Hierárquica 297

19.1 Tipos de Problemas de Classificação Hierárquica	297
19.2 Algoritmos para Classificação Hierárquica	299
19.2.1 Classificadores locais	301
19.2.2 Classificadores globais	302
19.3 Avaliação de Classificadores Hierárquicos	303
19.4 Considerações Finais	304

20 Computação Natural 305

20.1 Inteligência de Enxames	306
20.1.1 Otimização por colônia de formigas	306
20.1.2 Otimização por enxame de partículas	308
20.2 Computação Evolutiva	310
20.3 Considerações Finais	313

Parte V Aplicações 315

- 21 Agropecuária 318
- 22 Bioinformática 320
- 23 Ecologia e Meio Ambiente 323
- 24 Energia 326
- 25 Finanças 329
- 26 Mineração de Dados 331
- 27 Robótica 333
- 28 Saúde 335

Parte VI 339**Tendências Futuras 341****Referências Bibliográficas 343****Índice Remissivo 375**

Capítulo 1

Introdução

Problemas são resolvidos em computação por meio da escrita de um algoritmo ou pseudocódigo, que especifica passo a passo como o problema pode ser resolvido. No entanto, não é fácil escrever um programa de computador que realize com eficiência algumas tarefas que realizamos com facilidade no nosso dia a dia, como reconhecer pessoas pelo rosto ou pela fala. Que características dos rostos ou da fala serão consideradas? O que fazer para diferentes expressões faciais de uma mesma pessoa, alterações na face, como o uso de óculos ou bigode, cortes de cabelo, mudanças na voz por uma gripe ou estado de espírito? No entanto, os seres humanos conseguem realizar essas tarefas com relativa facilidade. Fazem isso por meio de reconhecimento de padrões, quando aprendem o que deve ser observado em um rosto ou na fala para conseguir identificar pessoas após terem tido vários exemplos de rostos ou falas com identificação clara.

Um bom médico consegue, dado o conjunto de sintomas e de resultados de exames clínicos, diagnosticar se um paciente está com problemas cardíacos. Para isso o médico utiliza conhecimento adquirido durante sua formação e experiência proveniente do exercício da profissão. Como escrever um programa de computador que, dados os sintomas e os resultados de exames clínicos, apresente um diagnóstico que seja tão bom quanto o de um médico experiente?

Também pode ser difícil escrever um programa que faça algumas análises de dados de venda de uma loja. Para descobrir quantas pessoas fizeram mais de uma compra em uma loja no ano anterior, programas para gerenciamento de bancos de dados podem ser facilmente utilizados. Entretanto, como escrever um programa que responda a questões mais complicadas como:

- Identificar conjuntos de produtos que são frequentemente vendidos em conjunto.
- Recomendar novos produtos a clientes que costumam comprar produtos semelhantes.
- Agrupar os consumidores da loja em grupos de forma a ter melhores resultados nas operações de marketing.

Apesar da dificuldade de escrever um programa de computador que possa lidar de forma eficiente com essas tarefas, o número de vezes em que tarefas tão complexas como essas precisam ser realizadas diariamente é muito grande. Isso, aliado ao volume de informações que precisam ser consideradas para sua realização, torna difícil ou mesmo impossível a sua realização por seres humanos.

Técnicas de Inteligência Artificial (IA), em particular de Aprendizado de Máquina (AM), têm sido utilizadas com sucesso em um grande número de problemas reais, incluindo os problemas citados anteriormente.

Este capítulo está organizado da seguinte forma. A Seção 1.1 apresenta a relação entre AM e IA, mostrando alguns exemplos da utilização de AM em problemas reais. Na Seção 1.2 é introduzida a relação entre um conjunto de dados e a qualidade da hipótese induzida por um algoritmo de AM. O conceito de viés indutivo, essencial para que o aprendizado ocorra, é discutido na Seção 1.3. A Seção 1.4 descreve as diferentes tarefas de aprendizado, que são agrupadas em aprendizado preditivo e aprendizado descritivo. Finalmente, a Seção 1.5 apresenta a estrutura dos capítulos do livro.

1.1 Inteligência Artificial e Aprendizado de Máquina

Até alguns anos atrás, a área de IA era vista como uma área teórica, com aplicações apenas em pequenos problemas curiosos, desafiadores, mas de pouco valor prático. Boa parte dos problemas práticos que precisam de computação era resolvida pela codificação em alguma linguagem de programação dos passos necessários para a sua solução.

A partir da década de 1970, houve uma maior disseminação do uso de técnicas de computação baseadas em IA para a solução de problemas reais. Muitas vezes, esses problemas eram tratados computacionalmente por meio da aquisição de conhecimento de especialistas de um dado domínio, por exemplo, do domínio da medicina, que era então codificado, frequentemente por regras lógicas, em um programa de computador. Esses programas eram conhecidos como Sistemas Especialistas ou Sistemas Baseados em Conhecimento.

O processo de aquisição do conhecimento normalmente envolvia entrevistas com os especialistas para descobrir que regras eles utilizavam quando da tomada de decisão. Esse processo possuía várias limitações, como subjetividade, decorrente do uso pelo especialista de sua intuição na tomada de decisão, e, muitas vezes, pouca cooperação por parte do especialista, por causa do seu receio de ser dispensado após repassar o conhecimento solicitado.

Nas últimas décadas, com a crescente complexidade dos problemas a serem tratados computacionalmente e do volume de dados gerados por diferentes setores, tornou-se clara a necessidade de ferramentas computacionais mais sofisticadas, que fossem mais autônomas, reduzindo a necessidade de intervenção humana e dependência de especialistas. Para isso, essas técnicas deveriam ser capazes de criar por si próprias, a partir da experiência passada, uma hipótese, ou função, capaz de resolver o problema que se deseja tratar. Um exemplo simples é a descoberta de uma hipótese na forma de uma regra ou conjunto de regras para definir que clientes de um supermercado devem receber material de propaganda de um novo produto, utilizando para isso dados de compras passados dos clientes cadastrados na base de dados do supermercado. A esse processo de indução de uma hipótese (ou aproximação de função) a partir da experiência passada dá-se o nome Aprendizado de Máquina (AM).

A capacidade de aprendizado é considerada essencial para um comportamento inteligente. Atividades como memorizar, observar e explorar situações para aprender fatos, melhorar habilidades motoras/cognitivas por meio de práticas e organizar conhecimento

novo em representações apropriadas podem ser consideradas atividades relacionadas ao aprendizado.

Existem várias definições de AM na literatura. Uma delas, apresentada em Mitchell (1997), define AM como:

“A capacidade de melhorar o desempenho na realização de alguma tarefa por meio da experiência.”

Em AM, computadores são programados para aprender com a experiência passada. Para tal, empregam um princípio de inferência denominado indução, no qual se obtêm conclusões genéricas a partir de um conjunto particular de exemplos. Assim, algoritmos de AM aprendem a induzir uma função ou hipótese capaz de resolver um problema a partir de dados que representam instâncias do problema a ser resolvido. Esses dados formam um conjunto, simplesmente denominado conjunto de dados (Seção 1.2).

Embora AM seja naturalmente associado à IA, outras áreas de pesquisa são importantes e têm contribuições diretas e significativas no avanço do AM, como Probabilidade e Estatística, Teoria da Computação, Neurociência, Teoria da Informação, para citar algumas. AM é uma das áreas de pesquisa da computação que mais tem crescido nos últimos anos. Diferentes algoritmos de AM, diferentes formas de utilizar os algoritmos existentes e adaptações de algoritmos são continuamente propostos. Além disso, surgem a todo instante novas variações nas características dos problemas reais a serem tratados.

Existem várias aplicações bem-sucedidas de técnicas de AM na solução de problemas reais, entre as quais podem ser citadas:

- Reconhecimento de palavras faladas;
- Predição de taxas de cura de pacientes com diferentes doenças;
- Detecção do uso fraudulento de cartões de crédito;
- Condução de automóveis de forma autônoma em rodovias;
- Ferramentas que jogam gamão e xadrez de forma semelhante a campeões;
- Diagnóstico de câncer por meio da análise de dados de expressão gênica.

Além do grande volume de aplicações que se beneficiam das características da área de AM, outros fatores têm favorecido a expansão dessa área, como o desenvolvimento de algoritmos cada vez mais eficazes e eficientes e a elevada capacidade dos recursos computacionais atualmente disponíveis.

Outras motivações para as pesquisas em AM incluem a possibilidade de aumentar a compreensão de como se dá o aprendizado nos seres vivos. Além disso, algumas tarefas são naturalmente mais bem definidas por meio de exemplos. Os modelos gerados são ainda capazes de lidar com situações não apresentadas durante seu desenvolvimento, sem necessariamente necessitar de uma nova fase de projeto.

1.2 Indução de Hipóteses

Para ilustrar a relação entre AM e indução de hipóteses, imagine um conjunto de dados composto de pacientes de um hospital. Nesse conjunto, que será denominado **hospital**, cada dado (também chamado objeto, exemplo, padrão ou registro) corresponde a um paciente e é uma tupla formada pelos valores de características ou atributos referentes ao paciente, que descrevem seus principais aspectos. Os atributos (também chamados campos ou variáveis) utilizados para cada paciente podem ser, por exemplo, sua identificação, nome, idade, sexo, estado de origem, sintomas e resultados de exames clínicos. Exemplos de sintomas podem ser presença e distribuição de manchas na pele, peso e temperatura do corpo.

Conforme será visto adiante, para algumas tarefas de aprendizado, um dos atributos é considerado um atributo de saída (também chamado atributo alvo ou atributo meta), cujos valores podem ser estimados utilizando os valores dos demais atributos, denominados atributos de entrada (também chamados atributos previsores). O objetivo de um algoritmo de AM utilizado nessas tarefas é aprender, a partir de um subconjunto dos dados, denominado conjunto de treinamento, um modelo ou hipótese capaz de relacionar os valores dos atributos de entrada de um objeto do conjunto de treinamento ao valor de seu atributo de saída.

Um requisito importante para algoritmos de AM é que eles sejam capazes de lidar com dados imperfeitos. Muitos conjuntos de dados apresentam algum tipo de problema, como presença de ruídos, dados inconsistentes, dados ausentes e dados redundantes. Algoritmos de AM devem, idealmente, ser robustos aos problemas presentes nos dados, minimizando sua influência no processo de indução de hipóteses. Entretanto, dependendo de sua extensão, esses problemas podem prejudicar o processo indutivo. Técnicas de pré-processamento são utilizadas com frequência para identificar e minimizar a ocorrência desses problemas.

Voltando ao exemplo dos pacientes, considere a situação em que um algoritmo de AM é utilizado para aprender uma hipótese (por exemplo, uma regra) capaz de diagnosticar pacientes de acordo com os valores associados aos seus atributos de entrada, representados por parte dos demais atributos. Os atributos referentes à identificação e nome do paciente não são considerados entradas relevantes, uma vez que não possuem relação nenhuma com o diagnóstico de uma doença.

O que se deseja, na verdade, é induzir uma hipótese capaz de fazer diagnósticos corretos para novos pacientes diferentes daqueles que foram utilizados para aprender a regra de decisão. Assim, uma vez induzida uma hipótese, é desejável que ela também seja válida para outros objetos do mesmo domínio ou problema que não fazem parte do conjunto de treinamento. A essa propriedade de uma hipótese continuar a ser válida para novos objetos dá-se o nome capacidade de generalização da hipótese. Para ser útil quando aplicada a novos dados, uma hipótese precisa apresentar uma boa capacidade de generalização.

Quando uma hipótese apresenta uma baixa capacidade de generalização, a razão pode ser que ela está superajustada aos dados (*overfitting*). Nesse caso, também é dito que a hipótese memorizou ou se especializou nos dados de treinamento. No caso inverso, o algoritmo de AM pode induzir hipóteses que apresentem uma baixa taxa de acerto mesmo no subconjunto de treinamento, configurando uma condição de subajustamento (*underfitting*). Essa situação pode ocorrer, por exemplo, quando os exemplos de treinamento disponíveis são pouco representativos ou o modelo usado é muito simples e não captura os

padrões existentes nos dados (Monard e Baranauskas, 2003). Na Seção 7.2.1, esses conceitos são ilustrados e discutidos novamente. São feitas então considerações e motivações sobre a escolha de modelos com boa capacidade de generalização.

1.3 Viés Indutivo

Quando um algoritmo de AM está aprendendo a partir de um conjunto de dados de treinamento, ele está procurando uma hipótese, no espaço de possíveis hipóteses, capaz de descrever as relações entre os objetos e que melhor se ajuste aos dados de treinamento.

Cada algoritmo utiliza uma forma ou representação para descrever a hipótese induzida. Por exemplo, redes neurais artificiais representam uma hipótese por um conjunto de valores reais, associados aos pesos das conexões da rede. Árvores de decisão utilizam uma estrutura de árvore em que cada nó interno é representado por uma pergunta referente ao valor de um atributo e cada nó externo está associado a uma classe. A representação utilizada define a preferência ou viés (*bias*) de representação do algoritmo e pode restringir o conjunto de hipóteses que podem ser induzidas pelo algoritmo. A Figura 1.1 ilustra o viés de representação utilizado por técnicas de indução de árvores de decisão, redes neurais artificiais e regras de decisão.

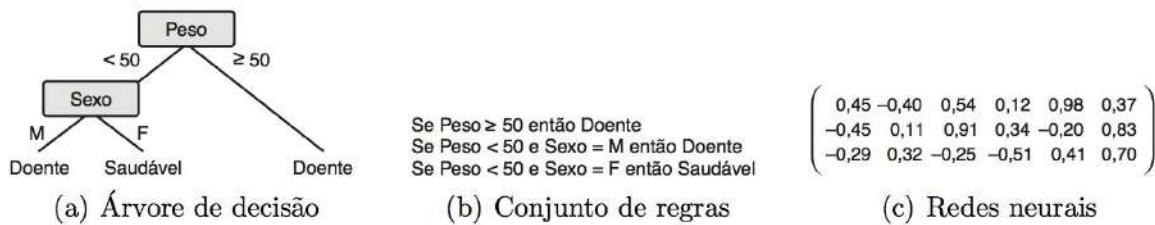


Figura 1.1 Diferentes vieses de representação.

Além do viés de representação, os algoritmos de AM possuem também um viés de busca. O viés de busca de um algoritmo é a forma como o algoritmo busca a hipótese que melhor se ajusta aos dados de treinamento. Ele define como as hipóteses são pesquisadas no espaço de hipóteses. Por exemplo, o algoritmo ID3, que é utilizado para indução de árvores de decisão, tem como viés de busca a preferência por árvores de decisão com poucos nós, conforme será apresentado no Capítulo 6.

Assim, cada algoritmo de AM possui dois vieses, um viés de representação e um viés de busca. O viés é necessário para restringir as hipóteses a serem visitadas no espaço de busca. Sem viés não haveria aprendizado/generalização. Os modelos seriam especializados para os exemplos individuais. Embora, à primeira vista, o viés pareça ser uma limitação dos algoritmos de AM, segundo Mitchell (1997), sem viés um algoritmo de AM não consegue generalizar o conhecimento adquirido durante seu treinamento para aplicá-lo com sucesso a novos dados.

1.4 Tarefas de Aprendizado

Algoritmos de AM têm sido amplamente utilizados em diversas tarefas, que podem ser organizadas de acordo com diferentes critérios. Um deles diz respeito ao paradigma

de aprendizado a ser adotado para lidar com a tarefa. De acordo com esse critério, as tarefas de aprendizado podem ser divididas em:

- Preditivas e
- Descritivas.

Em tarefas de previsão, a meta é encontrar uma função (também chamada de modelo ou hipótese) a partir dos dados de treinamento que possa ser utilizada para prever um rótulo ou valor que caracterize um novo exemplo, com base nos valores de seus atributos de entrada. Para isso, cada objeto do conjunto de treinamento deve possuir atributos de entrada e de saída.

Os algoritmos ou métodos de AM utilizados nessa tarefa induzem modelos preditivos. Esses algoritmos seguem o paradigma de aprendizado supervisionado. O termo supervisionado vem da simulação da presença de um “supervisor externo”, que conhece a saída (rótulo) desejada para cada exemplo (conjunto de valores para os atributos de entrada). Com isso, o supervisor externo pode avaliar a capacidade da hipótese induzida de prever o valor de saída para novos exemplos.

Em tarefas de descrição, a meta é explorar ou descrever um conjunto de dados. Os algoritmos de AM utilizados nessas tarefas não fazem uso do atributo de saída. Por isso, seguem o paradigma de aprendizado não supervisionado. Uma tarefa descritiva de agrupamento de dados, por exemplo, tem por meta encontrar grupos de objetos semelhantes no conjunto de dados. Outra tarefa descritiva é encontrar regras de associação que relacionam um grupo de atributos a outro grupo de atributos.

A Figura 1.2 apresenta uma hierarquia de aprendizado de acordo com os tipos de tarefas de aprendizado. No topo aparece o aprendizado indutivo, processo pelo qual são realizadas as generalizações a partir dos dados. Tem-se em seguida os tipos de aprendizado supervisionado (preditivo) e não supervisionado (descritivo). As tarefas supervisionadas se distinguem pelo tipo dos rótulos dos dados: discreto, no caso de classificação; e contínuo, no caso de regressão. As tarefas descritivas são genericamente divididas em: agrupamento, em que os dados são agrupados de acordo com sua similaridade; sumarização, cujo objetivo é encontrar uma descrição simples e compacta para um conjunto de dados; e associação, que consiste em encontrar padrões frequentes de associações entre os atributos de um conjunto de dados. Com exceção da sumarização, as demais tarefas serão descritas neste livro.



Figura 1.2 Hierarquia de aprendizado.

Deve ser observado que, apesar dessa divisão básica de modelos em preditivos e descriptivos, um modelo preditivo também provê uma descrição compacta de um conjunto de dados e um modelo descriptivo pode prover previsões após ser validado.

Uma tarefa de aprendizado que não se enquadra nas tarefas anteriores é a de aprendizado por reforço. Nessa tarefa, a meta é reforçar ou recompensar uma ação considerada positiva e punir uma ação considerada negativa. Um exemplo de tarefa de reforço é a de ensinar um robô a encontrar a melhor trajetória entre dois pontos. Algoritmos de aprendizado utilizados nessa tarefa, em geral, punem a passagem por trechos pouco promissores e recompensam a passagem por trechos promissores. Por causa do foco adotado para este livro, essa tarefa não será abordada.

1.5 Estrutura do Livro

Este livro tem por objetivo apresentar os principais conceitos e algoritmos de AM e mostrar como AM pode ser utilizado para a solução de problemas reais. Para isso, serão cobertos tanto temas tradicionais como resultados de pesquisas recentes na área.

De forma a agrupar os temas cobertos de uma maneira mais uniforme, os capítulos do livro foram organizados em cinco grandes temas ou módulos:

- **Preparação de dados:** engloba tópicos de descrição dos dados, análise estatística de dados e pré-processamento de dados.
- **Métodos preditivos:** é relacionado a aprendizado supervisionado e que, após definir os conceitos gerais referentes a esse tema, descreve os principais algoritmos de aprendizado preditivo, como hipóteses podem ser combinadas formando comitês, possíveis estratégias para planejar experimentos com esses métodos e as principais métricas empregadas para avaliar seu desempenho.
- **Métodos descriptivos:** é o equivalente do módulo anterior para aprendizado não supervisionado. Nesse módulo são abordados os principais conceitos básicos necessários para descrever essa abordagem de AM, os principais algoritmos utilizados, formas como eles podem ser avaliados e combinados. É também discutido como experimentos utilizando esses métodos podem ser planejados e avaliados.
- **Tópicos avançados:** incluem temas de pesquisas recentes na área de AM. Esses temas, que incluem fluxos de dados, meta-aprendizado, estratégias para classificação multiclasse, classificação hierárquica e classificação multirrótulo, são aplicados com sucesso em um grande número de problemas reais.
- **Aplicações:** ilustram alguns exemplos de aplicações reais relacionadas a diferentes domínios em que técnicas de AM têm sido empregadas com sucesso.

Esses tópicos foram cuidadosamente escolhidos para que os leitores tenham uma dosagem equilibrada em abrangência e profundidade dos temas básicos e avançados nas áreas de Inteligência Artificial que utilizam aprendizado para indução de modelos.

Parte I

Preparação de Dados

Conjuntos de Dados

A cada dia, uma enorme quantidade de dados é gerada. Existe uma estimativa de que a cada 20 meses dobra a quantidade de dados armazenada nos bancos de dados do mundo (Witten et al., 2011). Esses dados são gerados por transações financeiras, monitoramento ambiental, obtenção de dados clínicos e genéticos, captura de imagens, navegação na internet, apenas para citar algumas atividades. Os dados podem ainda assumir vários formatos diferentes, como séries temporais, *itemsets*, transações, grafos ou redes sociais, textos, páginas web, imagens (vídeos) e áudios (músicas). Com o aumento crescente da quantidade de dados gerada, tem aumentado muito a distância entre a quantidade de dados existente e a porção desses dados que é analisada e compreendida.

Conjuntos de dados são formados por objetos que podem representar um objeto físico, como uma cadeira, ou uma noção abstrata, como os sintomas apresentados por um paciente que se dirige a um hospital. Em geral, cada objeto é descrito por um conjunto de atributos de entrada ou vetor de características. Cada objeto corresponde a uma ocorrência dos dados. Cada atributo está associado a uma propriedade do objeto.

Formalmente, os dados podem ser representados por uma matriz de objetos $\mathbf{X}_{n \times d}$, em que n é o número de objetos e d é o número de atributos de entrada de cada objeto. O valor de d define a dimensionalidade dos objetos ou do espaço de objetos (também chamado de espaço de entradas ou espaço de atributos). Cada elemento dessa matriz, x_i^j , contém o valor da j -ésima característica para o i -ésimo objeto. Os d atributos também podem ser vistos como um conjunto de eixos ortogonais e os objetos, como pontos no espaço de dimensão d , chamado espaço de objetos.

A Figura 1.3 ilustra um exemplo de espaço de objetos. Nesse espaço, a posição de cada objeto é definida pelos valores de dois atributos de entrada ($d = 2$), nesse caso, os resultados de dois exames clínicos. O atributo de saída é representado pelo formato do objeto na figura: círculo para pacientes doentes e triângulo para pacientes saudáveis.

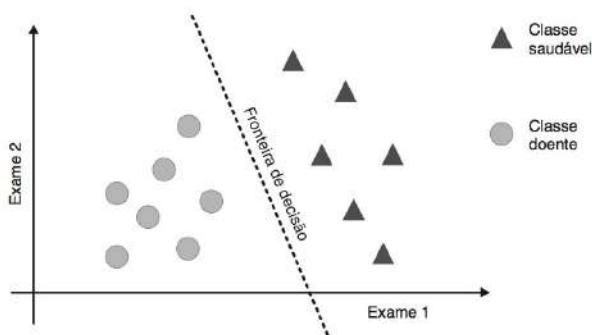


Figura 1.3 Espaço de objetos.

Apesar do crescente número de bases de dados disponíveis, na maioria das vezes não é possível utilizar algoritmos de AM diretamente sobre esses dados. Técnicas de pré-processamento são frequentemente utilizadas para tornar os conjuntos de dados mais adequados para o uso de algoritmos de AM. Essas técnicas podem ser agrupadas nos seguintes grupos de tarefas:

- Eliminação manual de atributos;
- Integração de dados;
- Amostragem de dados;
- Balanceamento de dados;
- Limpeza de dados;
- Redução de dimensionalidade;
- Transformação de dados.

Boa parte das empresas, órgãos do governo e outras organizações possuem seus dados armazenados em mais de uma base ou conjunto de dados. Assim, os dados podem vir de mais de uma fonte ou tabela atributo-valor. Quando dados presentes em diferentes conjuntos precisam ser utilizados por um algoritmo de AM, esses conjuntos devem ser integrados de forma a constituir um único conjunto ou tabela. Essa integração pode levar a inconsistências e redundâncias. Algoritmos de AM também podem ter dificuldades quando precisam lidar com uma quantidade muito grande de dados. Essa grande quantidade pode estar relacionada ao número de objetos, número de atributos ou ambos. Problemas como redundância e inconsistência muitas vezes estão relacionados a uma grande quantidade de dados. Técnicas de amostragem e de seleção de atributos têm sido empregadas nessas situações.

Em dados reais, a distribuição dos objetos entre as classes pode não ser uniforme. Assim, algumas classes podem ter um número de objetos muito superior a outras, formando um conjunto de dados desbalanceado. Alguns algoritmos de AM têm dificuldade de induzir um bom modelo a partir de conjuntos desbalanceados.

Boa parte dos conjuntos de dados reais apresenta problemas como presença de ruído e dados incompletos e/ou inconsistentes. Os dados podem estar incompletos por causa da ausência de valores. Os dados podem ser inconsistentes por causa de erros na sua geração, captação ou entrada. Grande parte dos algoritmos de AM tem seu desempenho afetado pela presença desses problemas. Para lidar com eles, diversas técnicas para limpeza de dados têm sido propostas e investigadas na literatura de AM.

Mesmo após a eliminação de atributos por especialistas no domínio, os atributos que restaram podem dificultar a tarefa de algoritmos de AM, devido a diferentes motivos, como a presença de um número muito grande de atributos, de atributos redundantes, irrelevantes e/ou inconsistentes.

Vários algoritmos de AM têm dificuldades em utilizar os dados no seu formato original. Para tratar desse problema, são realizadas transformações nos dados originais antes que eles sejam utilizados pelo algoritmo. Um exemplo de transformação é a transformação de valores simbólicos em valores numéricos.

Capítulo 2

Análise de Dados

A análise das características presentes em um conjunto de dados permite a descoberta de padrões e tendências que podem fornecer informações valiosas que ajudem a compreender o processo que gerou os dados. Muitas dessas características podem ser obtidas por meio da aplicação de fórmulas estatísticas simples. Outras podem ser observadas por meio do uso de técnicas de visualização.

Neste capítulo são descritas as principais características observadas para a descrição, análise e compreensão de um conjunto de dados utilizado em experimentos de AM, como os dados podem estar organizados e os tipos de valores que eles podem assumir. Serão apresentados ainda vários gráficos que facilitam a análise visual da distribuição dos valores em conjuntos de dados com uma ou mais variáveis.

Para isso, este capítulo está organizado da seguinte maneira. A Seção 2.1 descreve como os atributos de um conjunto de dados podem ser caracterizados pelo seu tipo e escala. Ao final, na Seção 2.2, são apresentadas várias medidas, assim como gráficos, que permitem descrever conjuntos de dados, tanto univariados quanto multivariados.

2.1 Caracterização de Dados

Conjuntos de dados são formados por objetos que podem representar um objeto físico, como uma cadeira, ou uma noção abstrata, como os sintomas apresentados por um paciente que se dirige a um hospital. Tais objetos também são comumente referidos como padrões, objetos, registros ou exemplos. Em geral, eles são representados por um vetor de características que os descreve, também denominadas atributos, campos ou variáveis. Cada objeto corresponde a uma ocorrência dos dados. Cada atributo está associado a uma propriedade do objeto.

Formalmente, os dados podem ser representados por uma matriz de objetos $\mathbf{X}_{n \times d}$, em que n é o número de objetos e d é o número de atributos de entrada de cada objeto. O valor de d define a dimensionalidade dos objetos ou do espaço de objetos. Cada elemento dessa matriz, x_i^j , contém o valor da j -ésima característica para o i -ésimo objeto. Os d atributos também podem ser vistos como um conjunto de eixos ortogonais e os objetos, como pontos no espaço de dimensão d , chamado espaço de objetos.

Para ilustrar os conceitos abordados neste capítulo com um exemplo prático, considere novamente o conjunto de dados provenientes de pacientes de um hospital, denominado `hospital`, ilustrado pela Tabela 2.1. Esse conjunto foi inicialmente apresentado no Capítulo 1.

No conjunto `hospital`, cada objeto corresponde a um paciente, sendo por isso formado pelos valores de atributos de entrada (também chamados atributos preditivos) referentes ao paciente. Esses atributos são: identificação, nome, idade, sexo, sintomas e resultados de exames clínicos. Exemplos de sintomas são presença e distribuição de manchas na pele, peso do paciente e temperatura do seu corpo.

Além desses atributos, a tabela apresenta um atributo alvo, também denominado atributo meta ou de saída, que representa o fenômeno de interesse sobre o qual se deseja fazer previsões. Em tarefas descritivas, os dados não apresentam tal atributo, e, muitas vezes, a definição desse tipo de atributo pode ser obtida como um de seus resultados. Já as tarefas preditivas se baseiam na presença desse atributo, como mencionado previamente no Capítulo 1. Na maioria dos casos, os dados apresentam apenas um atributo alvo. Entretanto, em pesquisas mais recentes, tem-se considerado dados com mais de um atributo alvo, e esse é o foco, por exemplo, da classificação multirrótulo, apresentada no Capítulo 18.

Quando um atributo alvo contém rótulos que identificam categorias às quais os objetos pertencem, ele é denominado classe e assume valores discretos $1, \dots, k$. Tem-se, nesse caso, um problema de classificação. Se as classes tiverem diferentes números de objetos, a classe com o maior número é denominada classe majoritária, e a com menos, minoritária. Se, por outro lado, o atributo alvo contém valores numéricos contínuos, tem-se um problema de regressão (Mitchell, 1997). Um caso especial de regressão é a previsão de valores em séries temporais, quando valores apresentam uma relação de periodicidade. Tanto em problemas de classificação quanto de regressão, os demais atributos são denominados atributos preditivos, por poderem ser utilizados para prever o valor do atributo alvo.

Na Tabela 2.1, para cada paciente são apresentados os valores para os atributos *Id.* (identificação do paciente), *Nome*, *Idade*, *Sexo*, *Peso*, *Manchas* (presença e distribuição de manchas no corpo), *Temp.* (temperatura do corpo), *# Int.* (número de internações), *Est.* (estado de origem) e *Diagnóstico*, que indica o diagnóstico do paciente e corresponde ao atributo alvo. Tabelas com esse formato também são denominadas tabelas atributo-valor.

Tabela 2.1 Conjunto de dados `hospital` com seus atributos

Id.	Nome	Idade	Sexo	Peso	Manchas	Temp.	# Int.	Est.	Diagnóstico
4201	João	28	M	79	Concentradas	38,0	2	SP	Doente
3217	Maria	18	F	67	Inexistentes	39,5	4	MG	Doente
4039	Luiz	49	M	92	Espalhadas	38,0	2	RS	Saudável
1920	José	18	M	43	Inexistentes	38,5	8	MG	Doente
4340	Cláudia	21	F	52	Uniformes	37,6	1	PE	Saudável
2301	Ana	22	F	72	Inexistentes	38,0	3	RJ	Doente
1322	Marta	19	F	87	Espalhadas	39,0	6	AM	Doente
3027	Paulo	34	M	67	Uniformes	38,4	2	GO	Saudável

Os valores que um atributo pode assumir podem ser definidos de diferentes formas. Neste livro, são definidos por dois aspectos: tipo e escala. O tipo de um atributo diz respeito ao grau de quantização nos dados, e a escala indica a significância relativa dos valores. Conhecer o tipo/escala dos atributos auxilia a identificar a forma adequada de preparar os dados e posteriormente de modelá-los.

As definições a seguir são utilizadas para classificar os valores que atributos podem assumir no que diz respeito a esses dois aspectos (Jain e Dubes, 1988; Barbara, 2000; Yang et al., 2005).

2.1.1 Tipo

O tipo define se o atributo representa quantidades, sendo então denominado quantitativo ou numérico, ou qualidades, quando é chamado de qualitativo, simbólico ou categórico, pois os valores podem ser associados a categorias. Exemplos de conjuntos de valores qualitativos são {pequeno, médio, grande} e {matemática, física, química}. Apesar de alguns atributos qualitativos poderem ter seus valores ordenados, operações aritméticas não podem ser aplicadas aos seus valores. Os atributos quantitativos são numéricos, como no conjunto de valores {23, 45, 12}. Os valores de um atributo quantitativo podem tanto ser ordenados quanto utilizados em operações aritméticas. Valores quantitativos podem ser ainda contínuos ou discretos.

Atributos contínuos podem assumir um número infinito de valores. Geralmente esses atributos são resultados de medidas. Frequentemente, esses valores são representados por números reais. O número infinito não ocorre quando computadores digitais são utilizados, pois a precisão para valores reais ou ponto flutuante adotada pelo computador utilizado define um limite. Exemplos de atributos contínuos são atributos que representam peso, tamanho e distância.

Atributos discretos contêm um número finito ou infinito contável de valores. Um caso especial dos atributos discretos são os atributos binários (ou booleanos), que apresentam apenas dois valores, como 0/1, sim/não, ausência/presença e verdadeiro/falso.

Por exemplo, os atributos presentes no conjunto de dados da Tabela 2.1 têm a classificação de tipo apresentada na Tabela 2.2.

Tabela 2.2 *Tipo dos atributos do conjunto hospital*

Atributo	Classificação
Id.	Qualitativo
Nome	Qualitativo
Idade	Quantitativo discreto
Sexo	Qualitativo
Peso	Quantitativo contínuo
Manchas	Qualitativo
Temp.	Quantitativo contínuo
#Int.	Quantitativo discreto
Est.	Qualitativo
Diagnóstico	Qualitativo

É importante observar que uma medida quantitativa possui, além do valor numérico, uma unidade, por exemplo, metro. No processo de extração de conhecimento, se um atributo altura tem valor 100, o valor em si não diz se a altura é medida em centímetros, metros ou jardas, e essa informação pode ser importante na avaliação do conhecimento adquirido.

Os atributos quantitativos ou numéricos podem assumir valores binários, inteiros ou reais. Por outro lado, atributos qualitativos são, geralmente, representados por um número finito de símbolos ou nomes. Entretanto, algumas vezes, atributos categóricos são representados por números, mas nesse caso não faz sentido a utilização de operadores aritméticos sobre os seus valores. Por exemplo, qual seria o sentido de calcular a média dos valores de um atributo categórico representando o número de identificação de um paciente?

2.1.2 Escala

A escala define as operações que podem ser realizadas sobre os valores do atributo. Em relação à escala, os atributos podem ser classificados como nominais, ordinais, intervalares e racionais. Os dois primeiros são do tipo qualitativo e os dois últimos são quantitativos. Essas quatro escalas são definidas em detalhes a seguir.

Na escala nominal, os valores são apenas nomes diferentes, carregando a menor quantidade de informação possível. Não existe uma relação de ordem entre seus valores. Assim, as operações mais utilizadas para manipulação de seus valores são as de igualdade e desigualdade de valores. Por exemplo, se o atributo representa continentes do planeta, é possível apenas ver se dois valores são iguais ou diferentes (a menos que se queira ordenar os continentes por ordem alfabética, mas nesse caso o atributo seria do tipo ordinal). São exemplos de atributos com escala nominal: nome do paciente, RG, CPF, número da conta no banco, CEP, cores (com as categorias verde, amarelo, branco etc.), sexo (com as categorias feminino e masculino).

Os valores em uma escala ordinal refletem também uma ordem das categorias representadas. Dessa forma, além dos operadores anteriores, operadores como $<$, $>$, \leq , \geq podem ser utilizados. Por exemplo, quando um atributo possui como valores pequeno, médio e grande, além de os valores serem categóricos, é possível definir se um valor é igual, maior ou menor que outro. Exemplos de atributos com escala ordinal são: hierarquia militar e avaliações qualitativas de temperatura, como frio, morno e quente.

Na escala intervalar, os atributos intervalares são representados por números que variam dentro de um intervalo. Assim, é possível definir tanto a ordem quanto a diferença em magnitude entre dois valores. A diferença em magnitude indica a distância que separa dois valores no intervalo de possíveis valores. O valor zero não tem o mesmo significado que o zero utilizado em operações aritméticas. Por exemplo, seja a escala de temperatura dada em graus Celsius. Se o serviço de previsão do tempo em dias de verão para uma dada cidade informa que a temperatura vai variar entre 26 e 34 graus e em dias de inverno, para a mesma cidade, informa que vai variar entre 13 e 21 graus, tem-se que a temperatura dessa cidade apresenta uma variação de 8 graus entre a mínima e a máxima, não importa se a estação do ano é verão ou inverno. Entretanto, 90 graus Celsius é diferente de 90 graus Fahrenheit, apesar de ambos os valores se referirem ao atributo temperatura, do mesmo modo que não é possível afirmar que no inverno a cidade é duas vezes menos quente que no verão, pois não faz sentido, para esse atributo, utilizar como informação a razão entre dois valores. Isso ocorre porque o ponto em que esse atributo assume o valor 0, chamado de ponto zero ou origem da escala, é definido de forma arbitrária. Isso é uma característica dos atributos intervalares. Esse problema seria eliminado se fosse utilizada a escala de temperatura Kelvin, cujo valor do ponto zero é o ponto zero verdadeiro

(Pyle, 1999). Outros exemplos são a duração de um evento em minutos e datas em um calendário.

Atributos com escala racional são os que carregam mais informações. Os números têm um significado absoluto, ou seja, existe um zero absoluto junto com uma unidade de medida, de forma que a razão tenha significado. Por exemplo, considerando o número de vezes que uma pessoa foi ao hospital, o ponto zero está associado a não ter ocorrido nenhuma visita. Se um paciente teve duas internações e outro teve oito internações, é correto dizer que o segundo paciente esteve internado quatro vezes mais que o primeiro. Ou seja, faz sentido, para esses atributos, utilizar a razão entre dois valores. Outros exemplos de atributos com escala de razão são tamanho, distância e valores financeiros, como salário e saldo em conta-corrente.

Baseado na descrição anterior, os atributos do conjunto de dados da Tabela 2.1 podem ser classificados como indicado na Tabela 2.3.

Tabela 2.3 Escala dos atributos do conjunto *hospital*

Atributo	Classificação
Id.	Nominal
Nome	Nominal
Idade	Racional
Sexo	Nominal
Peso	Racional
Manchas	Nominal
Temp.	Intervalar
#Int.	Racional
Est.	Nominal
Diagnóstico	Nominal

Recentemente, tipos mais complexos de atributos foram adotados, como os atributos hierárquicos, cujos valores representam uma hierarquia de valores, e atributos heterogêneos, cujos valores podem assumir mais de um tipo.

2.2 Exploração de Dados

Uma grande quantidade de informações úteis pode ser extraída de um conjunto de dados por meio de sua análise ou exploração. Informações obtidas na exploração podem ajudar, por exemplo, na seleção da técnica mais apropriada para pré-processamento dos dados e para aprendizado. Uma das formas mais simples de explorar um conjunto de dados é a extração de medidas de uma área da estatística denominada estatística descritiva. A estatística descritiva resume de forma quantitativa as principais características de um conjunto de dados. Muitas dessas medidas são calculadas rapidamente. Por exemplo, no conjunto de dados de pacientes, duas medidas estatísticas podem ser facilmente calculadas: a idade média dos pacientes e a porcentagem de pacientes do sexo masculino.

As medidas da estatística descritiva assumem que os dados são gerados por um processo estatístico. Como o processo pode ser caracterizado por vários parâmetros, as medidas podem ser vistas como estimativas dos parâmetros estatísticos da distribuição que gerou

os dados. Por exemplo, os dados podem ter sido gerados por uma distribuição normal com média 0 e variância 1. Essas medidas permitem capturar informações como:

- Frequência;
- Localização ou tendência central (por exemplo, a média);
- Dispersão ou espalhamento (por exemplo, o desvio padrão);
- Distribuição ou formato.

A medida de frequência é a mais simples. Ela mede a proporção de vezes que um atributo assume um dado valor em um determinado conjunto de dados. Ela pode ser aplicada a valores tanto numéricos quanto simbólicos, e é muito utilizada nesses últimos. Um exemplo de seu uso seria: *Em um conjunto de dados médicos, 40% dos pacientes têm febre.*

As outras medidas diferem para os casos em que os dados apresentam apenas um atributo (dados univariados) ou mais de um atributo (dados multivariados). Elas são geralmente aplicadas a dados numéricos. Apesar de a maioria dos conjuntos de dados utilizados em AM apresentar mais de um atributo, análises realizadas em cada atributo podem oferecer informações valiosas sobre os dados.

A seguir, serão apresentadas as principais medidas de localização, dispersão e distribuição para dados uni e multivariados.

2.2.1 Dados Univariados

Para os dados univariados, supor que um objeto x_i possua apenas um atributo. Um conjunto com n objetos pode ser então representado por $\mathbf{x}^j = \{x_1, x_2, \dots, x_n\}$, em que cada x_i é representado por um valor. É importante observar que o termo conjunto não tem o mesmo significado do termo utilizado em teoria dos conjuntos. Em um conjunto de dados, o mesmo valor pode aparecer mais de uma vez em um atributo.

Medidas de Localidade

As medidas de localidade definem pontos de referência nos dados e variam para dados numéricos e simbólicos. Para dados simbólicos, utiliza-se geralmente a moda, que é o valor encontrado com maior frequência para um atributo. Por exemplo, a moda do atributo manchas na Tabela 2.1 é o valor *Inexistentes*, que aparece em três dos oito pacientes.

Para atributos numéricos, medidas muito utilizadas são média, mediana e percentil. Supor o conjunto de n valores numéricos: $\mathbf{x}^j = \{x_1, x_2, \dots, x_n\}$. O valor médio desse conjunto pode ser facilmente calculado pela Equação 2.1.

$$\bar{x}^j = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.1)$$

Um problema da média é a sua sensibilidade à presença de *outliers*, que são valores muito diferentes dos demais valores observados para o mesmo atributo (ver detalhes no Capítulo 3). Média é um bom indicador do meio de um conjunto de valores apenas se os valores estão distribuídos simetricamente. Um valor muito mais alto ou muito mais

baixo que os demais valores do conjunto pode gerar um valor distorcido para a média, que poderia mudar radicalmente se for retirado o *outlier*. Esse problema é minimizado com o uso da mediana, que é menos sensível a *outliers*. Para usar a mediana, o primeiro passo é ordenar de forma crescente o conjunto de valores. Ordenados os valores, a Equação 2.2 pode ser utilizada para o cálculo da mediana.

$$\text{mediana}(\mathbf{x}^j) = \begin{cases} \frac{1}{2}(x_r + x_{r+1}) & \text{se } n \text{ for par } (n = 2r) \\ x_{r+1} & \text{se } n \text{ for ímpar } (n = 2r + 1) \end{cases} \quad (2.2)$$

Assim, se o número de valores, n , é ímpar, a mediana é igual ao valor do meio do conjunto ordenado. Caso contrário, se for par, é dada pela média dos dois valores do meio. Por exemplo, seja o conjunto de valores $\{17, 4, 8, 21, 4\}$. A ordenação desse conjunto gera a sequência de valores $(4, 4, 8, 17, 21)$. Observe que valores repetidos são mantidos na sequência. Como o número de valores é ímpar, 5, a mediana é dada pelo terceiro valor, assim, a mediana = 8. Se o conjunto de valores fosse formado pelos elementos, $\{17, 4, 8, 21, 4, 15, 13, 9\}$, como o número de elementos, 8, é par, a mediana é dada pela média entre o quarto e quinto valores da sequência ordenada $(4, 4, 8, 9, 13, 15, 17, 21)$. Nesse caso, mediana = $(9 + 13)/2 = 11$. O uso da mediana torna mais fácil observar se a distribuição é oblíqua (assimétrica) ou se existem *outliers*.

Existem ainda variações da média e da mediana, como, por exemplo, a média truncada, que minimiza os problemas da média por meio do descarte dos exemplos nos extremos da sequência ordenada dos valores. Para isso, é necessário definir a porcentagem dos exemplos a serem eliminados em cada extremidade.

Outras medidas muito utilizadas são os quartis e os percentis. Assim como a mediana, essas medidas são utilizadas após os valores serem ordenados. Enquanto a mediana divide os dados ao meio, essas outras medidas utilizam pontos de divisão diferentes. Os quartis dividem os valores ordenados em quartos. Assim, o 1º quartil de uma sequência, $Q1$, é o valor que tem 25% dos demais valores abaixo dele. Esse também é o valor do 25º percentil, $P25$. O 2º quartil é a mediana, que é igual ao 50º percentil, $P50$.

Seja p um valor entre 0 e 100. O p º percentil, P_p , é um valor x_i do conjunto de valores tal que $p\%$ dos valores observados são menores que x_i . Assim, o 40º percentil, $P40$, de um conjunto de valores é o valor para o qual 40% dos demais valores são menores ou iguais a ele. Para calcular o p º percentil, basta seguir os passos do Algoritmo 2.1.

Uma forma simples de visualizar a distribuição dos dados é utilizar técnicas de visualização, como, por exemplo, *boxplots*, histogramas e *scatter plots*, que serão vistas mais adiante neste capítulo. Para algumas das medidas e gráficos a serem vistos, será utilizado o conjunto de dados *iris* (Fisher, 1936). O conjunto de dados *iris* é um dos mais populares em AM. Ele contém 150 objetos, cada objeto formado por quatro atributos de entrada (tamanho da sépala, largura da sépala, tamanho da pétala e largura da pétala), que assumem valores contínuos, e um atributo alvo com três valores nominais ou classes, que representam as espécies íris setosa, íris versicolor e íris virgínica.

A Figura 2.1 mostra duas variações do gráfico *boxplot* para o atributo largura da sépala, do conjunto de dados *iris*. O *boxplot*, também chamado diagrama de Box e Whisker, apresenta um resumo dos valores para o 1º, 2º (mediana) e 3º quartis, além dos limites inferior e superior.

Do lado esquerdo da figura é apresentado o gráfico *boxplot* original. Nele, a linha horizontal mais baixa e a linha horizontal mais alta indicam, respectivamente, os valores

Algoritmo 2.1 Algoritmo para cálculo do percentil

Entrada: Um conjunto de n valores e o percentil p (valor real entre 0,0 (equivale a 0%) e 1,0 (equivale a 100%)) a ser retornado

Saída: Valor do percentil

- 1 Ordenar os n valores em ordem crescente
- 2 Calcular o produto np
- 3 se np não for um número inteiro então
 - 4 Arredondar para o próximo inteiro
 - 5 Retornar o valor dessa posição na sequência
- 6 fim
- 7 senão
 - 8 Considerar $np = k$
 - 9 Retornar a média entre os valores nas posições k e $k + 1$
- 10 fim

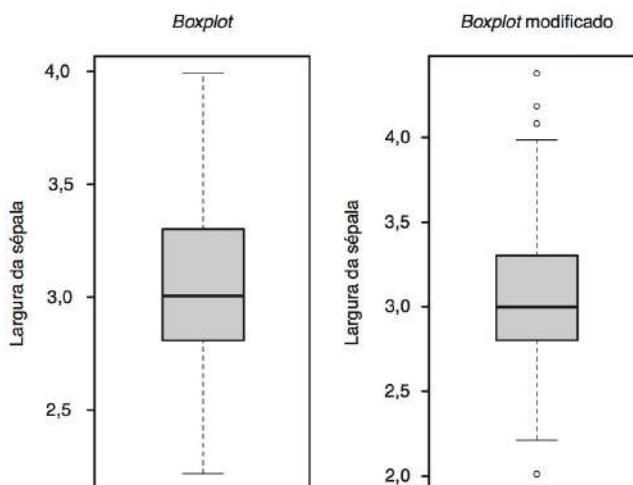


Figura 2.1 Boxplots para o atributo largura da sépala do conjunto de dados iris.

mínimo e máximo presentes nos dados. Os lados inferior e superior do retângulo representam o 1º quartil e o 3º quartil, respectivamente. A linha no interior do retângulo é o 2º quartil, ou mediana. O limite superior (inferior) da linha tracejada vai até o maior (menor) valor do conjunto de dados.

O gráfico da direita ilustra uma variação do gráfico *boxplot*, conhecido como *boxplot* modificado. Nesse gráfico, o limite superior (inferior) da linha tracejada vai até o maior (menor) valor apenas se esse valor não for muito distante do 3º (1º) quartil (no máximo $1,5 \times$ intervalo entre quartis). Os valores acima do limite superior e abaixo do limite inferior são considerados *outliers*. Nesse gráfico, 4 valores *outliers* são representados por círculos, 3 maiores que $3^{\text{o}} \text{ quartil} + 1,5 \times (3^{\text{o}} \text{ quartil} - 1^{\text{o}} \text{ quartil})$ e 1 menor que $1^{\text{o}} \text{ quartil} - 1,5 \times (3^{\text{o}} \text{ quartil} - 1^{\text{o}} \text{ quartil})$.

Medidas de Espalhamento

As medidas de espalhamento medem a dispersão ou espalhamento de um conjunto de valores. Assim, elas permitem observar se os valores estão amplamente espalhados ou relativamente concentrados em torno de um valor, por exemplo, a média. As medidas de espalhamento mais comuns são:

- Intervalo;
- Variância;
- Desvio padrão.

O intervalo é a medida mais simples e mostra espalhamento máximo entre os valores de um conjunto. Assim, sejam $\mathbf{x}^j = \{x_1, x_2, \dots, x_n\}$ os valores do atributo para n objetos. O intervalo desse conjunto é medido pela Equação 2.3.

$$\text{intervalo}(\mathbf{x}^j) = \max_{i=1, \dots, n} (x_i) - \min_{i=1, \dots, n} (x_i) \quad (2.3)$$

Se a maioria dos valores for próxima de um ponto, com um pequeno número de valores extremos, o intervalo não será uma boa medida do espalhamento dos valores.

A medida mais utilizada para avaliar o espalhamento de valores é a variância, que é dada pela Equação 2.4.

$$\text{variância}(\mathbf{x}^j) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x}^j)^2 \quad (2.4)$$

Nessa equação, \bar{x}^j é a média dos valores de x . O uso do denominador $n-1$, chamado correção de Bessel, dá uma melhor estimativa da variância verdadeira do que o uso de n . Outra medida de espalhamento, o desvio padrão, é dada pela raiz quadrada da variância.

Assim como a média, o valor da variância pode ser distorcido pela presença de *outliers*, pois a variância calcula a diferença entre cada valor e a média. Outras estimativas mais robustas de espalhamento frequentemente utilizadas são:

- Desvio médio absoluto (AAD, do inglês *absolute average deviation*), ilustrado pela Equação 2.5.
- Desvio mediano absoluto (MAD, do inglês *median absolute deviation*), ilustrado pela Equação 2.6.
- Intervalo interquartil (IQR, do inglês *interquartile range*), ilustrado pela Equação 2.7.

$$AAD(\mathbf{x}^j) = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}^j| \quad (2.5)$$

$$MAD(\mathbf{x}^j) = \text{mediana} (\{|x_1 - \bar{x}^j|, \dots, |x_n - \bar{x}^j|\}) \quad (2.6)$$

$$IQR(\mathbf{x}^j) = P_{75\%} - P_{25\%} \quad (2.7)$$

Medidas de Distribuição

As medidas que são definidas em torno da média de um conjunto de valores, como as medidas média e desvio padrão, são em sua maioria instanciações de uma medida denominada momento, que é definida pela Equação 2.8.

$$\text{momento}_k(\mathbf{x}^j) = \frac{\sum_{i=1}^n (x_i - \bar{x}^j)^k}{(n - 1)} \quad (2.8)$$

Para cada valor do parâmetro k , uma medida diferente de momento é definida. Assim:

- quando $k = 1$, tem-se o valor 0, que é o primeiro momento em torno da origem ou primeiro momento central;
- quando $k = 2$, tem-se a variância, que é o segundo momento central;
- quando $k = 3$, tem-se a obliquidade, que é o terceiro momento central;
- quando $k = 4$, tem-se a curtose, que é o quarto momento central.

Conforme já visto, os dois primeiros momentos, o valor 0 e o desvio padrão, são medidas de localidade e espalhamento, respectivamente. O terceiro e quarto momentos, obliquidade e curtose, são medidas de distribuição, por mostrarem como os valores estão distribuídos.

Outra forma simples de visualizar a distribuição dos dados é representá-los em um histograma. Um histograma divide os valores de um conjunto de dados em cestas. Para cada cesta, é desenhada uma barra cuja altura é proporcional ao número de elementos na cesta. Para valores categóricos, cada valor é uma cesta. Para valores numéricos, os valores são divididos em intervalos contíguos, geralmente, do mesmo tamanho, e cada intervalo é uma cesta. O formato do histograma depende do número de cestas utilizadas. Na Figura 2.2, é mostrado um histograma com a distribuição de valores para cada atributo do conjunto de dados *iris*.

O terceiro momento, obliquidade (em inglês *skewness*), mede a simetria da distribuição dos dados em torno da média. Em uma distribuição simétrica, se os valores forem distribuídos em intervalos do mesmo tamanho, um histograma com a quantidade de valores em cada intervalo tem a mesma aparência à direita e à esquerda do ponto central. A Equação 2.9 ilustra o cálculo da obliquidade, que utiliza $k = 3$ e divide o valor calculado pelo desvio padrão elevado ao cubo, s^3 , para tornar a medida independente de escala.

$$\text{obliquidade}(\mathbf{x}^j) = \frac{\text{momento}_3(\mathbf{x}^j)}{s^3} = \frac{\sum_{i=1}^n (x_i - \bar{x}^j)^3}{(n - 1)s^3} \quad (2.9)$$

A distribuição dos valores em um conjunto de dados está associada ao valor da obliquidade da seguinte forma:

- obliquidade = 0 (simétrica): a distribuição é aproximadamente simétrica (ocorre para uma distribuição normal);
- obliquidade > 0 (positiva): a distribuição concentra-se mais no lado esquerdo;

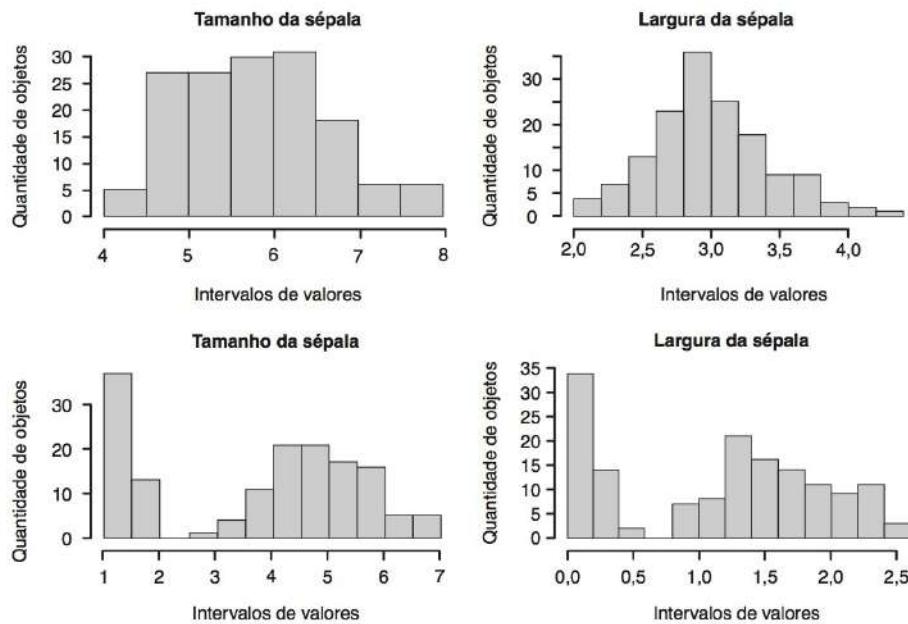


Figura 2.2 Histograma para a distribuição de valores dos atributos de entrada do conjunto *iris*.

- obliquidade < 0 (negativa): a distribuição concentra-se mais no lado direito.

A Figura 2.3 ilustra esses três tipos de distribuição medidos pela obliquidade.

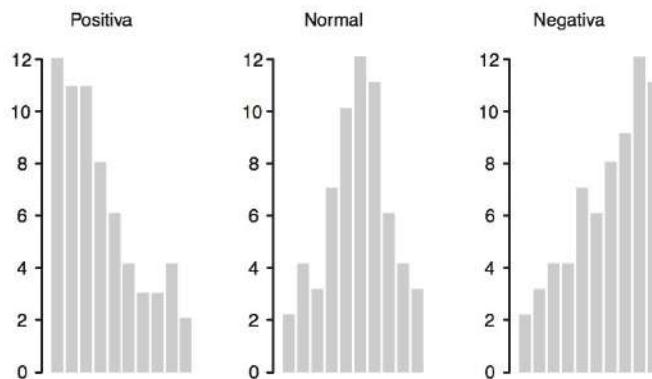


Figura 2.3 Distribuição dos valores de obliquidade.

O quarto momento calcula a curtose (em inglês *kurtosis*), que é uma medida de dispersão que captura o achatamento da função de distribuição. A medida de curtose, calculada pela Equação 2.10, verifica se os dados apresentam um pico ou são achatados em relação a uma distribuição normal. Como na Equação 2.9, para tornar a medida independente de escala, o quarto momento é dividido pelo desvio padrão elevado à quarta potência, s^4 .

$$\text{curtose}(\mathbf{x}^j) = \frac{\text{momento}_4(\mathbf{x}^j)}{s^4} = \frac{\sum_{i=1}^n (x_i - \bar{x}^j)^4}{(n-1)s^4} \quad (2.10)$$

Como para uma distribuição normal com média 0 e variância 1 o valor da curtose é igual a 3, é feita uma correção na sua fórmula para que a distribuição normal padrão tenha curtose igual a 0. Para isso, geralmente é utilizada a Equação 2.11.

$$\text{curtose}(\mathbf{x}^j) = \frac{\text{momento}_4(\mathbf{x}^j)}{\sigma^4} - 3 = \frac{\sum_{i=1}^n (x_i - \bar{x}^j)^4}{(n-1)\sigma^4} - 3 \quad (2.11)$$

Assim como na medida de obliquidade, a seguinte relação, ilustrada pela Figura 2.4, é observada entre o valor da curtose e a distribuição dos valores em um conjunto de dados:

- curtose = 0 (normal): o histograma de distribuição dos dados apresenta o mesmo achatamento que uma distribuição normal;
- curtose > 0 (positiva): o histograma de distribuição dos dados apresenta uma distribuição mais alta e concentrada que a distribuição normal;
- curtose < 0 (negativa): o histograma de distribuição dos dados apresenta uma distribuição mais achatada que a distribuição normal.

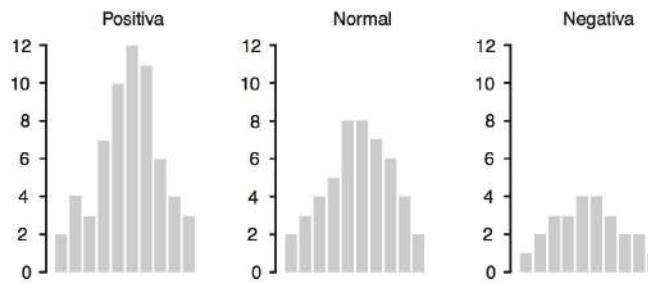


Figura 2.4 Distribuição dos valores de curtose.

Um outro gráfico muito utilizado para ilustrar a distribuição de um conjunto de valores é o gráfico de pizza. Nesse gráfico, cada valor ocupa uma fatia cuja área é proporcional ao número de vezes em que o valor aparece no conjunto de dados. Esse gráfico é indicado para valores qualitativos. A Figura 2.5 mostra a distribuição dos valores do atributo *Manchas* do conjunto de dados *hospital* utilizando um gráfico de pizza. Se um atributo quantitativo for utilizado, os valores podem, assim como no histograma, ser agrupados em cestas.

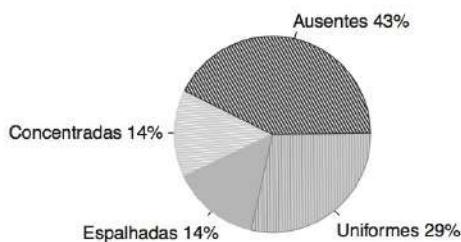


Figura 2.5 Gráfico de pizza para a distribuição de valores do atributo *Manchas*.

2.2.2 Dados Multivariados

Dados multivariados são aqueles que possuem mais de um atributo de entrada. O conjunto de dados *iris* é um conjunto multivariado.

Nesses casos, as medidas de localidade podem ser obtidas calculando a medida de localidade de cada atributo separadamente. Por exemplo, a média para um conjunto de objetos com d atributos pode ser calculada pela Equação 2.12.

$$\bar{\mathbf{x}} = (\bar{x}^1, \dots, \bar{x}^d) \quad (2.12)$$

As medidas de espalhamento podem ser calculadas para cada atributo independentemente dos demais utilizando qualquer medida de espalhamento.

Conforme visto na seção anterior para o conjunto de dados `iris`, cada atributo pode ter suas características graficamente exploradas quando os atributos são tomados individualmente.

Dados multivariados permitem ainda análises da relação entre dois ou mais atributos. Por exemplo, para atributos quantitativos, o espalhamento de um conjunto de dados é mais bem capturado por uma matriz de covariância, em que cada elemento é a covariância entre dois atributos. Cada elemento cov_{ij} de uma matriz de covariância \mathbf{Cov} mede a covariância entre os atributos \mathbf{x}^i e \mathbf{x}^j , que é dada pela Equação 2.13.

$$\text{covariância}(\mathbf{x}^i, \mathbf{x}^j) = \frac{1}{n-1} \sum_{k=1}^n (x_k^i - \bar{x}^i)(x_k^j - \bar{x}^j) \quad (2.13)$$

Nessa equação, \bar{x}^i representa o valor médio do i -ésimo atributo e x_k^i , o valor do i -ésimo atributo para o k -ésimo objeto. É importante observar que $\text{covariância}(\mathbf{x}^i, \mathbf{x}^i) = \text{variância}(\mathbf{x}^i)$. Assim, a matriz de covariância tem em sua diagonal as variâncias dos atributos.

A covariância entre dois atributos mede o grau com que os atributos variam juntos. Seu valor depende da magnitude dos atributos. Um valor próximo de 0 indica que os atributos não têm um relacionamento linear. Um valor positivo indica que os atributos são diretamente relacionados. Quando o valor de um dos atributos aumenta, o do outro também aumenta. O contrário ocorre se a covariância for negativa.

A medida de covariância é afetada pela dimensão dos atributos avaliados. Assim, dois atributos com dimensão elevada podem apresentar um valor de covariância maior que dois atributos mais semelhantes entre si, mas de menor dimensão. Por isso, não é possível avaliar o relacionamento entre dois atributos observando apenas a covariância entre eles. A medida de correlação elimina esse problema retirando a influência da dimensão. Como resultado, ela apresenta uma indicação mais clara da força da relação linear entre dois atributos. Por isso, a correlação é mais utilizada para explorar dados multivariados que a covariância. A matriz de correlação apresenta a correlação entre cada possível par de atributos de um conjunto de dados. Cada elemento da matriz de correlação tem seu valor definido pela Equação 2.14.

$$\text{correlação}(\mathbf{x}^i, \mathbf{x}^j) = \frac{\text{covariância}(\mathbf{x}^i, \mathbf{x}^j)}{s_i s_j} \quad (2.14)$$

Nessa equação, \mathbf{x}^i é o i -ésimo atributo e s_i é o desvio padrão dos valores desse atributo. Vale notar que $\text{correlação}(\mathbf{x}^i, \mathbf{x}^i) = 1$. Dessa forma, os elementos da diagonal têm valor 1. Os demais elementos têm um valor entre -1 (correlação negativa máxima) e $+1$ (correlação positiva máxima).

Assim como na medida de covariância, quando dois atributos apresentam uma correlação positiva, o aumento do valor de um deles é geralmente acompanhado por um aumento no valor do outro. Da mesma forma, quando dois atributos têm uma correlação negativa, a redução no valor de um deles é geralmente acompanhada da redução do valor do outro.

A análise dos dados multivariados pode ser facilitada pelo uso de recursos de visualização. Assim como histogramas, gráficos de pizza e *boxplots* são utilizados para visualizar dados univariados, outros diagramas têm sido adotados para visualizar dados multivariados, particularmente a relação entre os diferentes atributos. Entre esses diagramas, um dos mais utilizados é o *scatter plot*, que ilustra a correlação linear entre dois atributos.

Em um *scatter plot*, a cada objeto, considerando apenas dois de seus atributos, é associado uma posição ou ponto em um plano bidimensional. Os valores dos atributos, que podem ser números inteiros ou reais, definem as coordenadas desse ponto. Na Figura 2.6, um *scatter plot* mostra a correlação entre dois atributos do conjunto de dados *iris*: tamanho da sépala e largura da sépala.

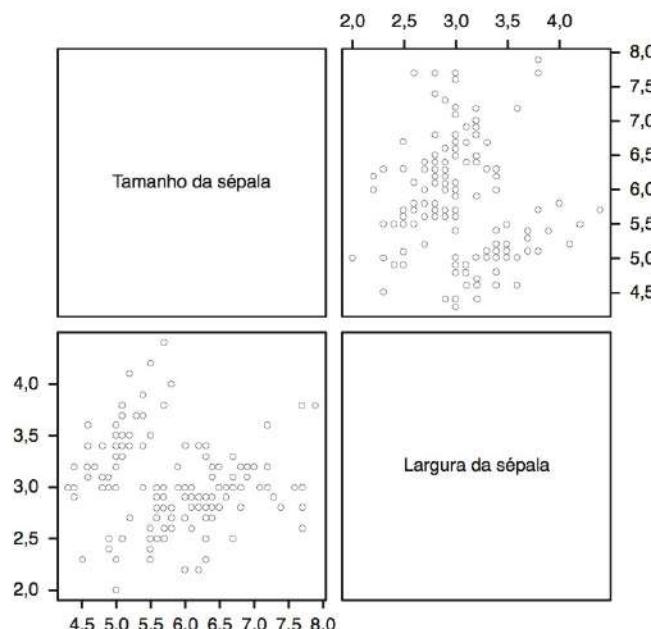


Figura 2.6 Matriz de scatter plot para dois atributos do conjunto de dados *iris*.

No *scatter plot* do canto superior direito dessa figura, cada ponto representa a *Largura da sépala* (eixo horizontal) e o *Tamanho da sépala* (eixo vertical) de um dos 150 objetos do conjunto de dados *iris*. No *scatter plot* do canto inferior esquerdo ocorre o contrário, cada ponto representa o *Tamanho da sépala* (eixo horizontal) e a *Largura da sépala* (eixo vertical) de um objeto.

Gráficos de *scatter plot* para diferentes combinações de atributos podem ser exibidos em matrizes de *scatter plot*. Embora *scatter plots* com duas dimensões sejam mais comuns, eles também podem ser utilizados para visualizar a relação entre três atributos, definindo *scatter plots* tridimensionais. Os atributos adicionais podem ser exibidos utilizando tamanho, formato e cor nos marcadores que representam os objetos.

Quando as classes dos objetos são disponibilizadas, o *scatter plot* pode ser usado para investigar o grau com que dois atributos separam as classes, se for possível separar a maioria dos objetos de uma das classes com uma reta.

Outras técnicas de visualização para dados multivariados bastante utilizadas são os *bagplots*, as faces de Chernoff, os *star plots* e os *heatmaps* (Wu et al., 1998), que são apresentados a seguir.

O *bagplot* é uma generalização bivariada do *boxplot* que permite apresentar, em uma mesma figura, o *boxplot* de dois atributos ou variáveis (Rousseeuw et al., 1999). Cada eixo do gráfico pode ser considerado um *boxplot* associado a um dos dois atributos.

A Figura 2.7 ilustra um gráfico de *bagplot* para dois atributos do conjunto de dados *iris*: tamanho da sépala e largura da sépala. O gráfico apresenta três regiões convexas. A menor delas tem apenas um objeto, representado por um asterisco, cujas coordenadas são definidas pela mediana de seus dois atributos. A segunda região, denominada *bag*, possui 50% dos objetos, cujas coordenadas são definidas pelos valores entre o primeiro e terceiro quartis para cada um dos atributos. A maior região, chamada *loop*, é a *bag* expandida três vezes o intervalo entre quartis nas duas dimensões, 1,5 vez em cada sentido. Para cada eixo, o valor máximo e o valor mínimo são definidos, respectivamente, pelos limites superior e inferior do atributo correspondente. Objetos fora da maior região são considerados *outliers*. Cada dimensão vista isoladamente representa o *boxplot* para o atributo associado.

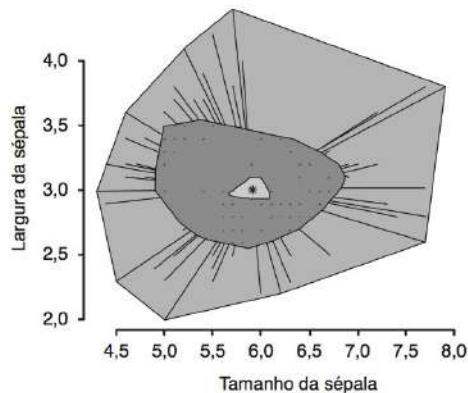


Figura 2.7 Diagramas de *bagplot* para dois atributos do conjunto de dados *iris*.

No diagrama de Chernoff, cada atributo é representado por uma ou mais características de uma face, como altura e largura da cabeça, da boca, do cabelo, do nariz, das orelhas, se a face tem um sorriso e o estilo do cabelo. O número de atributos representados é limitado pelo número de características presentes na implementação do algoritmo que desenha as faces. Se o conjunto de dados tem menos atributos que o número de possíveis características, um mesmo atributo pode ser representado por mais de uma característica da face. O diagrama de Chernoff para 15 exemplos do conjunto de dados *iris* pode ser visto na Figura 2.8. Nessa figura, o tamanho da sépala, por exemplo, é representado pelas características altura da face, largura da boca, altura do cabelo e largura do nariz. Os outros atributos do conjunto de dados *iris* são representados nas figuras por outras características da face.

Assim como o diagrama de Chernoff, o *star plot* desenha uma figura geométrica para cada objeto, normalmente um polígono. Cada linha do polígono corresponde a um dos atributos, e o tamanho da linha é proporcional ao valor do atributo. Um segmento de reta liga o centro do polígono a cada um de seus vértices. Um conjunto de *star plots* para os objetos do conjunto *iris* é ilustrado pela Figura 2.9. Quanto mais atributos são considerados, mais o polígono se assemelha a uma estrela. Quando dois ou mais

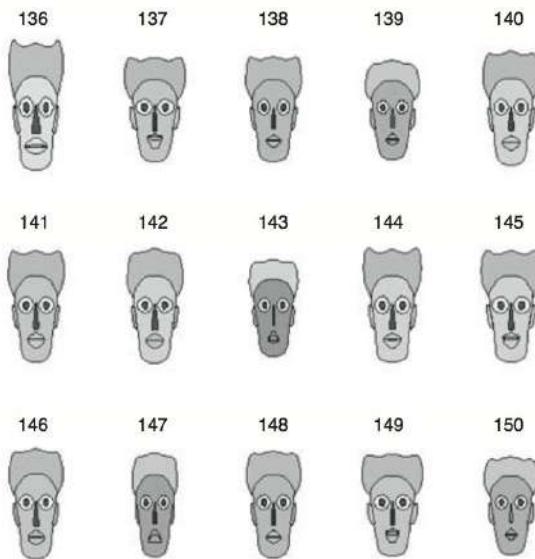


Figura 2.8 Diagramas de Chernoff para 15 objetos do conjunto de dados *iris*.

atributos de um objeto têm valores semelhantes, eles apresentam coordenadas similares no diagrama, deformando o formato de estrela.

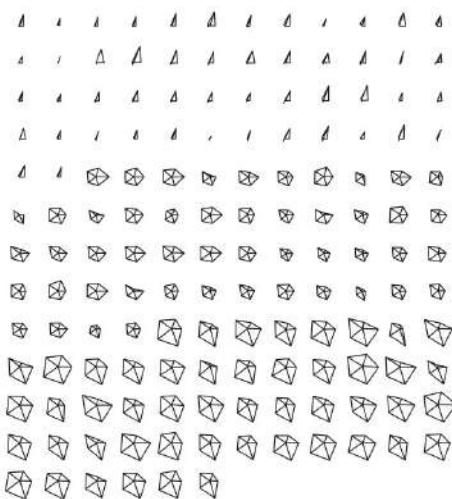


Figura 2.9 Star plots para os 150 objetos do conjunto de dados *iris*.

Um *heatmap* representa a relação entre os exemplos e as classes, associando um eixo para cada um. Para cada eixo ele constrói uma imagem gráfica referente ao agrupamento hierárquico (dendrograma) dos elementos do eixo, utilizando como entrada os elementos do outro eixo, como pode ser visto na Figura 2.10. Para facilitar a visualização, em vez do nome completo de cada atributo, foram utilizadas apenas suas iniciais. Assim, TS significa *Tamanho da Sépala*, por exemplo.

Nessa imagem, cada linha da imagem representa um objeto do conjunto de dados *iris* e cada coluna, uma das classes desse conjunto. Elementos semelhantes são agrupados

juntos, apresentando cores ou tonalidades semelhantes. No topo da imagem é apresentado um dendrograma das quatro classes do conjunto *iris* e no lado esquerdo um dendrograma para os 150 objetos. Como os elementos em cada eixo são agrupados na imagem, de acordo com sua semelhança, *heatmaps* permitem ver tendências nos valores. Em bioinformática, os *heatmaps* são muito utilizados para análise de dados de expressão gênica. Nesse caso, são agrupados os genes, de acordo com a similaridade de seus níveis de expressão em um dos eixos, e, no outro eixo, os níveis de expressão, de acordo com os genes que possuem valores similares para eles.

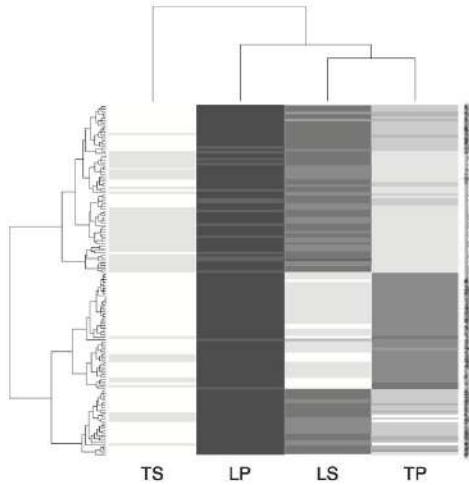


Figura 2.10 Heatmap para atributos de entrada do conjunto de dados *iris*.

2.3 Considerações Finais

Antes de aplicar algoritmos de AM a um conjunto de dados, é importante que os dados sejam analisados. Essa análise, que pode ser realizada por técnicas estatísticas e de visualização, permite uma melhor compreensão da distribuição dos dados e pode dar suporte à escolha de formas de abordar o problema.

Neste capítulo, foram apresentados conceitos considerados importantes para analisar os principais aspectos de um conjunto de dados. Após uma caracterização dos tipos de dados encontrados na maioria das aplicações de AM, foram apresentadas medidas estatísticas frequentemente utilizadas para exploração de dados univariados e multivariados. Por fim, diversas técnicas de visualização foram descritas para dados com uma ou mais varáveis: *boxplots*, histogramas, gráficos de pizza, *scatter plots*, *bagplots*, faces de Chernoff, *star plots* e *heatmaps*. Essas são apenas algumas das várias técnicas de visualização existentes.

Capítulo 3

Pré-processamento de Dados

Apesar de algoritmos de AM serem frequentemente adotados para extrair conhecimento de conjuntos de dados, seu desempenho é geralmente afetado pelo estado dos dados. Conjuntos de dados podem apresentar diferentes características, dimensões ou formatos. Por exemplo, conforme visto no capítulo anterior, os valores dos atributos de um conjunto de dados podem ser numéricos ou simbólicos. Podem ainda estar limpos ou conter ruídos e imperfeições, com valores incorretos, inconsistentes, duplicados ou ausentes; os atributos podem ser independentes ou relacionados; os conjuntos de dados podem apresentar poucos ou muitos objetos, que por sua vez podem ter um número pequeno ou elevado de atributos.

Técnicas de pré-processamento de dados são frequentemente utilizadas para melhorar a qualidade dos dados por meio da eliminação ou minimização dos problemas citados. Essa melhora pode facilitar o uso de técnicas de AM, levar à construção de modelos mais fiéis à distribuição real dos dados, reduzindo sua complexidade computacional, tornar mais fáceis e rápidos o ajuste de parâmetros do modelo e seu posterior uso. Isso pode, adicionalmente, facilitar a interpretação dos padrões extraídos pelo modelo.

Técnicas de pré-processamento de dados são úteis não apenas porque podem minimizar ou eliminar problemas existentes em um conjunto de dados, mas também porque podem tornar os dados mais adequados para sua utilização por um determinado algoritmo de AM. Por exemplo, alguns algoritmos de AM trabalham apenas com valores numéricos.

Neste capítulo serão apresentadas algumas operações de pré-processamento que podem ser realizadas nos conjuntos de dados antes da sua utilização por algoritmos de AM. Essas operações englobam uso de técnicas de amostragem, tratamentos para dados desbalanceados, modificações para adequação dos tipos de atributo, limpeza dos dados, integração de dados, transformações dos dados e redução de dimensionalidade. É importante observar que não existe uma ordem fixa para a aplicação das diferentes técnicas de pré-processamento. Pode, por exemplo, ser necessário utilizar novamente uma técnica já empregada sobre uma versão anterior do conjunto de dados.

Para isso, este capítulo está organizado de acordo com as diferentes tarefas de pré-processamento. Na Seção 3.1 ilustra-se como atributos claramente irrelevantes podem ser identificados e extraídos do conjunto de dados. Na Seção 3.2 é discutida a agregação de dados de diferentes fontes. A seleção de um subconjunto dos dados originais por meio de amostragem de dados é abordada em seguida, na Seção 3.3. A Seção 3.4 trata do tema de conjuntos de dados desbalanceados. Técnicas para a limpeza de conjuntos de dados são abordadas na Seção 3.5. Na Seção 3.6 são descritas operações para transformação do tipo dos valores de um atributo. Problemas causados quando o número de atributos

preditivos é elevado e formas de reduzir a quantidade desses atributos são vistos na Seção 3.7. Comentários finais são apresentados na Seção 3.8.

3.1 Eliminação Manual de Atributos

Observando novamente a Tabela 2.1 no capítulo anterior, para o conjunto de dados hospital, pode ser facilmente percebido que nem todos os atributos do conjunto original são necessários para o diagnóstico clínico de um paciente. Não faz sentido, por exemplo, usar os valores dos atributos *Nome* e *Id.* (identificação do paciente) para o diagnóstico. Quando um atributo claramente não contribui para a estimativa do valor do atributo alvo, ele é considerado irrelevante.

O conjunto de atributos que formarão o conjunto de dados a ser analisado é geralmente definido de acordo com a experiência de especialistas no domínio dos dados. Os especialistas podem decidir, por exemplo, que atributos associados à identificação do paciente, ao nome do paciente e ao estado de origem do paciente não são relevantes para seu diagnóstico clínico. A Tabela 3.1 mostra o conjunto de dados **hospital** da Tabela 2.1 sem esses três atributos considerados irrelevantes.

Tabela 3.1 Conjunto de dados sem atributos considerados irrelevantes

Idade	Sexo	Peso	Manchas	Temp.	# Int.	Diagnóstico
28	M	79	Concentradas	38,0	2	Doente
18	F	67	Inexistentes	39,5	4	Doente
49	M	92	Espalhadas	38,0	2	Saudável
18	M	43	Inexistentes	38,5	8	Doente
21	F	52	Uniformes	37,6	1	Saudável
22	F	72	Inexistentes	38,0	3	Doente
19	F	87	Espalhadas	39,0	6	Doente
34	M	67	Uniformes	38,4	2	Saudável

Existem outras situações em que um atributo irrelevante pode ser facilmente detectado. Supor, por exemplo, que um atributo possui o mesmo valor para todos os objetos. Tal atributo não contém informação que ajude a distinguir os objetos. Assim, ele pode ser considerado irrelevante. Um atributo não precisa ter exatamente o mesmo valor para todos os objetos para ser irrelevante. Técnicas de seleção de atributos, tema que será abordado mais adiante, podem ser utilizadas para eliminar atributos irrelevantes.

3.2 Integração de Dados

Conforme mencionado anteriormente, quando dados a serem utilizados em uma aplicação de AM estão distribuídos em diferentes conjuntos de dados, esses conjuntos devem ser integrados antes do início do uso da técnica de AM. Nesse caso, é possível que cada conjunto de dados represente diferentes atributos de um mesmo grupo de objetos. Assim, na integração, é necessário identificar quais são os objetos que estão presentes nos diferentes conjuntos a serem combinados. Esse problema é conhecido como problema de identificação de entidade. Essa identificação é realizada por meio da busca por atributos

comuns nos conjuntos a serem combinados. Como exemplo, conjuntos de dados médicos podem ter um atributo que identifica o paciente. Assim, os objetos dos diferentes conjuntos que possuem o mesmo valor para o atributo que identifica o paciente são combinados em um único objeto do conjunto integrado. É fácil ver que o(s) atributo(s) utilizado(s) para combinação deve(m) ter um valor único para cada objeto.

Alguns aspectos podem dificultar a integração. Por exemplo, atributos correspondentes podem ter nomes diferentes em diferentes bases de dados. Além disso, os dados a serem integrados podem ter sido atualizados em momentos diferentes. Para minimizar esses problemas, é comum o uso de metadados em bases de dados. Os metadados são dados sobre dados que, ao descrever as principais características dos dados, podem ser utilizados para evitar erros no processo de integração. O processo de integração origina um depósito ou repositório de dados (*data warehouse*), que funciona como uma base de dados centralizada.

Com ou sem integração, é cada vez mais frequente o uso de técnicas de AM em grandes conjuntos de dados, que muitas vezes crescem com o tempo. Embora um conjunto de dados seja considerado grande se tem um número elevado de objetos, conjuntos que não possuem muitos objetos, mas em que cada objeto possui um grande número de atributos, também podem ser considerados grandes.

A existência de um conjunto de dados grande, tanto em termos de número de objetos como de atributos, não implica que um algoritmo de AM deva utilizar todo ele. Muitas vezes é mais eficiente usar apenas parte do conjunto original. No que diz respeito ao número de atributos, um número elevado de atributos pode comprometer o desempenho do algoritmo de aprendizado por causa de um problema conhecido como maldição da dimensionalidade. Técnicas para lidar com um grande número de atributos serão abordadas na Seção 3.7. Com relação à quantidade de objetos, problemas podem ocorrer por causa de saturação de memória e aumento do tempo computacional necessário para ajustar os parâmetros do modelo. Para minimizar esses problemas, podem ser utilizadas técnicas de amostragem, que serão apresentadas a seguir.

3.3 Amostragem de Dados

Algoritmos de AM podem ter dificuldades em lidar com um número grande de objetos. Esse problema é fácil de ser observado em algoritmos de AM baseados em instâncias, como *k*-vizinhos mais próximos (*k*-NN, do inglês *k-nearest neighbours* (Fix, 1951)), que podem apresentar problemas de saturação de memória quando um conjunto de dados tem um grande número de exemplos.

Associado ao número de objetos em um conjunto de dados, existe um balanço entre eficiência computacional e acurácia (taxa de predições corretas). Quanto mais dados são utilizados, maior tende a ser a acurácia do modelo e menor a eficiência computacional do processo indutivo, pois um número muito grande de objetos pode tornar o tempo de processamento muito longo. Para se obter um bom compromisso entre eficiência e acurácia, geralmente trabalha-se com uma amostra ou subconjunto dos dados. Muitas vezes, o uso de uma amostra leva ao mesmo desempenho obtido com o uso do conjunto completo, porém com um custo computacional muito menor.

Deve ser observado que uma amostra pequena pode não representar bem o problema que se deseja modelar. A amostra deve ser representativa do conjunto de dados original.

Se as amostras não forem representativas, diferentes amostras de uma mesma população podem gerar modelos diferentes. Isso porque características importantes do problema ou distribuição que gerou os dados podem não estar presentes. Por outro lado, se a amostra for muito grande, são reduzidas as vantagens de utilizar amostragem. Assim, deve novamente ser buscado um compromisso entre a eficiência e a acurácia.

O ideal é que a amostra não seja grande, mas que seus dados obedeçam à mesma distribuição estatística que gerou o conjunto de dados original. Com isso, seria capaz de fornecer uma estimativa da informação contida na população original, permitindo tirar conclusões de um todo a partir de uma parte. Por exemplo, a média dos valores para cada atributo dos dados originais deve ser semelhante à média observada nos valores desses atributos na amostra gerada. Embora não seja possível garantir que isso aconteça, existem técnicas de amostragem estatística que aumentam as chances de isso ocorrer.

Existem basicamente três abordagens para amostragem:

- Amostragem aleatória simples;
- Amostragem estratificada;
- Amostragem progressiva.

A amostragem aleatória simples possui duas variações: amostragem simples sem reposição de exemplos, em que exemplos são extraídos do conjunto original para a amostra a ser utilizada e cada exemplo pode ser selecionado apenas uma vez; e amostragem simples com reposição, quando uma cópia dos exemplos selecionados é mantida no conjunto de dados original. A amostragem com reposição é mais fácil de analisar, pois a probabilidade de escolher qualquer objeto se mantém constante. No entanto, as duas formas são semelhantes quando o tamanho da amostra é bem menor que o tamanho do conjunto original.

A amostragem estratificada é usada quando as classes apresentam propriedades diferentes, por exemplo, números de objetos bastante diferentes. Em problemas de classificação, um cuidado que deve ser tomado na amostragem diz respeito à distribuição dos dados nas diferentes classes. A existência de classes com uma quantidade significativamente maior de exemplos que as demais pode levar à indução de classificadores tendenciosos para as classes majoritárias. Nesse caso, o conjunto de dados é dito desbalanceado. Esse problema será retomado na Seção 3.4. Por outro lado, pode ser que algumas classes sejam mais difíceis de classificar que outras e isso possa ser minimizado no processo de amostragem. Essa abordagem também possui variações. A mais simples delas é manter o mesmo número de objetos para cada classe. Outra opção é manter o número de objetos em cada classe proporcional ao número de objetos da classe no conjunto original.

A terceira alternativa, a amostragem progressiva, começa com uma amostra pequena e aumenta progressivamente o tamanho da amostra extraída, enquanto a acurácia preditiva continuar a melhorar. Como resultado, é possível definir a menor quantidade de dados necessária, reduzindo ou eliminando a perda de acurácia. O tamanho pode ser confirmado com outras amostras de tamanho semelhante. Essa abordagem geralmente fornece uma boa estimativa para o tamanho da amostra.

O especialista no domínio pode decidir também que um subconjunto dos objetos deve ser utilizado para suas análises. Por exemplo, em uma análise de pacientes de um hospital, podem ser utilizados apenas os objetos correspondentes a pacientes do sexo feminino.

3.4 Dados Desbalanceados

O problema de dados desbalanceados é tópico da área de classificação de dados. Em vários conjuntos de dados reais, o número de objetos varia para as diferentes classes. Isso é comum em aplicações em que dados de um subconjunto das classes aparecem com uma frequência maior que os dados das demais classes. Usando o exemplo do conjunto de dados de pacientes de um hospital, supor que 80% dos pacientes que vão a esse hospital estão com uma dada doença. O conjunto de dados apresentará então 20% de seus objetos relacionados a pacientes saudáveis e 80% associados a pacientes com a doença. A classe com pacientes doentes seria então a classe majoritária e a de pacientes saudáveis, a classe minoritária. Outro exemplo seria um conjunto de dados de clientes de um banco, em que cada cliente é rotulado como tendo ou não ficado com o saldo da sua conta negativo nos últimos 90 dias. Se a porcentagem de clientes que ficou com saldo negativo nesse período for de 5%, a classe majoritária terá 95% dos dados.

Para ser aceitável, a acurácia preditiva de um classificador para um conjunto de dados desbalanceados deve ser maior que a acurácia obtida atribuindo todo novo objeto à classe majoritária. Vários algoritmos de AM têm seu desempenho prejudicado na presença de dados desbalanceados. Quando alimentados com dados desbalanceados, esses algoritmos tendem a favorecer a classificação de novos dados na classe majoritária. Se for possível gerar novos dados pelo mesmo processo que gerou o conjunto atual, o conjunto de dados pode ser naturalmente balanceado. Na maioria das aplicações práticas, no entanto, isso não é possível. Nesses casos, técnicas que procuram balancear artificialmente o conjunto de dados podem ser utilizadas para lidar com o problema de desbalanceamento. As principais técnicas relatadas na literatura seguem uma destas alternativas:

- Redefinir o tamanho do conjunto de dados;
- Utilizar diferentes custos de classificação para as diferentes classes;
- Induzir um modelo para uma classe.

No primeiro caso, podem ocorrer tanto o acréscimo de objetos à classe minoritária como a eliminação de objetos da classe majoritária. Para o acréscimo de novos objetos, existe o risco de os objetos acrescentados representarem situações que nunca ocorrerão, induzindo um modelo inadequado para os dados. Além disso, pode ocorrer um problema conhecido como *overfitting*, em que o modelo é superajustado aos dados de treinamento. Quando dados são eliminados da classe majoritária, é possível que dados de grande importância para a indução do modelo correto sejam perdidos. Isso pode levar ao problema de *underfitting*, em que o modelo induzido não se ajusta aos dados de treinamento. Os conceitos de *overfitting* e *underfitting* serão explicados mais adiante no livro.

A utilização de custos de classificação diferentes para as classes majoritária e minoritária tem como dificuldade a definição desses custos. Por exemplo, se o número de exemplos da classe majoritária for o dobro do número de exemplos da classe minoritária, um erro de classificação para um exemplo da classe minoritária pode equivaler à ocorrência de dois erros de classificação para um exemplo da classe majoritária. Entretanto, a definição dos diferentes custos geralmente não é tão direta. Outro problema dessa abordagem é a dificuldade de incorporar a consideração de diferentes custos em alguns algoritmos de AM. Além disso, essa abordagem pode apresentar um baixo desempenho quando boa parte

dos objetos da classe majoritária apresenta um elevado grau de semelhança. A existência de um grande número de objetos semelhantes na classe majoritária pode aproximar sua distribuição de exemplos relevantes para o treinamento da distribuição presente na classe minoritária, fazendo com que custos diferentes tenham como efeito privilegiar a classificação na classe minoritária.

O último caso inclui as técnicas de classificação com apenas uma classe, em que a classe minoritária ou a classe majoritária (ou ambas as classes) são aprendidas separadamente. Nesse caso, pode ser utilizado algoritmo de classificação para uma classe apenas (Manevitz et al., 2001). Esses algoritmos são treinados utilizando apenas exemplos da classe positiva. A classe positiva pode ser, por exemplo a classe minoritária.

3.5 Limpeza de Dados

Conjuntos de dados podem também apresentar dificuldades relacionadas à qualidade dos dados. Exemplos mais frequentes dessas dificuldades são dados ruidosos (que possuem erros ou valores que são diferentes do esperado), inconsistentes (que não combinam ou contradizem valores de outros atributos do mesmo objeto), redundantes (quando dois ou mais objetos têm os mesmos valores para todos os atributos ou dois ou mais atributos têm os mesmos valores para dois ou mais objetos) ou incompletos (com ausência de valores para alguns dos atributos em parte dos dados). Dados inconsistentes, redundantes ou com valores ausentes são de fácil detecção. A principal dificuldade está na detecção de dados ruidosos.

Essas deficiências nos dados podem ser causadas por problemas nos equipamentos que realizam a coleta, a transmissão e o armazenamento dos dados ou problemas no preenchimento ou na entrada dos dados por seres humanos.

Algumas técnicas de AM conseguem lidar bem com algumas dessas imperfeições nos dados. Por exemplo, máquinas de vetores de suporte lidam com dados que apresentam ruídos e número elevado de atributos preditivos.. Outras técnicas têm dificuldades ou não conseguem lidar com dados que apresentem algumas das deficiências mencionadas anteriormente. O algoritmo de agrupamento de dados *k*-médias, por exemplo, não apresenta um bom desempenho para dados ruidosos.

Mesmo se a técnica for robusta o suficiente para lidar com tais imperfeições, elas podem reduzir a qualidade das análises realizadas. A presença dessas deficiências em um conjunto de dados pode resultar em estatísticas e análises incorretas. Portanto, todas as técnicas se beneficiam de uma melhora na qualidade dos dados. Cada um desses problemas será detalhado a seguir.

3.5.1 Dados Incompletos

Como já mencionado, um dos problemas que pode ser encontrado em conjuntos de dados é a ausência de valores para alguns atributos de alguns objetos. Na Tabela 3.2 é ilustrado um exemplo de um conjunto de dados em que três dos objetos possuem três, dois e um atributos com valores ausentes, respectivamente. Na tabela, os atributos com valores ausentes estão destacados pelo valor “—”.

Essa ausência de valores em alguns dos atributos da Tabela 3.2 pode ter diferentes causas, entre elas:

Tabela 3.2 Conjunto de dados com atributos com valores ausentes

Idade	Sexo	Peso	Manchas	Temp.	# Int.	Diagnóstico
—	M	79	—	38,0	—	Doente
18	F	67	Inexistentes	39,5	4	Doente
49	M	92	Espalhadas	38,0	2	Saudável
18	—	43	Inexistentes	38,5	8	Doente
21	F	52	Uniformes	37,6	1	Saudável
22	F	72	Inexistentes	38,0	3	Doente
—	F	87	Espalhadas	39,0	6	Doente
34	M	67	Uniformes	38,4	2	Saudável

- O atributo não foi considerado importante quando os primeiros dados foram coletados. Supor, por exemplo, que um dos atributos seja o *email* do paciente, que não era comum na década de 1990.
- Desconhecimento do valor do atributo por ocasião do preenchimento dos valores do objeto. Uma possível situação seria não saber o tipo sanguíneo de um paciente quando foi feito o seu cadastro.
- distração na hora do preenchimento.
- Falta de necessidade ou obrigação de apresentar um valor para atributo(s) para alguns objetos. Se houver um atributo renda, por exemplo, alguns pacientes podem não querer preenchê-lo.
- Inexistência de um valor para o atributo em alguns objetos. Poderia ocorrer se um dos atributos especificar o número de partos e o paciente em questão for do sexo masculino.
- Problema com equipamento ou processo utilizado para coleta, transmissão e armazenamento de dados.

Algumas técnicas de AM podem gerar erro de execução quando um ou mais atributos do conjunto de treinamento não apresentam valor. Várias alternativas têm sido propostas para lidar com esses atributos. As alternativas mais utilizadas são:

- Eliminar os objetos com valores ausentes. Essa alternativa é geralmente empregada quando um dos atributos com valores ausentes em um objeto é o que indica a sua classe. Essa alternativa não é indicada quando poucos atributos do objeto possuem valores ausentes, quando o número de atributos com valores ausentes varia muito entre os objetos com esse problema ou quando o número de objetos que restarem for pequeno.
- Definir e preencher manualmente valores para os atributos com valores ausentes. Essa alternativa não é factível quando o número de objetos ou atributos com valores ausentes for muito grande.

- Utilizar algum método ou heurística para automaticamente definir valores para atributos com valores ausentes. Essa é a alternativa mais utilizada. Diferentes abordagens podem ser seguidas para tal, como será discutido em seguida.
- Empregar algoritmos de AM que lidam internamente com valores ausentes. Esse é o caso, por exemplo, de alguns algoritmos indutores de árvores de decisão (Seção 6.1).

A definição automática de valores para completar os valores ausentes tem seguido três abordagens diferentes na literatura:

- Criar para o atributo um novo valor que indique que o atributo possuía um valor desconhecido. Esse valor pode ser comum a todos os atributos ou um valor diferente para cada atributo. O problema dessa alternativa é que o algoritmo indutor pode assumir que o valor desconhecido representa um conceito importante.
- Utilizar a média, moda (no caso de valor simbólico) ou mediana dos valores conhecidos para esse atributo. Essa medida pode ser calculada utilizando todos os objetos ou apenas os objetos da mesma classe do objeto com o atributo a ser preenchido. Uma outra variação dessa abordagem utiliza o valor mais frequente nos k objetos mais semelhantes daquele com o valor ausente a ser estimado. Se os objetos tiverem uma relação temporal, a medida pode ser calculada utilizando os objetos associados ao instante imediatamente anterior e posterior ao objeto modificado.
- Empregar um indutor para estimar o valor do atributo. Para isso, o valor a ser definido seria o atributo alvo e os demais atributos seriam os atributos de entrada. A vantagem desse método é justamente a utilização de informação presente nos demais atributos para inferir o valor do atributo ausente. Essa abordagem é a mais popular. Em geral, resulta na utilização do valor utilizado em objetos semelhantes.

Se os atributos com valores ausentes forem substituídos pelo valor da média (valores numéricos) ou da moda (valores simbólicos) dos valores de cada um desses atributos, a Tabela 3.2 seria substituída pela Tabela 3.3. A imputação de valores ausentes pela média pode levar a inconsistências, como, por exemplo, um paciente de 2 anos de idade com peso igual a 60 quilos.

Tabela 3.3 Conjunto de dados com substituição dos valores ausentes

Idade	Sexo	Peso	Manchas	Temp.	# Int.	Diagnóstico
27	M	79	Inexistentes	38,0	4	Doente
18	F	67	Inexistentes	39,5	4	Doente
49	M	92	Espalhadas	38,0	2	Saudável
18	F	43	Inexistentes	38,5	8	Doente
21	F	52	Uniformes	37,6	1	Saudável
22	F	72	Inexistentes	38,0	3	Doente
27	F	87	Espalhadas	39,0	6	Doente
34	M	67	Uniformes	38,4	2	Saudável

3.5.2 Dados Inconsistentes

Dados inconsistentes são aqueles que possuem valores conflitantes em seus atributos. Essa inconsistência pode se dar entre valores de atributos de entrada (por exemplo, valor 120 para o atributo *Peso* e o valor 3 para o atributo *Idade*) ou entre todos os valores dos atributos de entrada e o valor do atributo de saída (por exemplo, dois pacientes com os mesmos valores para os atributos de entrada e diagnósticos diferentes, um saudável e o outro doente). Inconsistências podem ser identificadas quando relações conhecidas entre os atributos são violadas. Por exemplo, quando é sabido que os valores de um atributo variam de forma inversamente proporcional em relação aos valores de um outro atributo. Dados inconsistentes podem ser resultado do processo de integração de dados de fontes ou tabelas diferentes ou da presença de ruídos nos dados.

Dados inconsistentes são muitas vezes produzidos no processo de integração de dados. Por exemplo, diferentes conjuntos de dados podem usar escalas diferentes para uma mesma medida (metros e centímetros) ou codificar de forma diferente para um atributo relacionado a tamanho.

Na Tabela 3.4 é ilustrado um exemplo de conjunto de dados com inconsistência. Essa inconsistência pode ser percebida nos dois objetos destacados, que apresentam valores de entrada iguais para valores de saída diferentes.

Tabela 3.4 Conjunto de dados com objetos inconsistentes

Idade	Sexo	Peso	Manchas	Temp.	# Int.	Diagnóstico
28	M	79	Concentradas	38,0	2	Doente
18	F	67	Inexistentes	39,5	4	Doente
49	M	92	Espalhadas	38,0	2	Saudável
18	M	43	Inexistentes	38,5	8	Doente
21	F	52	Uniformes	37,6	1	Saudável
22	F	72	Inexistentes	38,0	3	Doente
19	F	87	Espalhadas	39,0	6	Doente
22	F	72	Inexistentes	38,0	3	Saudável

Existem formas de prevenir a ocorrência de inconsistências. Algoritmos simples podem verificar automaticamente se relacionamentos existentes entre atributos são violados. Quando o conjunto de dados não é muito grande, dados inconsistentes podem ser removidos manualmente.

3.5.3 Dados Redundantes

Um conjunto de dados pode possuir tanto objetos como atributos redundantes. Um objeto é redundante quando ele é muito semelhante a um outro objeto do mesmo conjunto de dados, ou seja, seus atributos possuem valores muito semelhantes aos atributos de pelo menos um outro objeto. No caso extremo, possui o mesmo valor para cada um dos atributos. Um atributo é redundante quando seu valor para todos os objetos pode ser deduzido a partir do valor de um ou mais atributos. No caso extremo, possui o mesmo valor que um outro atributo para cada um dos objetos do conjunto de dados. Problemas na coleta, na entrada, no armazenamento, na integração ou na transmissão de dados podem gerar objetos ou atributos redundantes.

Para ilustrar a presença de objetos redundantes, observar a Tabela 3.5, que apresenta redundância entre três objetos. Nessa tabela, o segundo e o quarto pacientes têm os mesmos valores para todos os atributos, e por isso são considerados objetos redundantes, conforme destacado.

Tabela 3.5 Conjunto de dados com objetos redundantes

Idade	Sexo	Peso	Manchas	Temp.	# Int.	Diagnóstico
28	M	79	Concentradas	38,0	2	Doente
18	F	67	Inexistentes	39,5	4	Doente
49	M	92	Espalhadas	38,0	2	Saudável
18	F	67	Inexistentes	39,5	4	Doente
18	M	43	Inexistentes	38,5	8	Doente
21	F	52	Uniformes	37,6	1	Saudável
22	F	72	Inexistentes	38,0	3	Doente
19	F	87	Espalhadas	39,0	6	Doente
34	M	67	Uniformes	38,4	2	Saudável

Objetos redundantes em um conjunto de dados participam mais de uma vez do processo de ajuste de parâmetros de um modelo, contribuindo assim mais que os outros objetos para a definição do modelo final. Isso pode dar ao modelo a falsa impressão de que esse perfil de objeto é mais importante que os demais. Como resultado, o tempo necessário para a indução de um modelo pode aumentar. Por isso, geralmente é desejável a eliminação de redundâncias, que pode ser feita em dois passos:

- Identificação de objetos redundantes;
- Eliminação das redundâncias encontradas.

A eliminação da redundância pode ocorrer pela eliminação dos objetos semelhantes a um dado objeto ou pela combinação dos valores dos atributos dos objetos semelhantes. Essa eliminação de redundância geralmente é feita no final do processo de limpeza.

Se a redundância não for eliminada, o algoritmo de AM utilizado pode atribuir ao objeto repetido uma importância maior que aos demais objetos. Por exemplo, um objeto *A* com duas cópias adicionais em um conjunto de dados é considerado pelo algoritmo de indução três vezes mais importante que um objeto *B* que não possui cópias. Se o objeto *A* “puxar” a fronteira de decisão para o lado esquerdo e o objeto *B* para o lado direito, o movimento da fronteira para o lado esquerdo será três vezes maior que para o lado direito. Deve ser observado que algumas técnicas de AM, como *boosting*, utilizam esse artifício para duplicar a quantidade de exemplos difíceis de ser classificados.

Conforme dito no início da seção, não apenas objetos, mas atributos também podem apresentar redundância. Um atributo é considerado redundante se seu valor puder ser estimado a partir de pelo menos um dos demais atributos. Isso ocorre quando dois ou mais atributos têm a mesma informação preditiva. No caso extremo, dois atributos podem compartilhar o mesmo valor entre si para cada um dos objetos de um conjunto de dados. Um exemplo simples de redundância de atributos é a presença de um atributo *Idade* e de um atributo *Data de nascimento* em um conjunto de dados, pois é fácil definir o valor do atributo *Idade* usando o valor do atributo *Data de nascimento*. Outro exemplo

com mais de dois atributos seria ter um atributo *Quantidade de vendas*, um atributo *Valor por venda* e um atributo *Venda total*. Nesse caso, o valor do atributo *Venda total* pode ser facilmente definido a partir do valor dos dois outros atributos. Um atributo redundante pode supervalorizar um dado aspecto dos dados, por estar presente mais de uma vez, ou tornar mais lento o processo de indução, por causa do maior número de atributos a serem analisados pelo algoritmo de AM. Assim, o desempenho de um algoritmo de AM geralmente melhora com a eliminação de atributos redundantes. Muitas vezes a redundância entre atributos não é tão clara. Atributos redundantes são geralmente eliminados por técnicas de seleção de atributos, como será visto mais adiante.

A redundância de um atributo está relacionada à sua correlação com um ou mais atributos do conjunto de dados. Dois ou mais atributos estão correlacionados quando apresentam um perfil de variação semelhante para os diferentes objetos. Quanto mais correlacionados os atributos, maior o grau de redundância. Se a correlação ocorrer entre um atributo de entrada e um atributo rótulo, esse atributo de entrada terá uma grande influência na predição do valor do atributo rótulo.

Na Tabela 3.6 é ilustrado um conjunto de dados em que um dos atributos é claramente redundante. Essa tabela adiciona à Tabela 2.1 o atributo número de visitas (*# Vis.*), que indica quantas vezes um dado paciente esteve no hospital. O atributo redundante, *# Vis.*, está destacado em negrito. Nesses casos, para remover a redundância, basta manter um dos atributos e eliminar o outro.

Tabela 3.6 Conjunto de dados com atributos redundantes

Idade	Sexo	Peso	Manchas	Temp.	# Int.	# Vis.	Diagnóstico
28	M	79	Concentradas	38,0	2	2	Doente
18	F	67	Inexistentes	39,5	4	4	Doente
49	M	92	Espalhadas	38,0	2	2	Saudável
18	M	43	Inexistentes	38,5	8	8	Doente
21	F	52	Uniformes	37,6	1	1	Saudável
22	F	72	Inexistentes	38,0	3	3	Doente
19	F	87	Espalhadas	39,0	6	6	Doente
34	M	67	Uniformes	38,4	2	2	Saudável

3.5.4 Dados com Ruídos

Dados com ruídos são dados que contêm objetos que, aparentemente, não pertencem à distribuição que gerou os dados analisados. Ruído pode ser definido como uma variância ou erro aleatório no valor gerado ou medido para um atributo (Han e Kamber, 2000). Dados inconsistentes podem ser resultado da presença de ruídos.

Várias podem ser as causas da presença de ruídos, os quais foram discutidos anteriormente na Seção 3.5. Dados com ruídos podem levar a um superajuste do modelo utilizado, pois o algoritmo que induz o modelo pode se ater às especificidades relacionadas aos ruídos, em vez da distribuição verdadeira que gerou os dados. Por outro lado, a eliminação de dados ruidosos pode levar à perda de informação importante. A eliminação desses dados pode fazer com que algumas regiões do espaço de atributos não sejam consideradas no processo de indução de hipóteses.

É importante observar que não é possível ter certeza de que um valor é ou não resultado da presença de ruído, mas apenas ter uma indicação ou indício de que um dado valor para um atributo pode ter sido gerado com ruído. Um indicador da possível presença de ruído é a existência de *outliers*, que são valores que estão além dos limites aceitáveis ou são muito diferentes dos demais valores observados para o mesmo atributo, representando, por exemplo, exceções raramente vistas. Na Tabela 3.7, o valor do atributo *peso* do segundo objeto é um *outlier*.

Tabela 3.7 Conjunto de dados com ruído

Idade	Sexo	Peso	Manchas	Temp.	# Int.	Diagnóstico
28	M	79	Concentradas	38,0	2	Doente
18	F	300	Inexistentes	39,5	4	Doente
49	M	92	Espalhadas	38,0	2	Saudável
18	M	43	Inexistentes	38,5	8	Doente
21	F	52	Uniformes	37,6	1	Saudável
22	F	72	Inexistentes	38,0	3	Doente
19	F	87	Espalhadas	39,0	6	Doente
34	M	67	Uniformes	38,4	2	Saudável

Existem diversas técnicas de pré-processamento que podem ser aplicadas na detecção e remoção de ruídos. Em Estatística, esse problema é comumente solucionado por meio de técnicas baseadas em distribuição, em que os ruídos são identificados como observações que diferem de uma distribuição utilizada na modelagem dos dados (Barnett e Lewis, 1994). O maior problema dessa abordagem está em assumir que a distribuição dos dados é conhecida *a priori*, o que não reflete a verdade em grande parte das aplicações práticas.

Uma segunda categoria de técnicas aplicadas em Estatística utiliza o conceito de profundidade na busca por ruídos (Barnett e Lewis, 1994; Nuts e Rousseeuw, 1996). Baseados em alguma definição de profundidade, os dados são organizados em camadas. Os ruídos são identificados como objetos pertencentes a níveis superficiais. Esse tipo de estratégia é computacionalmente custosa, principalmente para dados de grandes dimensões.

Diversas outras técnicas podem ser utilizadas para reduzir o ruído em um atributo. De forma resumida, elas podem ser reunidas em cinco grupos:

- Técnicas de encestamento: Essas técnicas suavizam o valor de um atributo da seguinte forma. Primeiro, os valores encontrados para esse atributo em todos os objetos são ordenados. Em seguida, esses valores são divididos em faixas ou cestas, cada uma com o mesmo número de valores. Os valores em uma mesma cesta são substituídos, por exemplo, pela média ou mediana dos valores presentes na cesta. Alternativas podem ser utilizadas para a substituição dos valores.
- Técnicas baseadas em agrupamento dos dados: Essas técnicas podem ser utilizadas tanto para os objetos como para os atributos. No caso dos atributos, os valores dos atributos são agrupados por uma técnica de agrupamento (Capítulo 12). Valores de atributos que não formarem um grupo com outros valores são considerados ruídos ou *outliers*. O mesmo é dito de objetos que forem colocados em um grupo no qual os demais objetos pertencem a uma outra classe.

- Técnicas baseadas em distância: A presença de ruído em um ou mais atributos de um objeto frequentemente faz com que esse objeto se distancie dos demais objetos de sua classe. As técnicas baseadas em distância verificam a que classe pertencem os objetos mais próximos de cada objeto x . Se esses objetos mais próximos pertencem a uma outra classe, são boas as chances de o objeto x apresentar ruído, embora possa também ser *borderline* (próximo à fronteira de separação das classes). De qualquer forma, *borderlines* podem ser considerados relativamente inseguros, uma vez que mesmo uma pequena quantidade de ruídos pode mover-lhos para o lado incorreto da fronteira.
- Técnicas baseadas em regressão ou classificação: As técnicas baseadas em regressão utilizam uma função de regressão para, dado um valor com ruído, estimar seu valor verdadeiro. Se o valor a ser estimado for simbólico, uma técnica de classificação pode ser utilizada.

3.6 Transformação de Dados

Várias técnicas de AM estão limitadas à manipulação de valores de determinados tipos, por exemplo, apenas valores numéricos ou apenas valores simbólicos. Adicionalmente, algumas técnicas têm seu desempenho influenciado pelo intervalo de variação dos valores numéricos. Esta seção divide as diferentes técnicas para abordar esse problema em três partes. A primeira parte descreve técnicas que podem ser utilizadas para converter valores simbólicos em valores numéricos. A depender se os valores simbólicos são nominais ou ordinais, diferentes técnicas podem ser empregadas. A segunda parte mostra técnicas para converter valores numéricos em valores simbólicos. Finalmente, a terceira mostra casos em que a conversão não altera o tipo do atributo, notadamente relacionados a atributos com valores numéricos, ou seja, transformações que podem envolver, por exemplo, mudança de escala ou de intervalo de valores.

3.6.1 Conversão Simbólico-Numérico

Técnicas como redes neurais artificiais e *support vector machines* e vários algoritmos de agrupamento lidam apenas com dados numéricos. Assim, quando o conjunto de dados utilizado por essas técnicas apresenta atributos simbólicos, os valores desses atributos devem ser convertidos para valores numéricos.

Quando o atributo é do tipo nominal e assume apenas dois valores, se os valores denotam a presença ou ausência de uma característica ou se apresentam uma relação de ordem, um dígito binário é suficiente. No primeiro caso, o valor 0 indica a ausência e o valor 1, a presença da característica. No segundo caso, o menor valor ordinal assume o valor zero e o outro assume o valor 1.

Para um atributo simbólico com mais de dois valores, a técnica utilizada na conversão depende de o atributo ser nominal ou ordinal. Se não houver uma relação de ordem entre os valores do atributo, a inexistência de uma relação de ordem deve continuar para os valores numéricos gerados. Ou seja, a diferença entre quaisquer dois valores numéricos deve ser a mesma. Uma forma de conseguir isso é codificar cada valor nominal por uma sequência de c bits, em que c é igual ao número de possíveis valores ou categorias.

Na codificação $1 - de - c$, também denominada canônica ou topológica, cada sequência possui apenas um bit com o valor 1 e os demais com o valor zero. A diferença entre as sequências é definida pela posição que o valor 1 ocupa nelas. Para definir a diferença entre dois valores, pode ser utilizada a distância de Hamming. A distância de Hamming entre duas sequências binárias com mesmo número de elementos é igual ao número de posições em que as sequências apresentam valores diferentes. É fácil verificar que a distância de Hamming entre qualquer par de valores é igual a 2 (apenas duas posições do *string* binário têm valores diferentes).

Nessa codificação, cada posição da sequência binária corresponde a um possível valor do atributo nominal. Por exemplo, se a sequência binária possui 4 bits, o primeiro bit corresponde ao primeiro valor, o segundo bit ao segundo valor e assim por diante. Como apenas um dos bits pode assumir o valor 1, o bit que assumir esse valor sinaliza a presença do valor nominal correspondente àquele bit. A moda do valor de um atributo para o conjunto de objetos é definida pela posição (bit) da sequência que apresenta o maior número de valores iguais a 1. Isso indica que o valor correspondente àquela posição é o que aparece com maior frequência no conjunto de dados.

Na Tabela 3.8 é ilustrada a codificação $1 - de - c$ para um conjunto de seis valores nominais, em que cada valor representa uma cor.

Dependendo do número de valores nominais, a sequência binária para representar cada valor pode ficar muito longa. Imagine que você queira codificar os nomes de países com a codificação $1 - de - c$. Como existem 193 países (incluindo o Vaticano), seria necessário utilizar vetores com 193 elementos.

Tabela 3.8 Codificação $1 - de - c$

Atributo nominal	Código $1 - de - c$
Azul	100000
Amarelo	010000
Verde	001000
Preto	000100
Marrom	000010
Branco	000001

Uma alternativa é a representação dos possíveis valores nominais por um conjunto de pseudoatributos. Os valores dos pseudoatributos podem ser do tipo binário, inteiro ou real. Para o exemplo anterior, a Tabela 3.9 mostra como é possível transformar um conjunto de 193 atributos, um para cada país, em um conjunto de cinco pseudoatributos, que podem ser utilizados para identificar cada um dos países (os valores entre parênteses denotam se o valor é do tipo binário (b) ou inteiro (i), respectivamente). O primeiro desses cinco pseudoatributos é o atributo *continente*, um atributo nominal que pode assumir um dentre sete possíveis valores. Os outros quatro pseudoatributos, *Produto Interno Bruto (PIB)*, *População*, temperatura média anual (*TMA*) e *Área* podem assumir um valor inteiro cada. Assim como para o atributo original, uma combinação de valores para os cinco pseudoatributos representa um único país.

Quando existe uma relação de ordem, o atributo é do tipo ordinal, e a codificação deve preservar essa relação. Para isso, deve ser utilizada uma codificação em que a ordem dos valores esteja clara. Quando o valor numérico é um número inteiro ou real, essa

Tabela 3.9 Pseudoatributos e seus possíveis valores

Pseudoatributo	#Valores
Continente	7 (b)
PIB	1 (i)
População	1 (i)
TMA	1 (i)
Área	1 (i)

transformação é simples e direta: basta ordenar os valores categóricos ordinais e codificar cada valor de acordo com sua posição na ordem, como ilustrado no exemplo da Tabela 3.10.

Tabela 3.10 Conversão de valor ordinal para inteiro

Valor ordinal	Valor inteiro
Primeiro	0
Segundo	1
Terceiro	2
Quarto	3
Quinto	4
Sexto	5

Nessa tabela, a distância entre os valores varia de acordo com a proximidade deles, ao contrário do que ocorreu com a codificação ilustrada na Tabela 3.8.

Se for necessário converter valores ordinais em valores binários, pode ser utilizado o código cinza ou o código termômetro. O código cinza é constantemente utilizado para correções de erro em comunicações digitais. No código termômetro, o aumento dos valores se assemelha ao aumento de temperatura em um termômetro analógico. Esses dois códigos são ilustrados na Tabela 3.11.

Tabela 3.11 Conversão de valor ordinal para binário

Valor ordinal	Código cinza	Código termômetro
Primeiro	000	00000
Segundo	001	00001
Terceiro	011	00011
Quarto	010	00111
Quinto	110	01111
Sexto	100	11111

Pode ser visto nessa tabela que, nos dois casos, dois valores próximos diferem por apenas um bit (possuem distância de Hamming igual a 1). A tabela também mostra que o código termômetro, em que um valor 1 é acrescentado à medida que se avança na escala de valores, utiliza sequências binárias maiores (mais bits) que o código cinza.

3.6.2 Conversão Numérico-Simbólico

Algumas técnicas de AM foram desenvolvidas para trabalhar com valores qualitativos. Isso ocorre com uma parcela dos algorítimos de classificação e de associação. Alguns desses algoritmos que também podem lidar com dados quantitativos têm seu desempenho reduzido quando o fazem. Se o atributo quantitativo for do tipo discreto e binário, com apenas dois valores, a conversão é trivial. Basta associar um nome a cada valor. Se o atributo original for formado por sequências binárias sem uma relação de ordem entre si, cada sequência pode ser substituída por um nome ou categoria.

Nos demais casos, métodos de discretização permitem transformar atributos quantitativos em qualitativos. Para isso, eles transformam valores numéricos em intervalos ou categorias. A escolha do método de discretização depende do problema de aprendizado. Métodos de discretização podem ser utilizados de forma isolada ou composta, quando mais de um método é utilizado. Existe um grande número de métodos, que podem ser classificados de acordo com diferentes critérios (Yang et al., 2005).

Quando um atributo quantitativo é discretizado, o conjunto de possíveis valores é dividido em intervalos, e cada intervalo de valores quantitativos é convertido em um valor qualitativo. Em alguns métodos, o usuário pode influenciar a definição dos intervalos, definindo valores para parâmetros como número máximo de intervalos. Esses métodos são denominados paramétricos. Os métodos não paramétricos definem os intervalos utilizando apenas as informações presentes nos valores do atributo.

Os métodos de discretização podem ser supervisionados ou não supervisionados. No primeiro caso, é utilizada a informação sobre a classe dos exemplos. As técnicas supervisionadas geralmente levam a melhores resultados, uma vez que a definição dos intervalos sem conhecimento das classes pode levar a mistura de classes. Uma abordagem supervisionada simples seria escolher pontos de corte que maximizam a pureza dos intervalos. Isso pode ser feito utilizando o conceito de entropia. Deve ser observado que discretizar cada atributo separadamente em geral leva a resultados subótimos.

A definição de como mapear valores dos atributos quantitativos para valores qualitativos e do tamanho dos intervalos ou a quantidade de valores nos intervalos fica geralmente a cargo do método de discretização. Algumas das estratégias utilizadas pelos diferentes métodos são apresentadas a seguir:

- Larguras iguais: Divide o intervalo original de valores em subintervalos com mesma largura. O desempenho dessa estratégia pode ser afetado pela presença de *outliers*.
- Frequências iguais: Atribui o mesmo número de objetos a cada subintervalo. Essa estratégia pode gerar intervalos de tamanhos muito diferentes.
- Uso de um algoritmo de agrupamento de dados (Capítulo 12).
- Inspeção visual.

3.6.3 Transformação de Atributos Numéricos

Algumas vezes, o valor numérico de um atributo precisa ser transformado em outro valor numérico. Isso geralmente ocorre quando os limites inferior e superior de valores dos atributos são muito diferentes, o que leva a uma grande variação de valores, ou ainda quando vários atributos estão em escalas diferentes. Essa transformação é geralmente

realizada para evitar que um atributo predomine sobre outro. No entanto, pode haver situações em que essa variação deve ser preservada por ser importante para a indução de um bom modelo.

Quando necessário, a operação de transformação é aplicada aos valores de um dado atributo de todos os objetos. Por exemplo, supor que, dado um atributo com valores inteiros, para a indução de um bom modelo apenas a magnitude dos valores seja importante, não o sinal. Uma transformação necessária seria então converter os valores desse atributo para seu valor absoluto.

Outra transformação que é muito utilizada é a normalização de dados. A normalização de dados é recomendável quando os limites de valores de atributos distintos são muito diferentes, para evitar que um atributo predomine sobre outro (a menos que isso seja importante). Quando recomendada, ela é aplicada a cada atributo individualmente e pode ocorrer de duas formas:

- Amplitude;
- Distribuição.

A normalização por amplitude pode ser por reescala ou padronização. A primeira define uma nova escala de valores, limites mínimo e máximo, para todos os atributos. A segunda define um valor central e um valor de espalhamento comuns para todos os atributos.

Na normalização por reescala, também chamada normalização min-max, são inicialmente definidos os valores mínimo (min) e máximo (max) para os novos valores de cada atributo. Depois, as operações a seguir são realizadas para cada atributo. Primeiro, o menor valor do atributo, (menor), é subtraído de cada valor. Cada valor resultante é em seguida dividido pela diferença entre o maior e o menor valores originais do atributo, (maior – menor). Cada novo valor é depois multiplicado pela diferença entre os valores limites da nova escala, max – min. No final, o valor min é somado a cada valor produzido. Essas operações são ilustradas pela Equação 3.1. Para que os limites superior e inferior sejam 1 e 0, respectivamente, basta fazer max = 1 e min = 0.

$$v_{Novo} = \min + \frac{v_{Atual} - \text{menor}}{\text{maior} - \text{menor}} (\text{max} - \text{min}) \quad (3.1)$$

Para a normalização por padronização, a cada valor do atributo a ser normalizado é adicionada ou subtraída uma medida de localização e o valor resultante é em seguida multiplicado ou dividido por uma medida de escala. Com isso, diferentes atributos podem ter limites inferiores e superiores diferentes, mas terão os mesmos valores para as medidas de escala e espalhamento. Se as medidas de localização e de escala forem a média (μ) e a variância (σ), respectivamente, os valores de um atributo são convertidos para um novo conjunto de valores com média 0 e variância 1, que é obtido se for utilizada a Equação 3.2 nos valores originais dos atributos.

$$v_{Novo} = \frac{v_{Atual} - \mu}{\sigma} \quad (3.2)$$

Geralmente, é preferível padronizar a reescalar, pois a padronização lida melhor com *outliers*. É possível durante a operação normalização fazer com que os atributos mais

importantes possuam limites maiores. Para isso, basta fazer com que a padronização gere um maior intervalo (por exemplo, o limite máximo de um atributo importante pode ser o dobro do limite máximo utilizado para os demais atributos) ou que a medida de reescala gere uma maior variância para um atributo importante (por exemplo, esse atributo pode ter uma variância que é o dobro da variância dos demais atributos, o que faz com que sua faixa de valores seja o dobro da faixa utilizada pelos demais atributos).

A normalização por distribuição muda a escala de valores de um atributo. Um exemplo dessa normalização é a aplicação da função para ordenar os valores do atributo a ser normalizado e a substituição de cada valor pela posição que ele ocupa no *ranking* (por exemplo, a aplicação dessa normalização aos valores 1, 5, 9 e 3 gera, respectivamente, os valores 1, 3, 4 e 2). Se todos os valores originais forem distintos, o resultado é uma distribuição uniforme.

Um outro tipo de transformação para atributos de um mesmo tipo é a tradução, em que o valor de um atributo de um dado tipo é traduzido para um valor do mesmo tipo, mais facilmente manipulável. Por exemplo, a conversão de um atributo com data de nascimento para idade, de graus Celsius para Fahrenheit ou de localização dada por um aparelho de GPS para código postal.

3.7 Redução de Dimensionalidade

Muitos problemas que podem ser tratados por técnicas de AM apresentam um número elevado de atributos. Um exemplo de grupo de problemas em que o número de atributos é muito grande são as aplicações de reconhecimento de imagens. Se cada pixel (um pixel é a unidade básica de uma imagem) da imagem for considerado um atributo, cada imagem ou instância de uma imagem com 1024 por 1024 pixels teria mais de um milhão de atributos. Outro exemplo são os dados de expressão gênica, que geralmente apresentam algumas dezenas de objetos, cada um com milhares de atributos, quando se está trabalhando com as amostras como objetos e os genes como atributos. Poucas técnicas de AM podem lidar com um número tão grande de atributos.

O efeito do número muito grande de atributos em algoritmos de AM é descrito pelo problema da maldição da dimensionalidade. Se cada atributo for visto como uma coordenada em um espaço d -dimensional, em que d é o número de atributos, o hipervolume que representa esse espaço cresce exponencialmente com a adição de novos atributos. Para tornar mais claro esse problema, supor um conjunto de dados em que cada objeto possui apenas um atributo e que esse atributo pode assumir um dentre 10 valores. Esse conjunto de dados pode ter então 10^1 ou 10 objetos diferentes, um para cada valor diferente do atributo. Se o número de atributos passar para cinco, o número de possíveis objetos passa a ser 10^5 , que é um número de possíveis objetos muito maior do que quando apenas um atributo foi utilizado. Uma forma de minimizar o impacto do problema da dimensionalidade é combinar ou eliminar parte dos atributos irrelevantes.

Em muitos algoritmos de AM, para que dados com um número elevado de atributos possam ser utilizados, a quantidade de atributos precisa ser reduzida. A redução do número de atributos pode ainda melhorar o desempenho do modelo induzido, reduzir seu custo computacional e tornar os resultados obtidos mais compreensíveis. Diferentes técnicas originárias de áreas de pesquisa como Reconhecimento de Padrões, Estatística e

Teoria da Informação podem ser utilizadas para a redução do número de atributos. Essas técnicas podem ser divididas em duas grandes abordagens:

- Agregação;
- Seleção de Atributos.

Enquanto as técnicas de agregação substituem os atributos originais por novos atributos formados pela combinação de grupos de atributos, as técnicas de seleção mantêm uma parte dos atributos originais e descartam os demais atributos.

3.7.1 Agregação

As principais técnicas utilizadas para reduzir as dimensões por agregação combinam os atributos originais por meio de funções lineares ou não lineares. Uma das técnicas mais conhecidas é a de Análise de Componentes Principais (PCA, do inglês *Principal Component Analysis*) (Pearson, 1901). A técnica PCA correlaciona estatisticamente os exemplos, reduzindo a dimensionalidade do conjunto de dados original pela eliminação de redundâncias.

As técnicas de agregação, ao combinar os atributos, levam à perda dos valores originais. Em várias aplicações, por exemplo, nas áreas de biologia, finanças, medicina e monitoramento ambiental, geralmente é importante preservar os valores dos atributos para que os resultados obtidos possam ser interpretados, associando os resultados produzidos por uma técnica estatística ou de AM aos valores dos atributos. Por isso, nessas áreas, é mais frequente a redução do número de atributos pelo uso de técnicas de seleção.

3.7.2 Seleção de Atributos

Várias aplicações reais apresentam um grande número de atributos. Além do problema da maldição da dimensionalidade, parte desses atributos pode ser irrelevante, redundante ou conter grande quantidade de ruído.

A seleção de atributos permite:

- Identificar atributos importantes;
- Melhorar o desempenho de várias técnicas de AM;
- Reduzir a necessidade de memória e tempo de processamento;
- Eliminar atributos irrelevantes e reduzir ruído;
- Lidar com a maldição da dimensionalidade;
- Simplificar o modelo gerado e tornar mais fácil sua compreensão;
- Facilitar a visualização dos dados;
- Reduzir o custo de coleta de dados e com isso aumentar o acesso a novas tecnologias.

Conforme visto anteriormente, alguns atributos são claramente redundantes ou irrelevantes, podendo ser manualmente eliminados. No entanto, na prática, vários atributos passíveis de eliminação não são facilmente identificados, o que torna pouco eficiente o uso apenas de técnicas visuais. Dentre as razões para essa dificuldade, podem ser citadas:

- Número muito grande de exemplos;
- Número muito grande de atributos;
- Relações complexas entre atributos, que dificultam a descoberta de relações entre eles.

Para lidar com esses casos, diversas técnicas automáticas têm sido propostas na literatura para a seleção de atributos. Essas técnicas, de forma geral, procuram por um subconjunto ótimo de atributos de acordo com um dado critério. As técnicas propostas podem ser classificadas de diferentes formas.

Uma delas diz respeito à avaliação do conjunto de atributos selecionados. Nesse caso, as técnicas existentes podem estar integradas a um algoritmo de indução ou ser independentes do algoritmo. Para avaliar a qualidade ou desempenho de um subconjunto de atributos, três abordagens são então utilizadas:

- Embutida;
- Baseada em filtro;
- Baseada em *wrapper*.

Na abordagem embutida, a seleção do subconjunto é embutida ou integrada no próprio algoritmo de aprendizado. As árvores de decisão (Seção 6.1), por exemplo, realizam esse tipo de seleção interna de atributos.

Na abordagem baseada em filtro, em uma etapa de pré-processamento, é utilizado um filtro sobre o conjunto de atributos original que filtra um subconjunto de atributos do conjunto original, sem levar em consideração o algoritmo de aprendizado que utilizará esse subconjunto. As técnicas que seguem essa abordagem verificam, por exemplo, correlação entre atributos e são geralmente mais rápidas que as baseadas em *wrapper*. A abordagem baseada em *wrapper* utiliza o próprio algoritmo de aprendizado como uma caixa-preta para a seleção. Geralmente é utilizada junto com uma técnica de amostragem. Para cada possível subconjunto, o algoritmo é consultado e o subconjunto que apresentar a melhor combinação entre redução da taxa de erro e redução do número de atributos é em geral selecionado.

Algumas vantagens da abordagem baseada em filtro podem ser destacadas:

- como o processo de seleção não depende de nenhum indutor, as características selecionadas podem ser utilizadas por diferentes algoritmos de AM;
- as heurísticas utilizadas para avaliar um subconjunto são computacionalmente pouco custosas, assim os filtros podem ser bastante rápidos;
- os filtros conseguem lidar eficientemente com uma grande quantidade de dados.

A principal desvantagem dos filtros refere-se a sua independência em relação ao algoritmo de AM. Como a seleção de atributos e a classificação são processos separados, o viés¹ de um não interage com o viés de outro, o que pode levar à construção de classificadores com desempenho aquém do desejado.

Técnicas baseadas em *wrapper* representam uma alternativa simples e poderosa para selecionar atributos. Elas são algumas vezes criticadas por serem técnicas de força bruta, com um custo computacional elevado. Mas isso não é sempre verdade, estratégias de busca eficientes têm sido utilizadas por algumas dessas técnicas.

Em geral, as técnicas embutidas fazem melhor uso dos dados disponíveis do que as técnicas baseadas em *wrapper*. Além disso, por não precisar retreinar um algoritmo de AM para cada novo conjunto de atributos, as técnicas embutidas em geral são mais rápidas (Guyon e Elisseeff, 2003).

Outra forma de classificação está relacionada à seleção dos atributos ser feita de forma individual ou coletiva. No primeiro caso, os atributos são ordenados de acordo com a sua relevância para discriminar os objetos das diferentes classes. Boa parte das técnicas de ordenação pode ser empregada apenas em problemas de classificação binária (problemas com duas classes). A segunda alternativa seleciona um subconjunto dos atributos originais que melhor separe os exemplos das diferentes classes.

Finalmente, algumas técnicas de seleção de atributos utilizam a informação sobre a classe, sendo denominadas dependentes da classe, e outras, por não utilizar essa informação, são denominadas independentes da classe.

A seleção de atributos tanto por ordenação quanto por seleção de subconjunto, e utilizando ou não informação sobre a classe, pode ser feita com as abordagens filtro e *wrapper*. A abordagem embutida trabalha com seleção de subconjuntos. A seguir, as técnicas de ordenação e seleção por subconjunto são discutidas em mais detalhes.

3.7.3 Técnicas de Ordenação

Ordenação de atributos pode ser vista como uma forma simples de seleção, em que os atributos são ordenados de acordo com sua relevância para um dado critério, por exemplo, classificação dos objetos nas diferentes classes. Em problemas de classificação, os atributos no topo da ordenação são selecionados para utilização pelo classificador. Grande parte das técnicas existentes foi desenvolvida para a redução do número de genes em problemas de análise de expressão gênica por meio de microarranjos. Nesses problemas, cada gene pode ser visto como um atributo e cada objeto possui milhares de genes ou atributos.

A escolha da melhor abordagem para seleção de atributos depende das propriedades a serem medidas (Dopazo et al., 2001). Várias das abordagens descritas na literatura são técnicas para ordenação (*ranking*), também denominadas escores ou medidas. Neste texto, será adotado o termo medidas. Algumas dessas medidas avaliam similaridade (medidas de correlação), e outras avaliam a diferença (medidas de distância) entre vetores. As medidas podem ou não considerar a informação sobre a classe.

As medidas podem ainda ser divididas em paramétricas e não paramétricas. As medidas paramétricas fazem suposições sobre a distribuição estatística das medidas dentro de

¹O viés refere-se à tendência dos algoritmos em sistematicamente favorecer determinadas situações, dentre as muitas disponíveis. Em AM, um exemplo pode ser a preferência de certos algoritmos em construir classificadores mais ou menos complexos. Na seleção de atributos, um algoritmo pode preferir atributos que possuam certas propriedades.

cada grupo ou classe. As medidas não paramétricas não fazem essa suposição e são mais robustas (Ben-Dor et al., 2002).

As medidas não paramétricas geralmente especificam uma hipótese em termos de distribuições populacionais, em vez de parâmetros como média e desvio padrão. Essas medidas conseguem detectar diferenças entre populações quase tão bem quanto as medidas paramétricas quando suposições como normalidade precisam ser satisfeitas. Quando essas suposições não são satisfeitas, medidas não paramétricas podem ser, e frequentemente são, mais poderosas que as paramétricas para detectar diferenças entre populações.

Deve ser observado que, no caso da ordenação dependente de classe, o primeiro atributo é aquele que melhor discrimina os objetos das diferentes classes, o segundo é o segundo melhor atributo para essa discriminação e assim por diante. Na seleção do subconjunto, os atributos que fazem parte do subconjunto selecionado não necessariamente estariam no topo da lista se uma técnica de ordenação fosse utilizada. O que importa nesse caso é como os atributos selecionados atuam de forma coletiva, em conjunto. Isso ocorre, por exemplo, porque dois atributos situados próximos na lista ordenada podem estar correlacionados.

3.7.4 Técnicas de Seleção de Subconjunto

A seleção de um subconjunto de atributos é um processo computacionalmente mais custoso que a ordenação dos atributos. Essa desvantagem aumenta com o crescimento do número de atributos. A seleção de subconjuntos de atributos pode se tornar intratável quando o número de atributos é muito grande. Uma alternativa geralmente utilizada é primeiro ordenar os d atributos originais para em seguida selecionar um subconjunto a partir dos $d_{reduzido}$ atributos mais bem ordenados.

Por incorporar o viés do classificador, as técnicas baseadas em *wrapper* em geral conseguem obter um conjunto de atributos que leva a um melhor desempenho posterior do modelo.

A seleção de um subconjunto de atributos pode ser vista como um problema de busca. Cada ponto no espaço de busca pode ser visto como um possível subconjunto de atributos. Por essa ótica, um método de seleção deve definir (Blum e Langley, 1997):

- Qual(is) será(ão) o(s) ponto(s) de partida ou a direção em que a busca será realizada;
- Que estratégia de busca será adotada;
- Qual o critério a ser empregado para avaliação dos subconjuntos gerados;
- Qual será o critério de parada.

Os critérios de avaliação já foram discutidos anteriormente, que são as abordagens filtro, *wrapper* ou embutida. A seguir serão discutidos os outros três aspectos.

Com relação ao ponto de partida e direção da busca, quatro diferentes alternativas podem ser adotadas:

- Geração para trás (*backward generation*), que começa com todos os atributos e remove um por vez.
- Geração para a frente (*forward generation*), que começa sem nenhum atributo e inclui um atributo por vez.

- Geração bidirecional (*bidirectional generation*), em que a busca pode começar em qualquer ponto e atributos podem ser adicionados e removidos.
- Geração estocástica (*random generation*), quando o ponto de partida da busca e atributos a serem removidos ou adicionados são decididos de forma estocástica.

Com relação à estratégia de busca a ser adotada, as principais abordagens são:

- Busca completa (exponencial ou exaustiva), que avalia todos os possíveis subconjuntos.
- Busca heurística (sequencial), que utiliza regras e métodos para conduzir a busca e que não garante que uma solução ótima seja encontrada.
- Busca não determinística, que está relacionada com a geração estocástica. Nesse caso, embora uma boa solução pode ser encontrada antes do final da busca, não é possível garantir que será encontrada a melhor solução possível.

Finalmente, com relação ao final da busca, pode ser conduzida uma busca exaustiva, em que a busca é encerrada quando todos os subconjuntos forem testados, ou pode ser adotado um critério de parada, que define quando terminar a busca pelo melhor subconjunto de atributos. Esse critério pode ser, por exemplo, um número máximo de alternativas testadas, um número de atributos a serem selecionados sem degradação do desempenho do classificador ou o tempo de processamento.

É importante observar que técnicas de seleção de atributos podem ser utilizadas também em problemas de outra natureza, como problemas de agrupamento de dados, por exemplo. Entretanto, como em algumas abordagens a informação sobre a classe não é utilizada, outra informação que estime a qualidade do agrupamento deve ser empregada.

3.8 Considerações Finais

O processo de aprendizado de algoritmos de AM pode ser facilitado e o desempenho desses algoritmos melhorado se técnicas de pré-processamento forem previamente aplicadas ao conjunto de dados. Isso ocorre porque essas técnicas podem eliminar ou reduzir problemas presentes nos dados.

Neste capítulo foram apresentadas diversas técnicas utilizadas para pré-processamento de dados. Começando com a eliminação manual de atributos que se mostrarem claramente desnecessários para o uso dos dados por algoritmos de AM, discutiu-se a integração de dados provenientes de diferentes fontes, a utilização de técnicas de amostragem para selecionar subconjuntos representativos de dados, como lidar com dados desbalanceados e como operações de limpeza de dados podem tratar de dados incompletos, inconsistentes, redundantes e com ruídos. Em seguida, foram apresentadas alternativas para transformação de dados, para facilitar seu uso por diferentes algoritmos de AM, como a conversão de dados de um tipo para outro e a normalização de dados numéricos. Por fim, foi observada a importância da redução de dimensionalidade dos dados, principalmente quando o número de atributos é muito elevado.

Os problemas mencionados são muito frequentes em conjuntos de dados reais, e o tratamento deles em geral leva a um melhor desempenho do algoritmo de AM utilizado. Por isso, a etapa de pré-processamento é de grande importância em aplicações reais de AM.

Parte II

Modelos Preditivos

Introdução aos Modelos Preditivos

Um algoritmo de AM preditivo é uma função que, dado um conjunto de exemplos rotulados, constrói um estimador. O rótulo ou etiqueta toma valores num domínio conhecido. Se esse domínio for um conjunto de valores nominais, tem-se um problema de classificação, também conhecido como aprendizado de conceitos, e o estimador gerado é um classificador. Se o domínio for um conjunto infinito e ordenado de valores, tem-se um problema de regressão, que induz um regressor. Um classificador (ou regressor), por sua vez também é uma função, que, dado um exemplo não rotulado, atribui esse exemplo a uma das possíveis classes (ou a um valor real) (Dietterich, 1998).

Uma definição formal seria, dado um conjunto de observações de pares $\mathbf{D} = \{(\mathbf{x}_i, f(\mathbf{x}_i)), i = 1, \dots, n\}$, em que f representa uma função desconhecida, um algoritmo de AM preditivo aprende uma aproximação \hat{f} da função desconhecida f . Essa função aproximada, \hat{f} , permite estimar o valor de f para novas observações de \mathbf{x} .

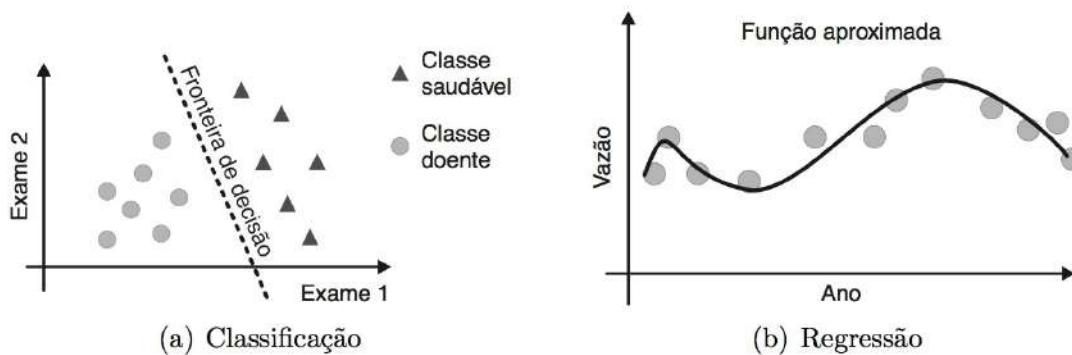
De acordo com a natureza de f , é comum distinguir duas possíveis situações:

- Classificação: $y_i = f(\mathbf{x}_i) \in \{c_1, \dots, c_m\}$, ou seja, $f(\mathbf{x}_i)$ assume valores em um conjunto discreto, não ordenado;
- Regressão: $y_i = f(\mathbf{x}_i) \in \mathbb{R}$, ou seja, $f(\mathbf{x}_i)$ assume valores em um conjunto infinito e ordenado de valores.

A Figura 3.1(a) ilustra um cenário com duas classes cujos exemplos possuem apenas dois atributos de entrada, que são os resultados de dois exames. A meta é encontrar uma fronteira de decisão que separe os exemplos de uma classe dos exemplos da outra. Se os exemplos de uma classe forem linearmente separáveis dos exemplos da outra classe, como os exemplos possuem dois atributos, a fronteira de decisão pode ser uma reta. Se os exemplos não forem linearmente separáveis, uma combinação de retas é necessária. Se os exemplos possuem mais de dois atributos de entrada, em vez de retas, são utilizados hiperplanos de separação.

Diferentes algoritmos de AM podem encontrar diferentes fronteiras de decisão. Além disso, diferenças no conjunto de treinamento, variações na ordem de apresentação dos exemplos durante o treinamento e processos internos estocásticos podem fazer com que um mesmo algoritmo de AM encontre fronteiras diferentes a cada novo treinamento.

A Figura 3.1(b) ilustra um caso de regressão em que a meta é aprender uma função que relate um ano à vazão de água de um dado rio nesse ano. Nesse caso é utilizado apenas um atributo de entrada. A ligação dos pontos formados pelos valores dia e vazão gera uma curva. Espera-se que essa curva se aproxime da curva verdadeira da função que, dado o valor do ano, retorna o valor correto da vazão. Se mais de um atributo de entrada for utilizado, a regressão vai definir superfícies com mais que duas dimensões.

**Figura 3.1** Gráfico ilustrativo das tarefas.

Nas Tabelas 3.12 e 3.13 são apresentados exemplos de dois conjuntos de dados. O primeiro conjunto (Tabela 3.12) apresenta exemplos de dados para um problema de classificação, o problema de classificação de dados *iris*, abordado no Capítulo 2. Nele, o atributo alvo, que aparece na última coluna, assume valores discretos e não ordenados. O segundo conjunto, apresentado na Tabela 3.13, se refere a um problema de regressão. O atributo alvo, que aparece na última coluna, assume valores contínuos e ordenados.

Tabela 3.12 Exemplo de conjunto de dados para problema de classificação

Tamanho (P)	Largura (P)	Tamanho (S)	Largura (S)	Espécie
5,1	3,5	1,4	0,2	Setosa
4,9	3,0	1,4	0,2	Setosa
7,0	3,2	4,7	1,4	Versicolor
6,4	3,2	4,5	1,5	Versicolor
6,3	3,3	6,0	2,5	Virgínea
5,8	2,7	5,1	1,9	Virgínea

Tabela 3.13 Exemplo de conjunto de dados para problema de regressão

Fertilidade	Agricultura	Educação	Renda	Mortalidade
80,2	17,0	12	9,9	22,2
83,1	45,1	9	84,8	22,2
92,5	39,7	5	93,4	20,2
85,8	36,5	7	33,7	20,3
76,9	43,5	15	5,2	20,6

Em ambos os conjuntos, cada linha corresponde a um objeto ou observação. A última coluna apresenta o valor da função f para essa observação, que corresponde ao atributo alvo. Essa coluna é também designada por variável dependente ou objetivo. No caso particular de problemas de classificação, ela é designada *classe*. As colunas restantes são designadas por atributos de entrada, atributos preditivos ou variáveis independentes. Os atributos serão utilizados como entrada do algoritmo na realização da predição.

O primeiro conjunto de dados, denominado *iris*, é bastante empregado para ilustrar o uso de técnicas de AM na solução de problemas de classificação. Nele, a partir de algumas

características de flores de uma planta denominada íris, no caso tamanhos e larguras das suas pétalas e sépalas, é possível separá-las em uma dentre três classes, que designam diferentes espécies dessa planta: setosa, versicolor e virgínica. Um dos motivos para esse conjunto de dados ser largamente empregado na prática está no fato de se conhecer de antemão que uma das classes, setosa, encontra-se bem separada das duas outras. No caso das classes versicolor e virgínica, por sua vez, a separação existe, mas não é tão evidente e de fato há vários objetos dessas classes próximos entre si.

O segundo conjunto de dados na figura é denominado **swiss**. Para esse conjunto, deseja-se relacionar estatísticas de uma população, tais como nível de educação, taxa de fertilidade, entre outras, à previsão da sua taxa de mortalidade. Em ambos os casos, o objetivo do aprendizado preditivo é aprender uma função $\hat{f}(\mathbf{x})$ que mapeia as variáveis independentes, os atributos de entrada, na variável objetivo, o atributo alvo.

A função \hat{f} pode assumir diferentes formas. Por exemplo, combinações lineares ou não lineares dos atributos de entrada, funções por ramos, expressões lógicas etc., que são estudadas nos capítulos seguintes. É importante notar que \hat{f} é uma aproximação de uma função desconhecida f , ou seja podem existir \mathbf{x} para os quais $\hat{f}(\mathbf{x}) \neq f(\mathbf{x})$. A estimativa da qualidade de um modelo preditivo é dada pelo custo associado às previsões do modelo. Dependendo do tipo de problema, classificação ou regressão, diferentes funções de custo são utilizadas. No caso de problemas de classificação, é usual utilizar a função de custo $0 - 1$: o custo de uma previsão incorreta ($\hat{f}(\mathbf{x}) \neq f(\mathbf{x})$) é 1, e o custo de uma previsão correta ($\hat{f}(\mathbf{x}) = f(\mathbf{x})$) é 0. Em problemas de regressão, é usual utilizar o erro quadrático médio. As funções de custo e as metodologias para as estimar são estudadas no Capítulo 9.

Considere, para simplificar, um problema de classificação com duas classes. Assuma que é conhecida a função densidade de probabilidade (pdf, do inglês *probability density function*) para cada classe. A Figura 3.2 mostra as duas pdfs (uma para cada classe) em um problema definido por um único atributo de entrada. O melhor classificador possível divide o domínio da variável no ponto de interseção das duas pdfs, ou seja, classifica um objeto na classe com maior função densidade de probabilidade. A área sombreada representa o erro cometido por esse classificador. É intuitivo que esse classificador tem erro mínimo: movendo em qualquer direção na superfície de decisão, o erro sempre cresce. O erro desse classificador é conhecido como *erro de Bayes ótimo* e é um mínimo teórico da capacidade de generalização de qualquer classificador. No Capítulo 9 são indicadas metodologias para estimar a qualidade de \hat{f} , ou seja, quanto aproximado é \hat{f} de f .

Na literatura de aprendizado automático é usual distinguir *modelos generativos* de *modelos discriminativos*. Os modelos generativos estimam uma função densidade de probabilidade condicionada à classe c_i . Um exemplo de teste é classificado na classe que maximiza a probabilidade *a posteriori*. São designados generativos porque funções densidade de probabilidade condicionada podem ser utilizadas para gerar dados sintéticos. O *naive Bayes* (Duda et al., 2001) é um exemplo clássico desse grupo de modelos. Os modelos discriminativos não modelam a distribuição de probabilidade, mas calculam diretamente as probabilidades *a posteriori*. Modelos tradicionais incluem árvores de decisão (Quinlan, 1993), redes neurais (Mitchell, 1997) e vizinhos mais próximos (Aha et al., 1991).

Nos capítulos seguintes serão descritos os principais métodos preditivos de AM. Para isso, eles são organizados em métodos baseados em distâncias, métodos probabilísticos, métodos baseados em procura e métodos baseados em otimização. Em um capítulo seguinte, serão apresentadas formas de combinar métodos de classificação em métodos preditivos.

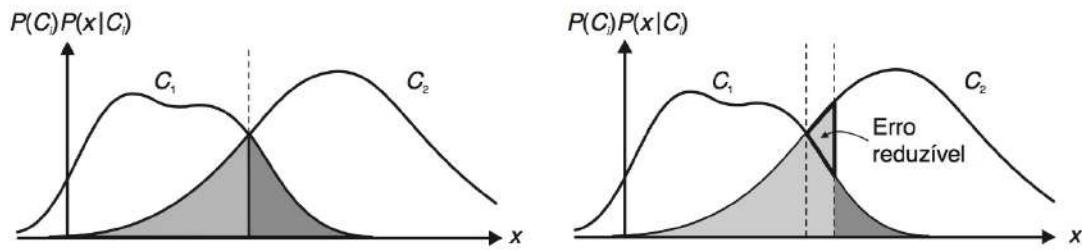


Figura 3.2 A figura da esquerda representa a superfície de decisão ótima. A área cinzenta corresponde à probabilidade de erro ao decidir por C_1 quando a classe correta é C_2 ; a área cinza-escura é o erro oposto. Se movemos a superfície de decisão, como se vê na figura da direita, o erro sempre aumenta.

tivos múltiplos. Finalmente, serão discutidas formas de planejar experimentos preditivos e de analisar os resultados obtidos.

Capítulo 4

Métodos Baseados em Distâncias

4.1 Introdução

Neste capítulo serão apresentadas técnicas de AM que consideram a proximidade entre os dados na realização de previsões. A hipótese base é que dados similares tendem a estar concentrados em uma mesma região no espaço de entrada. De maneira alternativa, dados que não são similares estarão distantes entre si.

A título ilustrativo, na Figura 4.1 é apresentada a projeção em duas dimensões do conjunto de dados *iris*. Os objetos da mesma classe estão representados por uma mesma cor. É fácil observar visualmente a existência de áreas densas com objetos pertencentes à mesma classe, evidenciando que a distância entre os objetos está relacionada à definição de suas classes.

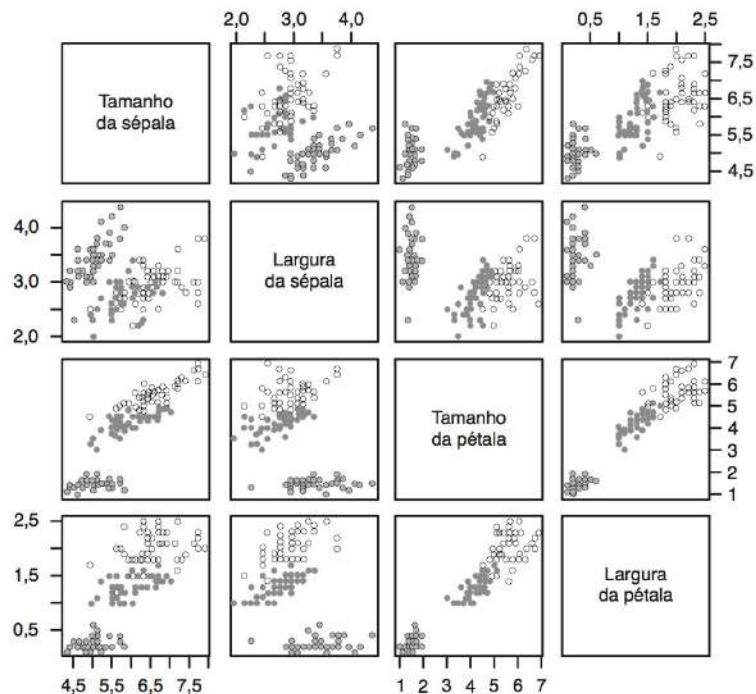


Figura 4.1 Projeção sobre o plano definido por dois atributos dos objetos do conjunto de dados *iris*.

Um método baseado em distância utilizado com frequência é o algoritmo dos vizinhos mais próximos. Esse algoritmo é o mais simples de todos os algoritmos de Aprendizado de Máquina. A intuição por trás do algoritmo é:

Objetos relacionados ao mesmo conceito são semelhantes entre si.

O algoritmo dos vizinhos mais próximos classifica um novo objeto com base nos exemplos do conjunto de treinamento que são próximos a ele. É um algoritmo preguiçoso (*lazy*), porque não aprende um modelo compacto para os dados, apenas memoriza os objetos de treinamento. Uma vantagem desse algoritmo é que ele pode ser utilizado tanto em problemas de classificação como em problemas de regressão de maneira direta, sem necessidade de alterações significativas. Para algumas técnicas, como as Máquinas de Vetores de Suporte, essa generalização envolve grandes alterações no algoritmo de aprendizado, conforme será discutido no Capítulo 7.

Neste capítulo serão apresentados alguns métodos de aprendizado baseados em distância. Na Seção 4.2 apresentamos a descrição do funcionamento do algoritmo dos vizinhos mais próximos utilizando o caso mais simples, apenas um vizinho. A Seção 4.3 apresenta o algoritmo k -NN, que é analisado na Seção 4.4. Variações recentemente desenvolvidas para o algoritmo k -NN são objeto da Seção 4.5. A Seção 4.6 apresenta uma metodologia de AM que utiliza distância para recuperar a solução de problemas passados que sejam similares ao problema que se deseja resolver, conhecida como raciocínio baseado em casos. A última Seção apresenta as considerações finais para os temas vistos no capítulo.

4.2 O Algoritmo do 1-Vizinho Mais Próximo

O algoritmo dos vizinhos mais próximos tem variações definidas pelo número de vizinhos considerados. Dessa variações, a mais simples é o algoritmo 1-vizinho mais próximo (1-NN, do inglês *1-Nearest Neighbour*).

Nesse algoritmo, cada objeto representa um ponto em um espaço definido pelos atributos, denominado espaço de entrada, como ilustrado no Capítulo 1. Definindo uma métrica nesse espaço, é possível calcular as distâncias entre cada dois pontos. A métrica mais usual para isso é a distância euclidiana, dada pela Equação 4.1, em que \mathbf{x}_i e \mathbf{x}_j são dois objetos representados por vetores no espaço \mathbb{R}^d , e x_i^l e x_j^l são elementos desses vetores, que correspondem aos valores da coordenada l (atributos). As propriedades da distância euclidiana, bem como outras medidas de distância ou similaridade, serão detalhadas no Capítulo 11.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{l=1}^d (x_i^l - x_j^l)^2} \quad (4.1)$$

Como dito anteriormente, o algoritmo 1-NN é muito simples, e está ilustrado no Algoritmo 4.1. Na fase de treinamento, o algoritmo memoriza os **exemplos rotulados** do conjunto de treinamento. Para classificar um exemplo não rotulado, ou seja, cuja classe não é conhecida, é calculada a distância entre o vetor de valores de atributos e cada exemplo rotulado em memória. O rótulo da classe associado ao exemplo de treinamento mais próximo do exemplo de teste é utilizado para classificar o novo exemplo.

A Figura 4.2 apresenta um exemplo ilustrativo de aplicação do algoritmo 1-NN em um problema de duas classes. Nesse exemplo, considera-se um conjunto de dados cujos objetos são indivíduos que podem ser classificados em saudáveis ou doentes, e o espaço de entrada é definido por dois atributos que representam o resultado de dois exames. O ponto representado por “?” é o ponto de teste, ou seja, o indivíduo a ser classificado. Todos

Algoritmo 4.1 Versão básica do algoritmo 1-vizinho mais próximo

Entrada: Um conjunto de treinamento: $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$

Um objeto de teste a ser classificado: $t = \{\mathbf{x}_t, y_t = ?\}$

A função de distância entre objetos: $d(\mathbf{x}_a, \mathbf{x}_b)$

Saída: y_t : Classe atribuída ao exemplo t

```

1  $d_{min} \leftarrow +\infty$ 
2 para cada  $i \in 1, \dots, n$  faça
3   se  $d(\mathbf{x}_i, \mathbf{x}_t) < d_{min}$  então
4      $d_{min} \leftarrow d(\mathbf{x}_i, \mathbf{x}_t)$ 
5      $idx \leftarrow i$ 
6   fim
7 fim
8  $y_t = y_{idx}$ 
9 Retorna:  $y_t$ 
```

os outros pontos são objetos em que a classe é conhecida: saudável, representada por um triângulo ou doente, representada por um círculo. No espaço definido pelos atributos, e usando a distância euclidiana, o objeto de treinamento mais próximo do objeto de teste pertence à classe doente, que é então atribuída ao objeto de teste.

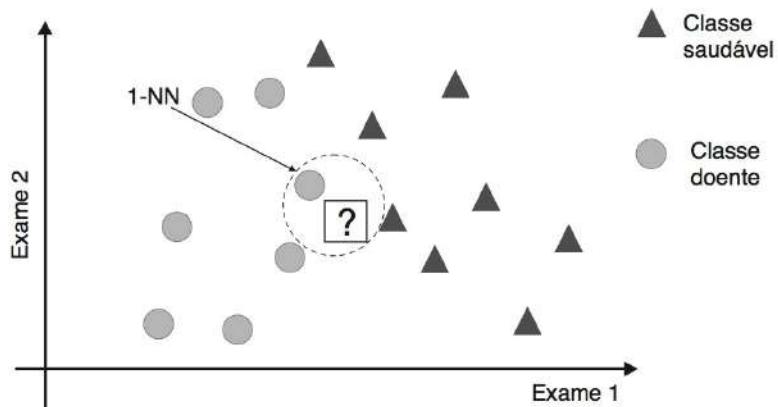


Figura 4.2 Exemplo ilustrativo do algoritmo 1-NN.

4.2.1 Superfícies de Decisão

Apesar da sua simplicidade, as superfícies de decisão desenhadas pelo algoritmo do 1-NN são muito complexas: são poliedros convexos com centro em cada objeto do conjunto de treinamento. Todos os pontos no interior de um poliedro pertencem à classe do objeto do conjunto de treinamento que define o centro desse poliedro. O conjunto desses poliedros é designado *diagrama de Voronoi*. A Figura 4.3 mostra a construção de um exemplo de diagrama de Voronoi.

Para entender a construção do diagrama de Voronoi, seja \mathbf{D} um conjunto de treinamento. A célula de Voronoi em torno de um ponto $\mathbf{x} \in \mathbf{D}$ é definida como o conjunto de pontos cuja distância a \mathbf{x} é menor que a distância a qualquer outro ponto de \mathbf{D} . A

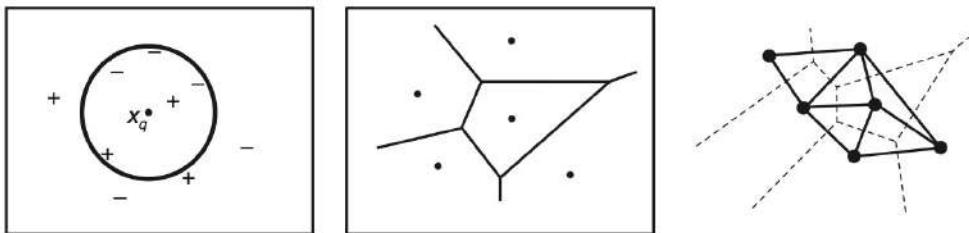


Figura 4.3 Superfície de decisão: diagrama de Voronoi.

superfície de decisão desenhada pelo algoritmo 1-NN é um conjunto de poliedros convexos contendo cada um dos objetos de treinamento.

4.2.2 Distâncias

Métodos baseados em distância, como o algoritmo dos vizinhos mais próximos, têm seu desempenho afetado pela medida ou função de distância utilizada. Um problema da medida de distância apresentada está em pressupor que os dados correspondem a pontos no espaço d -dimensional (\mathbb{R}^d), ou seja, que seus atributos são numéricos (contínuos). Contudo, diversos problemas possuem dados com atributos qualitativos. Outro aspecto que deve ser observado no cálculo da distância entre objetos é a escala utilizada para os valores dos atributos. Por exemplo, qual o efeito na função distância da representação de um atributo em *cm* ou *Km*? As medidas de distância são afetadas pela escala dos atributos. Para minimizar esse efeito, os atributos são usualmente normalizados.

4.3 O Algoritmo k -NN

Uma extensão imediata ao algoritmo 1-NN é considerar, em vez de 1 vizinho mais próximo, os k objetos do conjunto de treinamento mais próximos do ponto de teste \mathbf{x}_t , em que k é um parâmetro do algoritmo. Quando o valor de k é maior que 1, para cada ponto de teste, são obtidos k vizinhos. Cada vizinho vota em uma classe. As previsões dos diferentes vizinhos são agregadas de forma a classificar o ponto de teste. Essa agregação é efetuada de forma diferente em problemas de classificação e de regressão.

Em problemas de classificação, em que a classe toma valores em um conjunto discreto, cada vizinho vota em uma classe. O objeto de teste é classificado na classe mais votada. Formalmente, esse processo é equivalente a: $\hat{f}(\mathbf{x}_t) \leftarrow \text{moda}(f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_k))$, o que é justificado porque *a constante que minimiza a função de custo 0-1 é a moda*.

Em problemas de regressão, podem ser utilizadas duas estratégias, dependendo da função de custo usada. Se a função de custo for minimizar o erro quadrático, a média dos valores obtidos para cada um dos k vizinhos deve ser utilizada, o que pode ser formalmente definido como: $\hat{f}(\mathbf{x}_t) \leftarrow \text{média}(f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_k))$. Se a função de custo a ser minimizada for o desvio absoluto, em vez da média, deve ser utilizada a mediana. Nesse caso, a função passa a ser: $\hat{f}(\mathbf{x}_t) \leftarrow \text{mediana}(f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_k))$. A justificativa para esses procedimentos é porque a média é a constante que minimiza o erro quadrático, enquanto a constante que minimiza o desvio absoluto é a mediana.

A seguir será apresentado um exemplo ilustrativo simples do uso do algoritmo dos vizinhos mais próximos para diferentes valores de k .

Considere o problema da Figura 4.4 (trata-se do mesmo conjunto de dados apresentado anteriormente na Figura 4.2). Para $k = 3$, o objeto de teste seria classificado como pertencendo à classe “doente”, enquanto para $k = 5$ o objeto de teste seria classificado como pertencendo à classe “saudável”.

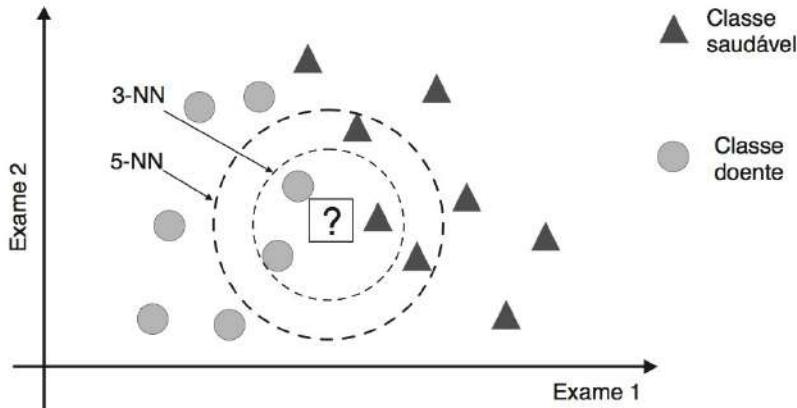


Figura 4.4 Impacto do valor de k no algoritmo k -NN.

A escolha do valor de k mais apropriado para um problema de decisão específico pode não ser trivial. O valor de k é definido pelo usuário. Frequentemente, o valor de k é pequeno e ímpar: $k = 3, 5, \dots$. Em problemas de classificação, não é usual utilizar $k = 2$ ou valores pares, para evitar empates.

Duas estratégias referidas na literatura consistem em:

- Estimar k por validação cruzada (ver Capítulo 9).
- Associar um peso à contribuição de cada vizinho.

Nesse último caso, a contribuição de cada um dos k vizinhos é pesada de forma inversamente proporcional à distância ao ponto de teste. Dessa forma, é possível utilizar $k = n$ (todos os objetos de treinamento).

- Em problemas de classificação:

– Moda ponderada: $y_t = \arg \max_{c \in Y} \sum_{i=1}^k w_i I(c, y_i)$, com $w_i = \frac{1}{d(\mathbf{x}_t, \mathbf{x}_i)}$ e $I(a, b)$, é uma função que retorna 1 se e só se $a = b$;

- Em problemas de regressão:

– Média ponderada: $y_t = \frac{\sum_{i=1}^k w_i y_i}{\sum w_i}$ com $w_i = \frac{1}{d(\mathbf{x}_t, \mathbf{x}_i)}$

Em que y_i é a classe do exemplo x_i , w_i é o peso associado ao exemplo x_i e c é a classe com maior moda ponderada.

4.4 Análise do Algoritmo

Nesta seção serão analisados os principais aspectos do algoritmo k -NN, salientando seus aspectos positivos e negativos.

4.4.1 Aspectos Positivos

O algoritmo k -NN representa um dos paradigmas mais conhecidos do aprendizado indutivo: *Objetos com características semelhantes pertencem ao mesmo grupo.* O k -NN é um algoritmo baseado em memória, ou um algoritmo preguiçoso, porque toda a computação é adiada até à fase de classificação, já que o processo de aprendizado consiste apenas em memorizar os objetos.

- O algoritmo de treinamento é simples (armazenar objetos apenas).
- O k -NN constrói aproximações locais da função objetivo, diferentes para cada novo dado a ser classificado. Essa característica pode ser vantajosa quando a função objetivo é muito complexa, mas ainda pode ser descrita por uma coleção de aproximações locais de menor complexidade (Mitchell, 1997).
- Ele é aplicável mesmo em problemas complexos.
- O algoritmo é naturalmente incremental: quando novos exemplos de treinamento estão disponíveis, basta armazená-los na memória.

Um dos aspectos mais relevantes desse algoritmo está relacionado com o seu comportamento no limite. Se forem considerados:

- e : erro do classificador Bayes ótimo;
- $e_{1nn}(\mathbf{D})$: erro do 1-NN;
- $e_{knn}(\mathbf{D})$: erro do k -NN.

Os seguintes teoremas foram provados:

- $\lim_{n \rightarrow \infty} e_{1nn}(\mathbf{D}) \leq 2 \times e$;
- $\lim_{n \rightarrow \infty, k \rightarrow n} e_{knn}(\mathbf{D}) = e$.

Ou seja, para um número infinito de objetos, o erro do 1-NN é majorado pelo dobro do erro do Bayes ótimo, e o erro do k -NN tende para o erro do Bayes ótimo.

4.4.2 Aspectos Negativos

Também por ser um algoritmo *lazy*, o algoritmo dos vizinhos mais próximos não obtém uma representação compacta dos objetos. De fato, não se tem um modelo explícito a partir dos dados. A fase de treinamento requer pouco esforço computacional. No entanto, classificar um objeto de teste requer calcular a distância desse objeto a todos os objetos de treinamento. Assim, a predição pode ser custosa, e para um conjunto grande de objetos de treinamento esse processo pode ser demorado. Como todos os algoritmos baseados em distâncias, ele é afetado pela presença de atributos redundantes e de atributos irrelevantes.

Outro problema do k -NN está relacionado à dimensionalidade dos exemplos. O espaço definido pelos atributos de um problema cresce exponencialmente com o número de atributos, ou seja, o número de atributos define o número de dimensões do espaço. O ponto que está mais perto de outro pode estar muito distante em problemas de alta dimensionalidade. Para ilustrar as dificuldades levantadas pela dimensionalidade de um problema, considere 100 pontos com distribuição uniforme:

- Em um quadrado cujo lado mede 1 unidade
- Em um cubo cujo lado mede 1 unidade
- ...

Calculando a distância média entre dois pontos, obtemos:

Núm. dimensões	Distância média
2	0,494
3	0,647
4	0,772
5	0,875
...	
10	1,280

O que se observa é um aumento da distância média entre dois pontos quaisquer. Ou seja, a densidade diminui e o conjunto de dados fica esparsa, rarefeito. Para 10 dimensões, a distância média é maior que o tamanho do lado do hipercubo! Quando a dimensão aumenta linearmente, para manter a mesma densidade de pontos, é necessário aumentar de forma exponencial o número de pontos. Beyer et al. (1999) mostram que, sob um amplo conjunto de condições (muito mais amplo do que dimensões independentes e identicamente distribuídas), com o aumento da dimensionalidade, a distância ao vizinho mais próximo aproxima-se da distância ao vizinho mais afastado. A dimensionalidade de um problema pode afetar de forma negativa o desempenho dos algoritmos baseados em distâncias. Uma das formas para reduzir o impacto da dimensionalidade de um problema consiste em selecionar um subconjunto de atributos relevantes para o problema tratado. No Capítulo 3 são apresentados vários algoritmos para seleção de atributos.

4.5 Desenvolvimentos

Uma grande parte dos trabalhos de pesquisa relacionados ao algoritmo k -NN investiga a redução do espaço do problema. Já foi salientado que o k -NN é um algoritmo lento no processo de classificação de exemplos de teste. Uma das possibilidades para minimizar esse inconveniente consiste em obter um subconjunto de exemplos representativos. Por exemplo, eliminando objetos redundantes, ou eliminando objetos em que todos os vizinhos são da mesma classe. Outra possibilidade consiste em eliminar objetos com ruído, por exemplo eliminando objetos em que todos os vizinhos são de outra classe.

Aha et al. (1991) apresentaram vários algoritmos para selecionar os objetos mais relevantes, designados por protótipos, para o problema de aprendizado, de forma a reter em memória apenas esses objetos. As versões do algoritmo Edit k -NN para eliminação sequencial, Algoritmo 4.2, e inserção sequencial, Algoritmo 4.3, são dois exemplos de algoritmos que armazenam apenas protótipos. No primeiro caso (Algoritmo 4.2), o algoritmo começa com todos os objetos e vai descartando os objetos que são corretamente classificados pelo conjunto de protótipos atual. No segundo (Algoritmo 4.3), o conjunto de protótipos é inicialmente vazio. Os objetos que são incorretamente classificados pelo conjunto de protótipos são acrescentados a esse conjunto.

Algoritmo 4.2 Algoritmo para *Edit k-NN*: eliminação sequencial

Entrada: Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
Saída: Um conjunto de treinamento $\mathbf{D}' = \{(\mathbf{x}_i, y_i), i = 1, \dots, m; m < n\}$

- 1 **para cada exemplo** $(\mathbf{x}_i, y_i) \in \mathbf{D}$ **faça**
- 2 **se** (\mathbf{x}_i, y_i) é corretamente classificado por $\mathbf{D} \setminus \{(\mathbf{x}_i, y_i)\}$ **então**
- 3 /* Remove (\mathbf{x}_i, y_i) de \mathbf{D} */ ;
- 4 $\mathbf{D} \leftarrow \mathbf{D} \setminus \{(\mathbf{x}_i, y_i)\}$;
- 5 **fim**
- 6 **fim**
- 7 **Retorna:** \mathbf{D} ;

Algoritmo 4.3 Algoritmo para *Edit k-NN*: inserção sequencial

Entrada: Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
Saída: Um conjunto de treinamento $\mathbf{D}' = \{(\mathbf{x}_i, y_i), i = 1, \dots, m; m < n\}$

- 1 $\mathbf{D}' \leftarrow \{\}$;
- 2 **para cada exemplo** $(\mathbf{x}_i, y_i) \in \mathbf{D}$ **faça**
- 3 **se** (\mathbf{x}_i, y_i) é incorretamente classificado por \mathbf{D}' **então**
- 4 /* Acrescenta (\mathbf{x}_i, y_i) a \mathbf{D}' */ ;
- 5 $\mathbf{D}' \leftarrow \mathbf{D}' \cup \{(\mathbf{x}_i, y_i)\}$;
- 6 **fim**
- 7 **fim**
- 8 **Retorna:** \mathbf{D}'

A busca linear realizada pelo algoritmo *k-NN* é ineficiente para grandes conjuntos de dados. Estruturas de indexação mais sofisticadas podem permitir uma busca mais eficiente, acelerando a classificação de novos objetos. Uma alternativa para isso é utilização de árvores de busca binária multidimensionais (*kd-trees*, do inglês *k-dimensional trees*) (Bentley, 1975) como estrutura de dados.

Uma *kd-tree* partitiona um espaço de busca *k*-dimensional utilizando os *k* eixos do sistema de coordenadas. Ela faz isso definindo, recursivamente, o particionamento do espaço dos dados em subespaços disjuntos. No contexto de AM, cada dimensão representa um atributo preditor.

Seja um conjunto de dados formado por *n* objetos. Uma *kd-tree* representando esse conjunto possui *n* nós, um para cada objeto. A cada nó é associado um atributo discriminador, que é um inteiro entre 0 e *k* – 1. Todos os nós em um mesmo nível na árvore possuem o mesmo discriminador. O atributo discriminador indica que atributo está sendo utilizado pelo nó para partitionar os objetos.

A Figura 4.5 ilustra uma *2d-tree* representando um conjunto formado por quatro objetos, x_1 , x_2 , x_3 e x_4 . O nó raiz da árvore, armazena o objeto x_1 . Como o atributo discriminador associado ao nó raiz atual é o atributo 0, o nó raiz da subárvore esquerda do nó raiz atual armazena o objeto com valor para o atributo 0 menor que o valor do atributo 0 do nó raiz, que maximiza a separação entre os objetos que satisfazem essa

restrição. Nesse caso, foi selecionado o objeto x_2 . Princípio semelhante é utilizado para escolher o objeto que será a raiz da subárvore à direita do nó raiz atual. Esse procedimento é seguido de forma recursiva até que cada objeto seja representado por um nó da árvore. A árvore final vai facilitar a busca de objetos no espaço k -dimensional.

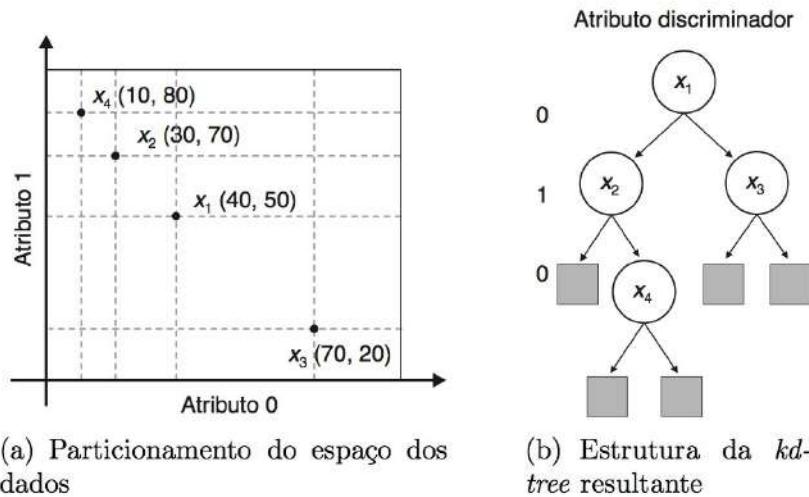


Figura 4.5 Os pontos mostrados em 4.5(a) identificam objetos armazenados como nós internos na *kd-tree* e são utilizados para partitionar o espaço de busca.

4.6 Raciocínio Baseado em Casos

Raciocínio baseado em casos (RBC) é uma metodologia de AM para a resolução de problemas fundamentada na utilização de experiências passadas. Assim, um sistema de RBC procura resolver um novo problema apresentado por meio da recuperação de problemas similares previamente solucionados de uma memória ou base de casos (BC) e adaptação da solução utilizada no problema recuperado para resolver o novo problema.

RBC difere de outros paradigmas de AM nos seguintes aspectos (Aamodt e Plaza, 1994):

- Enquanto outros paradigmas utilizam conhecimento geral do domínio ou constroem relações entre problemas e soluções, RBC é capaz de utilizar conhecimento específico de problemas vistos anteriormente.
- RBC possibilita de forma natural o aprendizado incremental, pela atualização da BC sempre que um novo problema é resolvido, tornando o novo conhecimento disponível para futuro uso.

4.6.1 Representação de Casos

O desempenho de um sistema de RBC depende da estrutura e conteúdo de sua base de casos. Para a construção de uma base de casos, é necessário decidir o que armazenar em um caso, encontrar uma estrutura apropriada para descrever o conteúdo dos casos e definir como os casos devem ser organizados e indexados, para possibilitar a recuperação rápida e a reutilização eficaz de soluções anteriores.

Um caso pode representar diferentes tipos de conhecimento e assumir distintas formas de representação. Uma forma simples de representar casos é por meio de um conjunto de pares atributo-valor. Esse conjunto é dividido em dois subconjuntos. O primeiro possui os atributos que descrevem o problema e o segundo, os atributos relacionados à sua solução.

A Figura 4.6 ilustra um exemplo de BC e de um novo caso. Nessa figura, a base de casos armazena casos relacionados ao problema de escolha de pacotes de viagens. Cada caso tem a descrição de um problema que ocorreu no passado, a descrição dos requisitos de um cliente para um pacote, e a solução utilizada para resolver o problema, com a sugestão de local a ser visitado, meio de transporte para o local, acomodação e opção para refeições. O novo caso descreve um conjunto de requisitos para um novo pacote de viagem. Utilizando essa descrição, um sistema de RBC recupera o caso mais semelhante e o adapta de forma a sugerir uma solução para esse novo problema. É importante observar que sistemas de RBC podem recuperar e adaptar um ou mais casos.

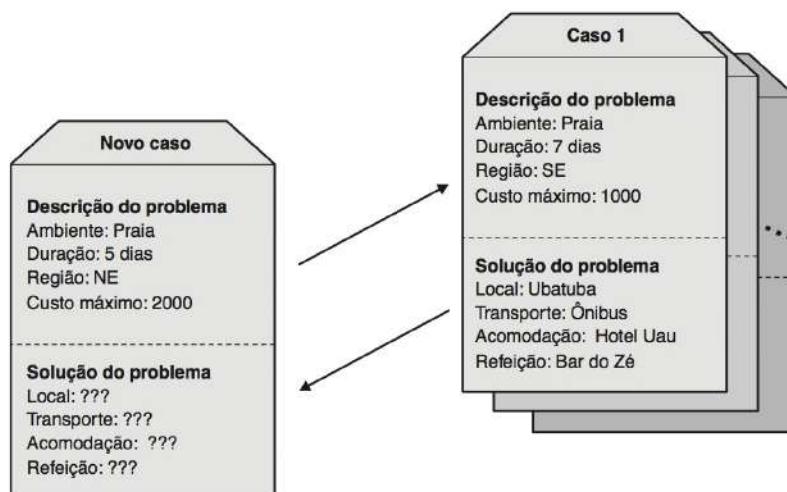


Figura 4.6 Exemplo de um novo caso e de uma BC.

Um aspecto importante em um sistema de RBC é a forma utilizada para a indexação de casos. No processo de indexação, os casos armazenados recebem índices, que são utilizados para futuras comparações e recuperações de casos (Aamodt e Plaza, 1994). A definição dos índices leva em conta os aspectos dos casos considerados relevantes para a recuperação de casos similares da BC. Em geral, os índices são um subconjunto dos atributos utilizados para a descrição dos problemas. Os índices precisam ser genéricos o suficiente para facilitar a identificação de casos similares, mas não tão genéricos para evitar a recuperação de casos que tenham pouca relação com o novo caso.

A forma como os casos são armazenados na base de casos influencia a facilidade de sua recuperação. Dois modelos de memória são geralmente utilizados: *memória plana* e *estrutura hierárquica* (Malek e Amy, 1994). Na primeira, todos os casos são armazenados em um mesmo nível. Na segunda, os casos são estruturados em forma de árvore.

4.6.2 O Ciclo de Raciocínio Baseado em Casos

Um modelo frequentemente utilizado para descrever as etapas de um sistema RBC é o ciclo de RBC proposto por Aamodt e Plaza (1994), ilustrado na Figura 4.7. Esse ciclo comprehende quatro etapas principais:

1. **Recuperação:** Recuperação do caso armazenado na BC mais similar ao novo problema apresentado.
2. **Reutilização:** Adaptação da parte de solução do caso recuperado. A solução do problema recuperado é geralmente utilizada como ponto de partida para propor uma solução para o novo problema. Essa etapa também é denominada *adaptação de casos*.
3. **Revisão:** A solução adaptada é revisada pelo usuário para validar sua relevância para resolver o novo problema.
4. **Retenção:** Caso, após a etapa de revisão, a solução adaptada seja considerada relevante, o novo problema, junto com essa solução, pode ser armazenado na BC.



Figura 4.7 Ciclo de raciocínio baseado em casos (Aamodt e Plaza, 1994).

A sequência de etapas no ciclo de RBC pode ser resumida da seguinte forma: a descrição de um novo problema define um novo caso. Esse novo caso é utilizado para a recuperação de um caso entre aqueles armazenados na coleção de casos previamente vistos (BC). A solução utilizada para resolver o caso recuperado é adaptada de forma a prover uma possível solução para o problema inicial. A solução proposta é avaliada em um processo de revisão, que pode ocorrer pela sua aplicação no mundo real, por meio de uma simulação, pela avaliação do usuário ou pelo uso do conhecimento da própria BC. Caso a nova solução seja considerada válida, ela, junto com a descrição do problema associado, pode ser armazenada na BC. Com isso, ela pode ser utilizada no futuro para resolver novos casos (Aamodt e Plaza, 1994).

4.7 Considerações Finais

Este capítulo descreveu técnicas de AM com base no cálculo de distância entre objetos. Para isso, foram apresentadas variações do algoritmo dos k vizinhos mais próximos e a metodologia de RBC.

O algoritmo k -NN é um dos principais algoritmos utilizados pela comunidade de AM, por ser simples e apresentar uma boa acurácia preditiva em vários conjuntos de dados. O

desempenho desse algoritmo sofre grande influência do valor de k e da medida de distância utilizada. Existem trabalhos que propõem técnicas para ajuste automático do valor de k . Algumas variações desse algoritmo foram apresentados neste capítulo.

RBC pode amenizar alguns problemas encontrados em outros paradigmas de AM. Deve ser observado que RBC não é a resposta para todos os problemas encontrados em AM. Existem situações em que RBC não pode ser aplicado para a resolução de problemas, situações em que RBC não constitui a solução mais adequada e situações em que RBC deve ser aplicado em conjunto com outras técnicas.

De acordo com Main et al. (2001), Malek (2001), Jarmulak et al. (2001), Wiratunga et al. (2002), Plaza e Arcos (2002), Prentzas e Hatzilygeroudis (2002) e Falkman (2002), as atividades de pesquisa em RBC estão focadas em problemas como adaptação de casos, utilização de métricas de adaptabilidade durante a recuperação dos casos, sistemas híbridos que combinam RBC com outras técnicas de AM, manutenção da eficiência do sistema durante seu uso e manutenção do conhecimento armazenado na BC.

Capítulo 5

Métodos Probabilísticos

Uma outra forma de lidar com tarefas preditivas em AM, principalmente quando as informações disponíveis são incompletas ou imprecisas, é por meio do uso de algoritmos baseados no teorema de Bayes, os métodos probabilísticos Bayesianos. Os métodos probabilísticos bayesianos assumem que a probabilidade de um evento A , que pode ser uma classe (por exemplo, um doente apresentar uma determinada doença), dado um evento B , que pode ser o conjunto de valores dos atributos de entrada (por exemplo, ter resultado positivo em um exame de raios X), não depende apenas da relação entre A e B , mas também da probabilidade de observar A independentemente de observar B (Mitchell, 1997).

A probabilidade de ocorrência do evento B pode ser estimada pela observação da frequência com que esse evento ocorre. De forma semelhante, é possível estimar a probabilidade de que um evento B ocorra, para cada classe ou evento A , $P(B|A)$. Mas como determinar a probabilidade de que um evento A , quando for observado um evento B , $P(A|B)$? O teorema de Bayes mostra como calcular $P(A|B)$. O teorema de Bayes fornece uma maneira de calcular a probabilidade de um evento ou objeto pertencer a uma classe $P(A|B)$ utilizando a probabilidade *a priori* da classe, $P(A)$, a probabilidade de observar vários objetos que pertencem à classe, $P(B|A)$, e a probabilidade de ocorrência desses objetos, $P(B)$.

Os possíveis valores do exame ou experimento, valores do conjunto de atributos de entrada, definem o espaço de resultados ou espaço amostral (Ω). A probabilidade P de um evento E (por exemplo, pacientes cujo resultado num exame foi positivo) é designada por $P(E)$, que satisfaz os axiomas de Kolmogorof (Pestana e Velosa, 2002):

- $0 \leq P(E);$
- Se Ω é o espaço de eventos, então $P(\Omega) = 1;$
- Se A e B são eventos disjuntos, então $P(A \cup B) = P(A) + P(B).$

A partir desses axiomas e definições, é possível derivar a *lei da probabilidade total*, que afirma que se B_1, B_2, \dots, B_n formam uma partição em Ω , então, para qualquer evento A , tem-se que:

$$P(A) = \sum_{i=1}^n P(A|B_i) \times P(B_i)$$

É possível também derivar a *lei da probabilidade condicional*:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Esses teoremas permitem deduzir o teorema de Bayes, tendo em conta que:

$$\begin{aligned} P(A \cap B) &= P(B \cap A) \\ P(A|B)P(B) &= P(A \cap B) = P(B|A)P(A) \\ P(A|B) &= \frac{P(B|A)P(A)}{P(B)} \end{aligned}$$

Na próxima seção, é feita uma introdução ao aprendizado bayesiano, representação de informação usando modelos probabilísticos gráficos, e à utilização do teorema de Bayes em inferência. A Seção 5.2 descreve um dos classificadores bayesianos mais populares: o *naive Bayes*. Por último, na Seção 5.3, são apresentadas redes bayesianas para classificação.

5.1 Aprendizado Bayesiano

Para exemplificar como métodos probabilísticos podem ser utilizados em AM, considere o seguinte cenário: Supor que a probabilidade de observar alguém com uma dada doença é de 8%. Existe um teste para o diagnóstico dessa doença, cujo resultado possui um grau de incerteza. É sabido que em 75% dos casos em que o resultado do teste foi positivo a doença foi confirmada e que em 96% em que o resultado do teste foi negativo o paciente não tinha a doença.

Como é possível representar essa informação? A doença pode ser vista como uma variável aleatória com dois possíveis valores: presente e ausente. O resultado do teste também tem dois possíveis valores alternativos: positivo ou negativo. É fácil observar que o valor da *Doença* influencia o valor do *Teste*, mas o oposto não é verdade. É útil representar essa informação sob a forma de um grafo.

A Figura 5.1 apresenta um grafo para esse problema. Nesse grafo, os nós representam variáveis ou atributos e as arestas direcionadas representam a influência entre variáveis. A informação gráfica é qualitativa, pois a direção da aresta nos diz que o valor da variável *doença* influencia o valor da variável *teste*. Também temos uma informação quantitativa: a probabilidade *a priori* de observar a doença:

$$P(\text{Doença} = \text{presente}) = 0,08 \text{ e } P(\text{Doença} = \text{ausente}) = 0,92$$

A partir de experiências passadas da utilização do teste, é possível inferir as probabilidades condicionais para a variável *Teste*:

$$P(\text{Teste} = \text{positivo} | \text{Doença} = \text{presente}) = 0,75$$

$$P(\text{Teste} = \text{negativo} | \text{Doença} = \text{ausente}) = 0,96$$

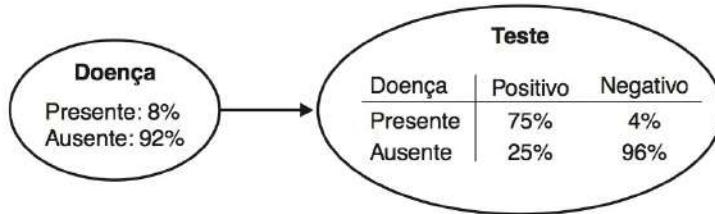


Figura 5.1 Modelo probabilístico gráfico para representar a informação no problema médico. A figura mostra o modelo qualitativo e o modelo quantitativo.

Essa informação é definida nos nós do modelo gráfico, tal como ilustrado na Figura 5.1. O modelo probabilístico gráfico é constituído pelo modelo qualitativo – um grafo cujos nós representam variáveis – e pelo modelo quantitativo – tabelas com a distribuição das variáveis. A probabilidade de verdadeiros positivos (o *Teste* deu positivo quando a *Doença* está presente) é de 75% (taxa denominada *sensibilidade*) e a probabilidade de verdadeiros negativos (o *Teste* deu negativo quando a *Doença* está ausente) é de 96% (taxa denominada *especificidade*).

Qual é o poder preditivo do *Teste* com respeito à *Doença*? É possível calcular as probabilidades *a priori* para a variável *Teste*. Levando em conta que $P(A) = P(A|B) \times P(B)$, obtemos:

$$\begin{aligned} P(\text{Teste} = \text{positivo}) &= \\ &= P(\text{Teste} = \text{positivo} | \text{Doença} = \text{presente}) \times P(\text{Doença} = \text{presente}) + \\ &+ P(\text{Teste} = \text{positivo} | \text{Doença} = \text{ausente}) \times P(\text{Doença} = \text{ausente}) \\ &= 0,75 \times 0,08 + 0,04 \times 0,92 = 0,0968 \end{aligned}$$

e

$$\begin{aligned} P(\text{Teste} = \text{negativo}) &= \\ &= P(\text{Teste} = \text{negativo} | \text{Doença} = \text{presente}) \times P(\text{Doença} = \text{presente}) + \\ &+ P(\text{Teste} = \text{negativo} | \text{Doença} = \text{ausente}) \times P(\text{Doença} = \text{ausente}) \\ &= 0,25 \times 0,08 + 0,96 \times 0,92 = 0,9032 \end{aligned}$$

Considere agora que, para um dado paciente, o resultado do *Teste* foi positivo. Podemos concluir que o paciente está doente?

No aprendizado bayesiano, o valor de uma variável aleatória tem uma probabilidade associada. A questão que devemos colocar é: Qual é a probabilidade $P(\text{Doença} = \text{presente} | \text{Teste} = \text{positivo})$?

O teorema de Bayes é usado para calcular a probabilidade *a posteriori* de um evento, dados sua probabilidade *a priori* e a verossimilhança do novo dado. Nesse exemplo, precisamos inverter a probabilidade $P(\text{Teste} = \text{positivo} | \text{Doença} = \text{presente})$. Na próxima seção será visto como isso pode ser feito.

5.1.1 O Problema de Inferência e o Teorema de Bayes

A literatura de reconhecimento de padrões (Duda et al., 2001) e aprendizado de máquina (Mitchell, 1997) apresenta diversas propostas para lidar com o problema de aprendizado em um cenário probabilístico. Suponha que $P(y_i|\mathbf{x})$ denota a probabilidade de um exemplo \mathbf{x} pertencer à classe y_i . A função de custo zero-um, que representa custo de associar \mathbf{x} à classe incorreta, é minimizada se, e somente se, \mathbf{x} é associado à classe y_k para a qual $P(y_k|\mathbf{x})$ é máxima (Duda et al., 2001). Esse método é designado por estimativa MAP (do inglês, *Maximum A Posteriori*). Formalmente, a classe que deve ser associada ao exemplo \mathbf{x} é dada pela expressão:

$$y_{MAP} = \arg \max_i P(y_i|\mathbf{x}) \quad (5.1)$$

na qual $\arg \max_i$ retorna a classe y_i com maior probabilidade de estar associada a \mathbf{x} , que é aquela que possui o valor máximo para $P(y_i|\mathbf{x})$.

Qualquer função que calcula as probabilidades condicionadas $P(y_i|\mathbf{x})$ é referida como uma *função discriminante*, por separar exemplos de classes diferentes. Dado um exemplo \mathbf{x} , o teorema de Bayes provê um método para calcular $P(y_i|\mathbf{x})$:

$$P(y_i|\mathbf{x}) = \frac{P(y_i)P(\mathbf{x}|y_i)}{P(\mathbf{x})} \quad (5.2)$$

O denominador, $P(\mathbf{x})$, pode ser ignorado, uma vez que é o mesmo para todas as classes, não afetando os valores relativos de suas probabilidades. Assumindo que as probabilidades *a priori* das hipóteses y_i são iguais, a Equação 5.2 pode ser simplificada considerando apenas o termo $P(\mathbf{x}|y_i)$ para calcular a hipótese mais provável. $P(\text{Dados}|\text{hipótese})$ é designado por verossimilhança, e a hipótese que maximiza $P(\text{Dados}|\text{hipótese})$ é designada por máxima verossimilhança, que pode ser expressa por:

$$h_{MV} = \arg \max_i P(\mathbf{x}|y_i) \quad (5.3)$$

Embora essa regra seja ótima, sua aplicabilidade é reduzida devido ao grande número de exemplos necessários para calcular, de forma viável, $P(\mathbf{x}|y_i)$. Para superar esse problema, várias hipóteses são usualmente propostas. Dependendo das hipóteses propostas, diferentes funções discriminantes são obtidas, levando a diferentes classificadores. Neste capítulo, será estudado um tipo de função discriminante que leva ao classificador *naive Bayes*.

5.2 Classificador *Naive Bayes*

Assumindo que os valores dos atributos de um exemplo são independentes entre si dada a classe, $P(\mathbf{x}|y_i)$ pode ser decomposto no produto $P(\mathbf{x}_1|y_i) \times \dots \times P(\mathbf{x}_d|y_i)$, em que

\mathbf{x}_j é o j -ésimo atributo do exemplo \mathbf{x} . Com isso, a probabilidade de um exemplo pertencer à classe y_i é proporcional à expressão:

$$P(y_i|\mathbf{x}) \propto P(y_i) \prod_{j=1}^d P(\mathbf{x}_j|y_i) \quad (5.4)$$

O classificador obtido pelo uso da função discriminante dada pela Equação 5.4 e pela regra de decisão ilustrada na Equação 5.1 é conhecido como classificador *naive* Bayes. O termo *naive* vem da hipótese de que os valores dos atributos de um exemplo são independentes de sua classe.

A fórmula do *naive* Bayes pode ser expressa em uma forma aditiva. Aplicando logaritmos à Equação 5.4, obtém-se:

$$\log(P(y_i|\mathbf{x})) \propto \log(P(y_i)) + \sum_j \log(P(\mathbf{x}_j|y_i)) \quad (5.5)$$

Para o caso particular de duas classes, a Equação 5.4 pode ser reescrita como:

$$\log \frac{P(y_1|\mathbf{x})}{P(y_2|\mathbf{x})} \propto \log \frac{P(y_1)}{P(y_2)} + \sum_j \log \frac{P(\mathbf{x}_j|y_1)}{P(\mathbf{x}_j|y_2)} \quad (5.6)$$

Nessa nova formulação, o sinal de cada termo indica a contribuição de cada atributo para cada classe. Se o quociente $P(\mathbf{x}_j|y_1)/P(\mathbf{x}_j|y_2)$ é maior que 1, o logaritmo é positivo e o atributo contribui para a predição da classe y_1 .

5.2.1 Detalhes de Implementação

Todas as probabilidades necessárias para a obtenção do classificador *naive* Bayes são computadas a partir dos dados de treinamento. Para calcular a probabilidade *a priori* de observar a classe y_i , $P(y_i)$, é necessário manter um contador para cada classe. Para calcular a probabilidade condicional de observar um valor de um atributo dado que o exemplo pertence a uma classe, é necessário distinguir entre atributos nominais e atributos contínuos.

No caso de atributos nominais, o conjunto de possíveis valores é um conjunto enumerável. Para calcular a probabilidade condicional, basta manter um contador para cada valor de atributo por classe. No caso de atributos contínuos, quando o número de possíveis valores é infinito, há duas possibilidades. A primeira é assumir uma distribuição particular para os valores do atributo, e geralmente é assumida a distribuição normal. A segunda alternativa é discretizar o atributo em uma fase de pré-processamento. Já foi mostrado que a primeira possibilidade produz piores resultados que a última (Dougherty et al., 1995; Domingos e Pazzani, 1997).

Vários métodos para discretização aparecem na literatura. Uma boa discussão sobre discretização é apresentada em Dougherty et al. (1995). Domingos e Pazzani (1997) propõem que o número de intervalos seja fixado em $k = \min(10, \text{número de valores diferentes})$ intervalos do mesmo tamanho. Uma vez que o atributo foi discretizado, um contador para cada classe e para cada intervalo pode ser utilizado para calcular a probabilidade condicional $P(\text{Atributo}_j|\text{Classe}_i)$.

5.2.2 Um Exemplo Ilustrativo

Este exemplo utiliza um conjunto de dados para o problema `balance`, que é apresentado na Figura 5.2. Esse conjunto de dados foi gerado para modelar resultados de experimentos psicológicos. Nesse problema, cada exemplo é classificado em uma de três posições de uma balança: a balança está inclinada para a direita, para a esquerda ou sem inclinação para um dos lados (balanceada). Os atributos são o peso do lado esquerdo, a dimensão do braço esquerdo, o peso do lado direito e a dimensão do braço direito. A forma correta para encontrar a classe é o maior valor entre: $Distância_{Esq} \times Peso_{Esq}$ e $Distância_{Dir} \times Peso_{Dir}$. Se esses valores são iguais, o estado da balança, sua classe, é balanceado.

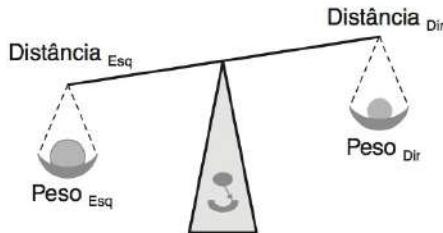


Figura 5.2 O problema do equilíbrio da balança.

Na versão do repositório UCI (Frank e Asuncion, 2010) para esse conjunto de dados, o domínio de todos os atributos é o conjunto $\{1, 2, 3, 4, 5\}$. O conjunto contém 625 exemplos, distribuídos da seguinte forma: em 49 exemplos a balança está平衡ada, em 288 exemplos a balança está inclinada para a esquerda e nos 288 exemplos restantes a balança está inclinada para a direita.

Para calcular as probabilidades *a priori*, $P(Classe_i)$, é necessário contar o número de exemplos para cada classe. Os resultados são apresentados na Tabela 5.1.

Tabela 5.1 Contagem de valores e probabilidade *a priori* para as classes

	Equilibrada	Esquerda	Direita
Contagem	49	288	288
$P(Classe)$	0,078	0,461	0,461

Para calcular a probabilidade condicional de observar um valor específico de atributo dada a classe, $P(Atributo_j|Classe_i)$, é necessário descobrir o tipo do atributo. Nesse problema, todos os atributos são numéricos, pois dizem respeito a distâncias e pesos. Pode-se assumir que seu domínio é um conjunto de \mathbb{R} . Sem mais nenhuma informação, a hipótese mais razoável é que eles são normalmente distribuídos. Considerando essa hipótese, para estimar as probabilidades condicionais, é preciso calcular a média e o desvio padrão dos valores dos atributos para cada classe.

Como alternativa, pode-se discretizar os atributos. Nesse problema, aplicando a regra $k = \min(10; \text{número de valores diferentes})$, são obtidos cinco intervalos. A Tabela 5.2 apresenta a distribuição de valores para cada atributo em cada classe.

A Figura 5.3 ilustra graficamente a Tabela 5.2. Na primeira linha são apresentadas as distribuições discretizadas para os atributos *Peso_{Esq}* e *Peso_{Dir}*, assim como a *Classe*. Podemos observar que para a classe **equilibrado** todas as contagens são similares. Para os atributos *Peso_{Esq}* e *Distância_{Esq}*, a contagem aumenta para a classe esquerda e diminui

Tabela 5.2 Tabelas de distribuição dos valores dos atributos por classe

Peso_{Esq}	Distribuição normal		Discretização				
	Média	Desvio padrão	V1	V2	V3	V4	V5
Equilibrado	2,938	1,42	10	11	9	10	9
Esquerda	3,611	1,23	17	43	63	77	88
Direita	2,399	1,33	98	71	53	38	28
Distância_{Esq}	Média	Desvio padrão	V1	V2	V3	V4	V5
Equilibrado	2,938	1,42	10	11	9	10	9
Esquerda	3,611	1,22	17	43	63	77	88
Direita	2,399	1,33	98	71	53	38	28
Peso_{Dir}	Média	Desvio padrão	V1	V2	V3	V4	V5
Equilibrado	2,938	1,42	10	11	9	10	9
Esquerda	2,399	1,33	71	53	38	28	17
Direita	3,611	1,22	17	43	63	77	88
Distância_{Dir}	Média	Desvio padrão	V1	V2	V3	V4	V5
Equilibrado	2,938	1,42	10	11	9	10	9
Esquerda	2,399	1,33	98	71	53	38	28
Direita	3,611	1,22	17	43	63	77	88

para a classe direita. Para os atributos $Peso_{Dir}$ e $Distância_{Dir}$, a contagem aumenta para a classe direita e diminui para a classe esquerda. A segunda linha mostra, para os mesmos atributos, a função densidade de probabilidade por classe, assumindo uma distribuição normal.

5.2.3 Análise do Algoritmo

A superfície de decisão de um classificador *naive* Bayes em um problema de duas classes definido por atributos booleanos é um hiperplano, ou seja, a superfície de decisão é linear. Todas as probabilidades exigidas pela Equação 5.4 podem ser calculadas a partir do conjunto de treinamento em uma única passagem. O processo de construir o modelo é bastante eficiente. Outro aspecto interessante do algoritmo é que ele é fácil de implementar de uma forma incremental.

Domingos e Pazzani (1997) mostram que o *naive* Bayes tem um bom desempenho em uma grande variedade de domínios, incluindo muitos em que há claras dependências entre os atributos. Os autores argumentam que, em problemas de classificação e para a função de custo 0 – 1, um exemplo é corretamente classificado desde que a ordenação das classes dada pelas estimativas das probabilidades *a posteriori* esteja correta, independentemente de essas estimativas serem (ou não) realistas.

Kononenko (1991) sugere que esse classificador é robusto à presença de ruídos e atributos irrelevantes. Eles também notaram que as teorias aprendidas são fáceis de compreender pelos especialistas do domínio. Essa observação é devida ao fato de que o *naive* Bayes resume a variabilidade do conjunto de dados em tabelas de contingência, e assume que estas são suficientes para distinguir entre as classes.

O desempenho do *naive* Bayes não decresce na presença de atributos irrelevantes.

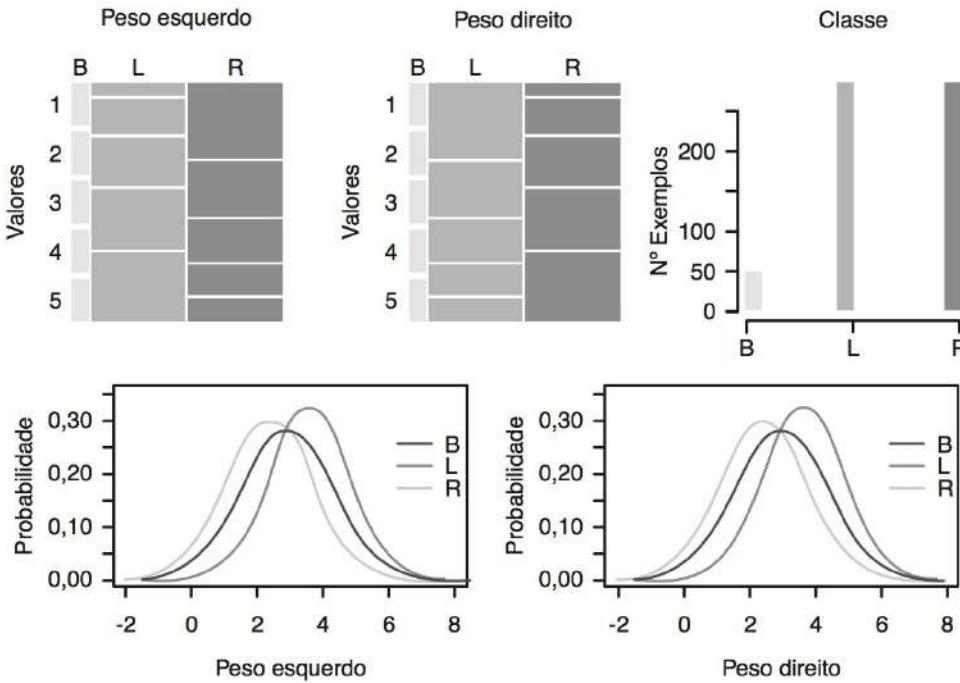


Figura 5.3 *Naive Bayes para o problema da balança.*

Suponha um problema de duas classes, em que o i -ésimo atributo é irrelevante: $P(\mathbf{x}_i|y_1) = P(\mathbf{x}_i|y_2)$. A partir da fórmula do classificador *naive Bayes*:

$$P(y_k|\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_d) \propto P(y_k)P(\mathbf{x}_i|y_k) \prod_{l=1}^{i-1} P(\mathbf{x}_l|y_k) \prod_{l=i+1}^d P(\mathbf{x}_l|y_k) \quad (5.7)$$

$$P(y_k|\mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_d) \propto P(y_k|\mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_{i+1}, \dots, \mathbf{x}_d) \quad (5.8)$$

O impacto das variáveis redundantes deve ser levado em consideração. Assuma que o $(i-1)$ -ésimo e o i -ésimo atributos são redundantes, ou seja, para todas as classes y $P(\mathbf{x}_{i-1}|y) = P(\mathbf{x}_i|y)$.

$$P(y_k|\mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_i, \dots, \mathbf{x}_d) \propto P(y_k)P(\mathbf{x}_{i-1}|y_k)P(\mathbf{x}_i|y_k) \prod_{l=1}^{i-2} P(\mathbf{x}_l|y_k) \prod_{l=i+1}^d P(\mathbf{x}_l|y_k) \quad (5.9)$$

$$P(y_k|\mathbf{x}_1, \dots, \mathbf{x}_{i-1}, \mathbf{x}_i, \dots, \mathbf{x}_d) \propto P(y_k)P(\mathbf{x}_i|y_k)^2 \prod_{l=1}^{i-2} P(\mathbf{x}_l|y_k) \prod_{l=i+1}^d P(\mathbf{x}_l|y_k) \quad (5.10)$$

5.2.4 Desenvolvimentos

Várias técnicas foram desenvolvidas para melhorar o desempenho do classificador *naive Bayes*. Algumas dessas técnicas aplicam diferentes classificadores *naive Bayes* para diferentes regiões do espaço de entrada. Por exemplo, Langley (1993) apresentou um algoritmo *naive Bayes* que, recursivamente, constrói uma hierarquia das descrições dos conceitos probabilísticos. Kohavi (1996) apresentou uma árvore de *naive Bayes*. É um algoritmo híbrido que gera uma árvore de decisão univariada regular, cujas folhas contêm

um classificador *naive* Bayes. O classificador associado a cada nó folha é construído a partir de exemplos que levam a esse nó. A proposta retém a interpretabilidade do *naive* Bayes e das árvores de decisão, resultando em um classificador que frequentemente supera ambos os constituintes, especialmente em grandes conjuntos de dados.

Outras técnicas constroem novos atributos que refletem interdependências entre atributos originais. Por exemplo, Kononenko (1991) apresentou um classificador semi-*naive* Bayes. O classificador procura combinar pares de atributos, fazendo um atributo produto-cruzado, baseado em testes estatísticos para independência. A avaliação experimental foi não conclusiva. Em outro exemplo, Pazzani (1996) apresentou um classificador bayesiano construtivo. Para isso, emprega um modelo encapsulado (John et al., 1994) para encontrar os melhores atributos do produto cartesiano a partir de atributos nominais existentes. O classificador também considera a eliminação de atributos existentes. Notou-se com isso uma melhora no classificador *naive* Bayes.

Técnicas que abordam o problema da presença de atributos contínuos estão também presentes na literatura. John (1997) apresentou o *Bayes Flexível*, que utiliza, para atributos contínuos, uma estimativa de densidade de funções *kernel* (em vez de uma única hipótese gaussiana), mas retém a hipótese de independência. Para cada atributo contínuo, a densidade estimada é obtida pela expressão $P(\mathbf{x}_t|y_i) = \frac{1}{n} \sum_i N(\mathbf{x}_t, \mathbf{x}_i, \sigma_c)$, em que n representa o número de exemplos de treino da classe y_i , e σ_c representa o tamanho de banda do *kernel*. Dessa forma, $P(\mathbf{x}_t|y_i)$ é obtido agregando sobre todos os exemplos de treino da classe y_i .

A avaliação experimental em conjuntos de dados do repositório UCI mostra que o *Bayes flexível* alcança, em muitos domínios, acurárias preditivas significantemente maiores que o *naive* Bayes. Gama (2000) apresentou um algoritmo *Linear Bayes* que utiliza uma distribuição normal multivariada para cada classe para calcular a verossimilhança $P(\mathbf{x}_i, \dots, \mathbf{x}_j|y_i)$, em que $\mathbf{x}_i, \dots, \mathbf{x}_j$ representa o conjunto de atributos contínuos. Essa estratégia mostrou-se melhor do que o *naive* Bayes usando discretização ou uma distribuição gaussiana univariada.

5.3 Redes Bayesianas para Classificação

Já foi mencionada a incapacidade do classificador *naive* Bayes para lidar com interdependências entre atributos. Vale lembrar que duas variáveis aleatórias (atributos) X, Y são *independentes* quando $P(X, Y) = P(X) \times P(Y)$, ou seja, conhecer o valor de uma não traz informação sobre o valor da outra variável, o que implica $P(X|Y) = P(X)$. Esta seção tem como tema central a *independência condicional*: casos em que existe uma relação estatística entre duas variáveis quando uma terceira variável é conhecida. Formalmente, X é condicionalmente independente de Y dado Z se $P(X|Y, Z) = P(X|Z)$.

Os modelos gráficos probabilísticos, ou redes bayesianas (Pearl, 1988), utilizam o conceito de *independência condicional* entre variáveis para obter um equilíbrio entre o número de parâmetros a calcular e a representação de dependências entre as variáveis. Esses modelos representam a distribuição de probabilidade conjunta de um grupo de variáveis aleatórias em um domínio específico.

Formalmente, seja $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ um conjunto de variáveis aleatórias para um dado domínio. Uma rede bayesiana (RB) sobre \mathbf{x} é uma tupla (S, Θ_S) em que o primeiro componente, a *estrutura da rede* S , é um *grafo acíclico direcionado* (DAG, do inglês, *Directed Acyclic Graph*).

ted Acyclic Graph) cujos *nós* representam as variáveis aleatórias e as *arestas* representam dependências diretas entre variáveis. Um arco entre dois nós, denota *influência* ou *correlação*. O conjunto de variáveis aleatórias (nós do DAG) que influenciam uma variável x_i é usualmente designado por **Pais** de x_i . A segunda componente Θ_S é o conjunto de *tabelas de probabilidade condicional*.

Assumindo que as variáveis são discretas, cada $P(\mathbf{x}_i|\text{Pai}_i) \in \Theta_S$ representa a *tabela de probabilidade condicional* (TBC) sobre os valores de x_i dados os valores do seu pai Pai_i . Além disso, o DAG S satisfaçõa à condição de Markov: *cada nó é independente de todos os seus não descendentes dados os seus pais em S* . Isso permite que a distribuição de probabilidade conjunta sobre \mathbf{x} seja representada na forma fatorada: $P(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) = \prod_{i=1}^n P(\mathbf{x}_i | \text{Pai}_i)$. A Figura 5.4 apresenta um exemplo de uma rede bayesiana. É apresentado o modelo qualitativo – um grafo cujos nós representam variáveis – e o modelo quantitativo – tabelas com a distribuição de probabilidades da variável *Salário* dadas as variáveis que a influenciam. A fatoração conjunta para as variáveis do modelo é:

$$P(\text{Classe})P(\text{Edu}|\text{Classe})P(\text{Salário}|\text{Edu}, \text{Classe})P(\text{Área}|\text{Salário}, \text{Classe}).$$

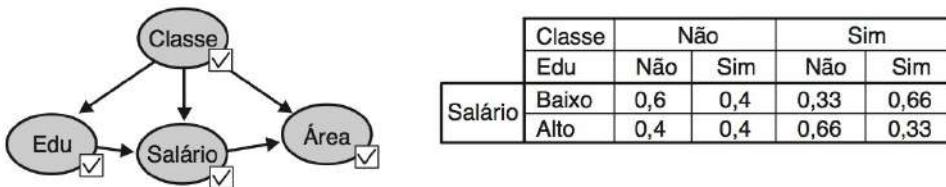


Figura 5.4 A Figura mostra o modelo qualitativo – um grafo cujos nós representam variáveis – e o modelo quantitativo para a variável Salário – tabelas com a distribuição de probabilidades dos valores da variável Salário dado o valor das variáveis que a influenciam.

Uma rede bayesiana pode ser utilizada para classificação de uma forma relativamente simples. Uma das variáveis é selecionada como atributo alvo, e todas as outras variáveis são atributos de entrada. O conjunto de variáveis que influenciam o atributo alvo é designado por *Markov Blanket*: é constituído pelas variáveis pais da variável alvo, pelos filhos da variável alvo e pelos pais dos filhos da variável alvo. Assim, uma rede RB pode ser utilizada como um classificador que, dado um exemplo \mathbf{x} , fornece a distribuição de probabilidade *a posteriori* $P(y | \mathbf{x})$ do nó classe $y \in Y$. É possível calcular a probabilidade *a posteriori* $P(y | \mathbf{x}, S)$ para cada classe $y \in Y$ marginalizando a distribuição de probabilidade conjunta $P(y, \mathbf{x} | S)$ e então retornar a classe \hat{y} que a maximiza:

$$\hat{y} = h_{\text{CRB}}(\mathbf{x}) = \arg \max_{j=1 \dots k} P(y_j, \mathbf{x} | S) \quad (5.11)$$

Dado um conjunto de treinamento, o problema de aprendizado consiste portanto em selecionar o classificador baseado em RB (CRB), isto é, a hipótese $h_C = (S, \Theta_S)$ que produz a classificação com maior acurácia para dados não conhecidos.

Esse problema pode ser resolvido inicialmente pela escolha de um modelo de classe adequado que define o espaço de possíveis estruturas RB. Em seguida, dentro desse modelo de classe, uma *estrutura* é selecionada. Finalmente, os parâmetros são estimados a partir dos dados.

O problema de escolher a estrutura mais apropriada para um determinado problema está relacionado à *seleção de modelos*, uma área da inferência estatística que estuda a seleção, dentre um conjunto de modelos concorrentes, daquele que “*melhor se ajusta*”, em algum sentido, aos dados disponíveis. Considere as propostas *baseadas em pontuação* para a seleção do modelo, em que a noção de “*melhor se ajusta*” é definida via uma *função de pontuação* que mede a qualidade de cada hipótese candidata. Propostas baseadas em pontuação podem ser descritas como um *problema de busca*, em que cada estado no espaço de busca identifica um possível DAG. O método de busca utiliza o valor retornado pela pontuação para ajudar a guiar a busca.

O problema da seleção de uma função de pontuação apropriada para aprendizado de CRBs tem recebido muita atenção (Domingos e Pazzani, 1997; Friedman et al., 1997; Kontkanen et al., 1999). Quando uma RB é induzida para classificação, o objetivo principal é construir um classificador com elevada acurácia preditiva. Por isso, foi sugerido que estratégias de busca para aprendizado de CRBs deveriam selecionar entre modelos utilizando pontuações especializadas para classificação (*pontuações supervisionadas*); caso contrário a busca poderia resultar em um escolha subótima (Kontkanen et al., 1999). Desse modo, uma pontuação baseada na distribuição da probabilidade conjunta não necessariamente será ótima em problemas de classificação.

Outro aspecto que pode também influenciar o desempenho dos CRBs induzidos é a seleção de um *modelo de classe* apropriado, que define o espaço de busca e, consequentemente, a complexidade dos CRBs induzidos. A seleção do modelo procura um equilíbrio *viés-variancia* a fim de selecionar um modelo com a complexidade apropriada que é automaticamente regularizado pela função de pontuação (Hastie et al., 2001). Podemos obter um desempenho desejado dos CRBs induzidos se, a cada momento, tentarmos selecionar o modelo de classe apropriado, com a complexidade adequada, para os dados de treinamento disponíveis.

5.3.1 Classificadores Bayesianos com k -Dependências

Classificadores bayesianos com k -dependências (Sahami, 1996) (k -CBDs) representam um enquadramento unificado para todas aquelas classes de CRBs que contêm a estrutura de Bayes simples. Um k -CBD, além disso, permite que cada atributo tenha no máximo k nós atributos como pais.

Como ilustrado na Figura 5.5, é possível variar o valor de k e obter classificadores bayesianos de complexidade crescente, que se movam gradualmente ao longo do espectro de dependências entre atributos. O *naive Bayes* é um 0-CBD e encontra-se no extremo mais restritivo, porque não permite dependências entre atributos. Um classificador TAN (Friedman et al., 1997) é um 1-CBDs (ele permite no máximo um atributo como pai de outro atributo). O BAN (Friedman et al., 1997; Cheng e Greiner, 1999), mostrado na Figura 5.5, é um 2-CBD (ele tem um máximo de duas dependências entre atributos). No extremo mais geral está o classificador $(n - 1)$ -CBD, que assume que todos os atributos interagem entre si. A fatoração da probabilidade conjunta $P(Classe, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ das cinco variáveis, codificada pelos modelos apresentados na Figura 5.5, é apresentada a seguir:

- *Naive Bayes*:

$$P(Classe)P(\mathbf{x}_1|Classe)P(\mathbf{x}_2|Classe)P(\mathbf{x}_3|Classe)P(\mathbf{x}_4|Classe)$$

- TAN:

$$P(\text{Classe})P(\mathbf{x}_1|\text{Classe})P(\mathbf{x}_2|\mathbf{x}_1, \text{Classe})P(\mathbf{x}_3|\mathbf{x}_2, \text{Classe})P(\mathbf{x}_4|\mathbf{x}_3, \text{Classe})$$

- BAN:

$$P(\text{Classe})P(\mathbf{x}_1|\text{Classe})P(\mathbf{x}_2|\mathbf{x}_1, \text{Classe})P(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2, \text{Classe})P(\mathbf{x}_4|\mathbf{x}_3, \text{Classe})$$

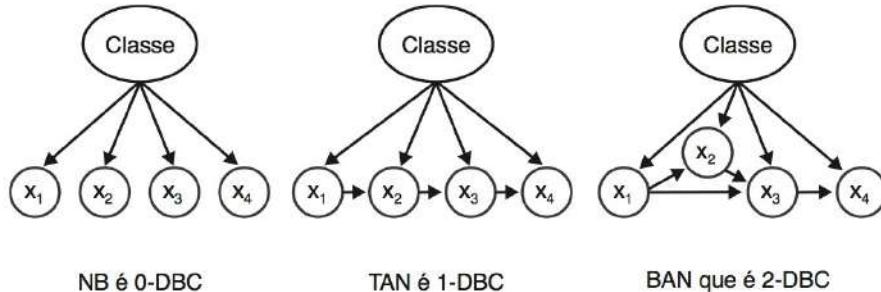


Figura 5.5 Exemplos de classificadores bayesianos com k -dependências.

Algoritmo 5.1 O algoritmo subida de encosta para aprendizado k -CBDs

Entrada: Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$

k : número de dependências admissíveis.

Saída: Um k -DBC com baixo valor de $\text{Erro}(S, \mathbf{D})$

```

1 /*  $S$  é o espaço de possíveis DAGs restritos por  $k$  */ ;
2 Inicialize  $S$  com a estrutura Bayes simples;
3 continuar ← Verdadeiro ;
4 enquanto continuar faça
5   Calcular  $\text{Erro}(S, \mathbf{D})$ ;
6   /* Seja  $(x', x'')$  a aresta que minimiza a função de avaliação.*/ ;
7    $(x', x'') = \arg \min \text{Erro}(S \cup (x_i, x_j), \mathbf{D}) \wedge$ 
8    $\wedge |\text{pai}(x_i) \setminus y| < k \wedge |\text{pai}(x_j) \setminus y| < k$  ;
9   se Existe a aresta  $(x', x'')$   $\wedge \text{Erro}(S \cup (x', x''), \mathbf{D}) < \text{Erro}(S, \mathbf{D})$  então
10    Adicionar a aresta  $(x', x'')$  em  $S$  ;
11  fim
12 senão
13  continuar ← Falso ;
14 fim
15 fim
16 Estimar os parâmetros  $\Theta_S$  dado  $S$  a partir dos dados  $\mathbf{D}$ ;
17 Retorna:  $k$ -DBC=( $S, \Theta_S$ );

```

Sahami (1996) propôs um algoritmo para indução de k -CBDs que usa o conceito de entropia condicional. Em Castillo (2006); Castillo e Gama (2005), é proposto um algoritmo de aprendizado subida de encosta (*hill-climbing*). É um algoritmo simples, incremental e de fácil implementação computacional. O algoritmo, cujo pseudocódigo é apresentado no Algoritmo 5.1, se inicia com a estrutura do *naive Bayes*. Iterativamente, ele adiciona arestas entre dois atributos que resultam em melhorias máximas na pontuação até

que não haja mais melhoras para aquela pontuação ou até que não seja possível adicionar uma nova aresta. A função de avaliação de modelos é a taxa de erro no conjunto de treinamento. Alternativamente, pode ser usada a taxa de erro num conjunto de validação.

Resultados obtidos em experimentos apresentados em Sahami (1996) e relacionados aos estudos usando *k*-CBDs (Blanco et al., 2005; Castillo e Gama, 2005; Webb et al., 2005) mostram que *modelagem de dependências de atributos* pode melhorar os resultados da classificação de Bayes simples. Conhecendo como o desempenho de classificação muda com o aumento do valor de *k*, podemos obter uma noção de nível de dependência entre atributos para cada domínio particular. Um resultado interessante foi apresentado em Castillo (2006); Castillo e Gama (2005), onde é estudada a relação entre o número de exemplos de treino e o valor de *k*. Nesse trabalho, são apresentados resultados experimentais que mostram que, para valores crescentes do número de exemplos de treinamento, o melhor desempenho é obtido crescendo o valor de *k*.

5.4 Considerações Finais

Os modelos gráficos probabilísticos representam a distribuição de probabilidade conjunta de um conjunto de variáveis aleatórias. É possível obter classificadores bayesianos de complexidade crescente, que consideram diferentes graus de dependências entre atributos. O *naive Bayes* é o mais restritivo porque não permite dependências entre atributos. No extremo mais geral, temos classificadores que assumem que todos os atributos interagem entre si. Entre os dois extremos, temos modelos de granularidade crescente. Os modelos gráficos probabilísticos são usados para diferentes tarefas de aprendizado, desde *previsão*, em que se pretende obter o resultado mais provável para os dados de entrada, até o *diagnóstico*, em que se pretende obter as causas mais prováveis para os efeitos observados.

Capítulo 6

Métodos Baseados em Procura

O problema de aprendizado de máquina pode ser formulado como um problema de procura num espaço de possíveis soluções. Dada uma linguagem para representar generalizações de exemplos e uma função de avaliação de hipóteses, o problema de aprendizado procura no espaço de hipóteses definido pela linguagem de representação. Os modelos que estudamos neste capítulo são os modelos baseados em árvores (árvores de decisão e regressão) e os modelos baseados em regras.

6.1 Árvores de Decisão e Regressão

Uma árvore de decisão usa a estratégia dividir para conquistar para resolver um problema de decisão. Um problema complexo é dividido em problemas mais simples, aos quais recursivamente é aplicada a mesma estratégia. As soluções dos subproblemas podem ser combinadas, na forma de uma árvore, para produzir uma solução do problema complexo. A força dessa proposta vem da capacidade de dividir o espaço de instâncias em subespaços e cada subespaço é ajustado usando diferentes modelos. Essa é a ideia básica por trás de algoritmos baseados em árvores de decisão, tais como: ID3 (Quinlan, 1979), ASSISTANT (Cestnik et al., 1987), CART (Breiman et al., 1984), C4.5 (Quinlan, 1993). Recentemente, diversos pacotes estatísticos, *S_{plus}*, *Statistica*, *SPSS* (Mattison, 1998), *R* (Ihaka e Gentleman, 1996) e *Microsoft SQL Server* (Seidman, 2001) incorporaram funções que implementam árvores de decisão para problemas de classificação e regressão. Os modelos em árvore são designados *árvores de decisão* no caso de problemas de classificação e *árvores de regressão* nos problemas de regressão. Quer em árvores de decisão quer em árvores de regressão, a interpretação dos modelos assim como os algoritmos de indução das árvores são muito semelhantes, pelo que iremos usar o termo *árvores de decisão* de uma forma genérica, explicitando quando necessário.

Formalmente, uma árvore de decisão é um grafo acíclico direcionado em que cada nó ou é um *nó de divisão*, com dois ou mais sucessores, ou um *nó folha*:

- Um *nó folha* é rotulado com uma *função*. Usualmente são considerados apenas os valores da variável alvo nos exemplos que chegam a um nó folha. No caso mais simples, a função é a constante que minimiza a função de custo. Em problemas de classificação, e assumindo a função de custo 0-1, essa constante é a *moda*. Em problemas de regressão, a constante que minimiza a função de custo do *erro do médio quadrático* é a *média*, enquanto para a função de custo do *desvio absoluto* é a *mediana*.

- Um *nó de divisão* contém um *teste condicional* baseado nos valores do atributo. Na proposta padrão, os testes são univariados: as condições envolvem um único atributo e valores no domínio desse atributo. Exemplos de teste condicional são:
 - Idade > 18;
 - Profissão ∈ {professor, estudante};
 - $0,3 + 0,2 \times x_1 - 0,5 \times x_2 \leq 0$.

A Figura 6.1 representa uma árvore de decisão e a divisão correspondente no espaço definido pelos atributos x_1 e x_2 . Cada nó da árvore corresponde a uma região nesse espaço. As regiões definidas pelas folhas da árvore são mutuamente excludentes, e a reunião dessas regiões cobre todo o espaço definido pelos atributos. A interseção das regiões abrangidas por quaisquer duas folhas é vazia. A união de todas as regiões (todas as folhas) é U .

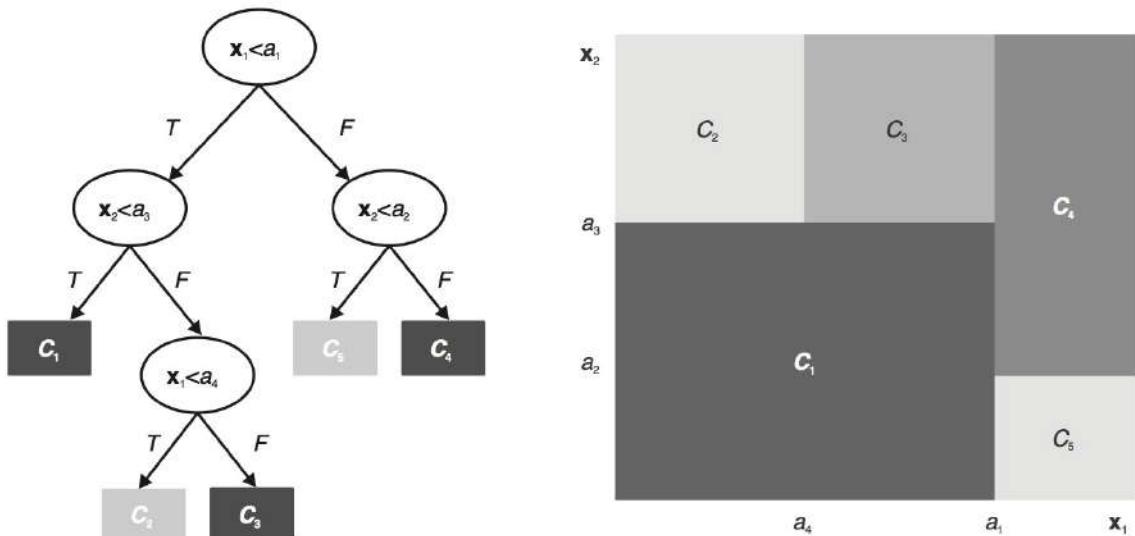


Figura 6.1 Uma árvore de decisão e as regiões de decisão no espaço de objetos.

Uma árvore de decisão abrange todo o espaço de instâncias. Esse fato implica que uma árvore de decisão pode fazer previsões para qualquer exemplo de entrada.

O espaço de hipóteses das árvores de decisão enquadra-se dentro do formalismo Forma Normal Disjuntiva (FND). Classificadores gerados por esses sistemas codificam uma FND para cada classe. Para cada FND, as condições ao longo de um ramo (um percurso entre a raiz e uma folha) são conjunções de condições e os ramos individuais são disjunções. Dessa forma, cada ramo forma uma regra com uma parte condicional e uma conclusão. A parte condicional é uma conjunção de condições. Condições são testes que envolvem um atributo particular, operador (por exemplo $=$, \geq etc.) e um valor do domínio do atributo. Esses tipos de testes correspondem, no espaço de entrada, a um hiperplano que é ortogonal aos eixos do atributo testado e paralelo a todos os outros eixos. As regiões produzidas por esses classificadores são todas hiper-retângulos, o que pode ser visualizado na Figura 6.1.

6.1.1 Indução de Árvores de Decisão e Regressão

O algoritmo que constrói uma árvore de decisão a partir de dados é muito simples. Os passos principais do algoritmo são descritos no Algoritmo 6.1. A entrada para a função

GeraÁrvore é um conjunto de dados \mathbf{D} . No passo 3, o algoritmo avalia o critério de parada. Se mais divisões do conjunto de dados são necessárias, é escolhido o atributo que maximiza alguma medida de impureza (passo 5). No passo 7, a função **GeraÁrvore** é recursivamente aplicada a cada partição do conjunto de dados \mathbf{D} .

Algoritmo 6.1 Algoritmo para construção de uma árvore de decisão

```

Entrada: Um conjunto de treinamento  $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$ 
Saída: Árvore de Decisão
1 /* Função GeraÁrvore( $\mathbf{D}$ ) */ ;
2 se critério de parada( $\mathbf{D}$ ) = Verdadeiro então
3   Retorna: um nó folha rotulado com a constante que minimiza a função perda ;
4 fim
5 Escolha o atributo que maximiza o critério de divisão em  $\mathbf{D}$  ;
6 para cada partição dos exemplos  $\mathbf{D}_i$  baseado nos valores do atributo escolhido faça
7   Induz uma subárvore  $\text{Árvore}_i = \text{GeraÁrvore}(\mathbf{D}_i)$  ;
8 fim
9 Retorna: Árvore contendo um nó de decisão baseado no atributo escolhido, e
descendentes  $\text{Árvore}_i$  ;

```

É conhecido que o problema da construção de uma árvore de decisão minimal (em termos do número de nós), condizente com um conjunto de dados, é um problema *NP completo* (Rivest, 1987). Usualmente os algoritmos exploram heurísticas que localmente executam uma pesquisa olha para a frente um passo. Uma vez que uma decisão é tomada, ela nunca é reconsiderada. Essa pesquisa de subida de encosta (*hill-climbing*) sem *backtracking* é suscetível aos riscos usuais de convergência a uma solução ótima localmente que não é ótima globalmente. Por outro lado, essa estratégia permite construir árvores de decisão em tempo *linear* no número de exemplos.

Nas próximas subseções iremos descrever brevemente os aspectos mais importantes de um algoritmo. Focamos nos dois sistemas mais bem-sucedidos e representativos de árvore de decisão: CART e C4.5.

Regras de Divisão para Classificação

Uma regra de divisão é guiada por uma medida de “*goodness of split*”, que indica quão bem um dado atributo discrimina as classes. Ela é usada para selecionar o atributo que maximiza essa medida (veja passo 5 do Algoritmo 6.1). Uma regra de divisão tipicamente funciona como uma heurística olha para a frente um passo. Para cada teste possível, o sistema hipoteticamente considera os subconjuntos dos dados obtidos. O sistema escolhe o teste que maximiza (ou minimiza) algumas funções heurísticas sobre os subconjuntos.

Considere um nó t , em que a probabilidade de observar um exemplo da classe c_i é p_i . A probabilidade de observar exemplos de cada classe é dada por p_1, p_2, \dots, p_k , tal que $\sum p_i = 1$. A impureza do nó t é uma função sobre a proporção da classe daquele nó: $i(t) = \phi(p_1, p_2, \dots, p_k)$. Suponha um teste de divisão S que divide os exemplos de treinamento em dois subconjuntos L e R . A redução na impureza dos testes pode ser medida como:

$\delta(S) = \phi(p_1, p_2, \dots, p_k) - P_L \times \phi(p_{1L}, p_{2L}, \dots, p_{kL}) - P_R \times \phi(p_{1R}, p_{2R}, \dots, p_{kR})$ em que P_L e P_R representam a probabilidade de que um exemplo de t vá para o subconjunto L e R , respectivamente. As características gerais de qualquer função de impureza são:

1. Simetria;
2. Ter um máximo quando $p_1 = p_2 = \dots = p_k$;
3. Ter um mínimo se $\exists i : p_i = 1$, ou seja, todos os exemplos são de uma classe, o que implica todas as outras $p_j = 0$.

Uma proposta natural é rotular cada subconjunto da divisão por sua classe mais frequente e escolher a divisão que tem menores erros. Há diversos problemas com essa proposta (veja por exemplo Breiman et al. (1984)). Um deles é que $\phi(p_1+p_2) \geq \phi(p_1)+\phi(p_2)$.

Diversos métodos foram descritos na literatura. A maioria deles concorda nos pontos extremos, isto é, que uma divisão que mantém a proporção de classes em todo o subconjunto não tem utilidade, e uma divisão na qual cada subconjunto contém somente exemplos de uma classe tem utilidade máxima. Casos intermediários podem ser classificados diferentemente pelas diferentes medidas.

Martin (1997) agrupa as medidas nas seguintes categorias das chamadas *funções de mérito*:

1. Medidas de diferença entre a distribuição no nó pai (antes da divisão) e a distribuição nos subconjuntos obtidos por alguma função baseada nas proporções de classe (tal como a entropia). Essas medidas enfatizam a *pureza* dos subconjuntos. CART chama essas medidas de funções de *impureza*.
2. Medidas da diferença entre os subconjuntos divididos com base em alguma função sobre as proporções de classe (tipicamente a distância ou um ângulo). Essas medidas enfatizam a *disparidade* dos subconjuntos.
3. Medidas estatísticas de independência (tipicamente um teste χ^2) entre as proporções de classe e os subconjuntos divididos. Essas medidas enfatizam o peso da evidência, a *confiança* das previsões da classe baseadas no relacionamento do subconjunto.

O resto desta seção explica as regras de divisão baseadas no *Ganho de Informação*, usado no C4.5, e no índice Gini, usado no CART.

Ganho de Informação. O conceito fundamental nessa proposta é o conceito de *entropia*. Entropia mede a aleatoriedade de uma variável aleatória. Suponha uma variável aleatória discreta A cujo domínio é $\{a_1, a_2, \dots, a_v\}$. Suponha que a probabilidade de observar cada valor é p_1, p_2, \dots, p_v . A entropia de A é dada por: $H(A) = -\sum_i p_i \times \log_2 p_i$. A entropia é medida em *bits* usando logaritmos na base 2. Algumas propriedades da função entropia de uma variável aleatória que pode aceitar v valores distintos são:¹

1. $H(A) \in [0, \log_2(v)]$;

¹É assumido que $0 \times \log_2(0) = 0$.

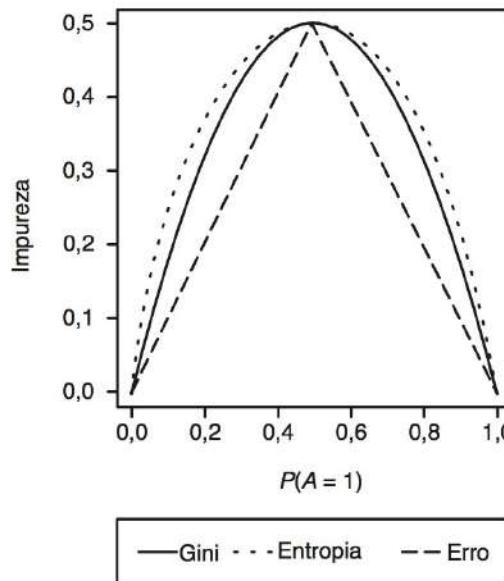


Figura 6.2 Gráfico da entropia, índice Gini e da taxa de erro de uma variável booleana aleatória A .

2. $H(A)$ tem um máximo igual a $\log_2(v)$ se $p_i = p_j, \forall i \neq j$;
3. $H(A)$ tem um mínimo igual a 0 se $\exists i: p_i = 1$.

Suponha uma variável aleatória booleana A . A função entropia, $H(A)$, é $-p \times \log_2(p) - (1-p) \times \log_2(1-p)$, em que p é a probabilidade de observar $A = 0$ e $1-p$ é a probabilidade de observar $A = 1$. A Figura 6.2 mostra o gráfico da entropia para esse caso.

No contexto de uma árvore de decisão, entropia é usada para medir a aleatoriedade (dificuldade para predizer) do atributo alvo. A cada nó de decisão, o atributo que mais reduz a aleatoriedade da variável alvo será escolhido para dividir os dados. Dado um conjunto de exemplos classificados, qual atributo selecionar como teste de divisão? Os valores de um atributo definem partições no conjunto de exemplos. Para cada atributo, o *ganho de informação* mede a redução na entropia nas partições obtidas de acordo com os valores do atributo. Informalmente, o ganho de informação é dado pela diferença entre a entropia do conjunto de exemplos e a soma ponderada da entropia das partições. A construção da árvore de decisão é guiada pelo objetivo de reduzir a entropia, isto é, a aleatoriedade (dificuldade para predizer) da variável alvo.

Considerando uma árvore de decisão uma fonte de informação que envia uma mensagem a respeito da classificação de um objeto e sendo que p e q denotam o número de objetos de duas classes diferentes, o conteúdo da informação da mensagem esperado é:

$$H(p, q) = -\frac{p}{p+q} \log\left(\frac{p}{p+q}\right) - \frac{q}{p+q} \log\left(\frac{q}{p+q}\right) \quad (6.1)$$

em que a probabilidade de cada possível mensagem é computada a partir do conjunto de treinamento. Se o atributo A é selecionado, e assumindo que o domínio de A tem v diferentes valores, a árvore resultante tem um conteúdo de informação esperado de:

$$E(A, p, q) = \sum_{i=1}^v \frac{p_i + q_i}{p+q} H(p_i, q_i) \quad (6.2)$$

em que p_i e q_i são o número de objetos de cada classe na subárvore associada com a partição i baseada nos valores do atributo A . O ganho de informação (IG) alcançado é

$$IG(A, p, q) = I(p, q) - E(A, p, q) \quad (6.3)$$

A heurística correspondente seleciona o atributo que resulta no máximo ganho de informação para aquele passo.

Muitas vezes um teste em um atributo nominal irá dividir os dados em tantos subconjuntos quantos os valores do atributo, embora divisões binárias baseadas na relação de pertinência a um subconjunto ($att_i \in \{V_j, \dots, V_k\}$ e $(att_i \notin \{V_j, \dots, V_k\})$) também sejam possíveis.

Exemplo Ilustrativo. Suponha o conjunto de treinamento apresentado na Tabela 6.1. O problema de decisão é decidir quando alguém joga ou não algum esporte dadas as condições do tempo. O problema é definido por quatro atributos de entrada: *Tempo*, *Temperatura*, *Umidade* e *Vento*. O conjunto de treinamento contém 14 exemplos que descrevem observações factuais do comportamento do indivíduo (coluna *Joga*), dadas as condições do tempo. A primeira variável (atributo), *Tempo*, é qualitativa. O domínio dessa variável é o conjunto: {Chuvoso, Ensolarado, Nublado}. As variáveis *Temperatura* e *Umidade* são quantitativas. Seu domínio é um subconjunto de \mathbb{R} . A variável *Vento* é booleana. O domínio da variável alvo, a variável que queremos predizer, é o conjunto: {Não, Sim}.

Tabela 6.1 Exemplo de conjunto de dados

Tempo	Temperatura	Umidade	Vento	Joga
Chuvoso	71	91	Sim	Não
Ensolarado	69	70	Não	Sim
Ensolarado	80	90	Sim	Não
Nublado	83	86	Não	Sim
Chuvoso	70	96	Não	Sim
Chuvoso	65	70	Sim	Não
Nublado	64	65	Sim	Sim
Nublado	72	90	Sim	Sim
Ensolarado	75	70	Sim	Sim
Chuvoso	68	80	Não	Sim
Nublado	81	75	Não	Sim
Ensolarado	85	85	Não	Não
Ensolarado	72	95	Não	Não
Chuvoso	75	80	Não	Sim

Qual atributo melhor discrimina as classes? Nos exemplos há cinco observações em que a variável alvo recebe o valor *Não* e nove exemplos em que o valor *Sim* foi observado. A entropia da classe para o conjunto de exemplos é:

$$p(Joga = Sim) = 9/14$$

$$p(Joga = Não) = 5/14$$

$$H(Joga) = -9/14 * \log_2(9/14) - 5/14 * \log_2(5/14) = 0,940 \text{ bit}$$

Calcular o Ganho de Informação para um Atributo Nominal. Considere o atributo *Tempo*. Dividindo o conjunto de treinamento pelos valores desse atributo, obteremos três partições. Para calcular a informação das partições, devemos estimar (a partir do conjunto de treinamento) as probabilidades de observar uma classe dado cada valor do atributo. A Tabela 6.2 resume as contagens necessárias. A partir dessa tabela, obtemos a entropia de cada partição e o ganho da divisão:

$$p(\text{Jogar} = \text{Sim} | \text{Tempo} = \text{Ensolarado}) = 2/5$$

$$p(\text{Jogar} = \text{Não} | \text{Tempo} = \text{Ensolarado}) = 3/5$$

$$H(\text{Jogar} | \text{Tempo} = \text{Ensolarado}) = -2/5 * \log_2(2/5) - 3/5 * \log_2(3/5) = 0,971 \text{ bit}$$

Similarmente, para as outras partições:

$$H(\text{Jogar} | \text{Tempo} = \text{Nublado}) = 0,0 \text{ bit}$$

$$H(\text{Jogar} | \text{Tempo} = \text{Chuvoso}) = 0,971 \text{ bit}$$

A entropia ponderada para o atributo *Tempo* é:

$$H(\text{Tempo}) = 5/14 * 0,971 + 4/14 * 0 + 5/14 * 0,971 = 0,693 \text{ bit}$$

O ganho de informação obtido pela divisão do conjunto de exemplos usando os valores do atributo *Tempo* é:

$$IG(\text{Tempo}) = 0,940 - 0,693 = 0,247 \text{ bit}$$

O que esse resultado significa? Antes de dividir os exemplos, conhecendo o valor do atributo *Tempo*, necessitamos de menos bits para codificar o valor do atributo alvo.

Tabela 6.2 Distribuição dos valores da classe pelos valores do atributo *Tempo*

	Ensolarado	Nublado	Chuvoso
Sim	2	4	3
Não	3	0	2

Calcular o Ganho da Informação para um Atributo Contínuo. O domínio de um atributo contínuo é um subconjunto do \mathbb{R} . Ele contém um número infinito de valores. A estratégia usada no caso de atributos nominais não é aplicada a atributos contínuos. A estratégia usual pesquisa por uma partição binária do conjunto de treinamento:

- Conjunto dos exemplos em que o *atributo* \leq *valor*
- Conjunto dos exemplos em que o *atributo* $>$ *valor*

Considere o teste $\text{Temperatura} = 70,5$. Esse teste dividirá os exemplos em duas partições: exemplos em que $\text{Temperatura} \leq 70,5$ e exemplos em que $\text{Temperatura} > 70,5$. A Figura 6.3 mostra a distribuição das classes em cada partição.

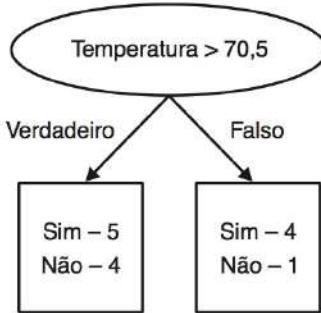


Figura 6.3 Partições usando o teste $\text{Temperatura} > 70,5$.

Como medir o ganho de informação para essa partição? A partir de cada partição, estimamos as probabilidades condicionais:

$$\begin{aligned} p(\text{Joga} = \text{Sim} | \text{Temperatura} \leq 70,5) &= 4/5 \\ p(\text{Joga} = \text{Não} | \text{Temperatura} \leq 70,5) &= 1/5 \\ p(\text{Joga} = \text{Sim} | \text{Temperatura} > 70,5) &= 5/9 \\ p(\text{Joga} = \text{Não} | \text{Temperatura} < 70,5) &= 4/9 \end{aligned}$$

A informação das partições é:

$$\begin{aligned} H(\text{Joga} | \text{Temperatura} \leq 70,5) &= -4/5 * \log_2(4/5) - 1/5 * \log_2(1/5) = 0,721 \text{ bit} \\ H(\text{Joga} | \text{Temperatura} > 70,5) &= -5/9 * \log_2(5/9) - 4/9 * \log_2(4/9) = 0,991 \text{ bit} \\ H(\text{Temperatura}) &= 5/14 * 0,721 + 9/14 * 0,991 = 0,895 \text{ bit} \\ IG(\text{Temperatura}) &= 0,940 - 0,895 = 0,045 \text{ bit} \end{aligned}$$

Um teste em um atributo contínuo dividirá os dados em dois subconjuntos: $\text{atributo} > \text{valor}$ e $\text{atributo} \leq \text{valor}$. Para obter o ponto de corte, os valores do atributo contínuo são primeiro ordenados. O ponto médio entre dois valores consecutivos é um possível ponto de corte e é avaliado pela função mérito. O possível ponto de corte que maximiza a função mérito é escolhido. Note-se que não é necessário testar todos os possíveis pontos de corte. Fayyad e Irani (1992) mostraram que, entre todos os possíveis pontos de corte, aqueles que maximizam qualquer função de mérito convexa dividem exemplos de classes diferentes.

O primeiro candidato para o ponto de corte é 64,5, e o último candidato a ponto de corte é 84. Considerando o ponto de corte 70,5, a Figura 6.3 mostra as distribuições de classe em cada partição.

Breiman et al. (1984) propuseram a função Gini para medir a impureza, definida como:

$$i(t) = 1 - \sum_i p_i^2 \tag{6.4}$$

em que p_i é a probabilidade para cada classe. Quando um atributo é examinado, a impureza média ponderada dos nós descendentes implícitos é subtraída de $i(t)$ e o atributo que resulta na maior diminuição da impureza é selecionado.

Mingers (1989b) desenvolve uma comparação empírica entre diversos critérios de divisão. Ele conclui:

“Os resultados mostram que a escolha da medida afeta o tamanho da árvore,

mas não sua precisão, que permanece a mesma ainda quando atributos são selecionados aleatoriamente.”

Mais tarde, diversos estudos (Buntine e Niblett, 1992; Esposito et al., 1997) questionaram o desenvolvimento da metodologia usada por Mingers (1989b). Nos estudos citados, a conclusão é que a divisão aleatória conduz a um erro aumentado. No entanto, não há um critério de divisão que seja sistematicamente o melhor de todos.

Regras de Divisão para Regressão

Em problemas de regressão, a função de custo a minimizar é, usualmente, o erro quadrático. Como já referimos, a média é a constante que minimiza o erro quadrático. Por esse motivo, a constante associada às folhas de uma árvore de regressão é a média dos valores do atributo alvo dos exemplos de treinamento que caem na folha. A construção de uma árvore de regressão é em tudo semelhante à construção de uma árvore de regressão, tendo em conta a função de custo referida.

Para estimar o mérito de uma partição obtida por um teste no valor de uma variável, Breiman et al. (1984) propuseram a métrica SDR (do inglês *Standard Deviation Reduction*). Assuma um conjunto de exemplos \mathbf{D} , com n exemplos. A variância da variável alvo, \mathbf{y} , é dada pela expressão:

$$sd(\mathbf{D}, \mathbf{y}) = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2} \quad (6.5)$$

Consideremos um teste hipotético h_A sobre o atributo A , por exemplo $A \leq a_1$. Os exemplos do conjunto \mathbf{D} serão divididos em dois subconjuntos, \mathbf{D}_L e \mathbf{D}_R , com tamanhos n_L e n_R , tais que $n = n_L + n_R$. A variância de \mathbf{y} , a variável alvo, em cada subconjunto \mathbf{D}_L e \mathbf{D}_R , é sempre menor ou igual à variância de \mathbf{y} antes da divisão. Podemos estimar a redução em variância obtida pela aplicação do teste h_A :

$$SDR(h_A) = sd(\mathbf{D}, \mathbf{y}) - \frac{n_L}{n} \times sd(\mathbf{D}_L, \mathbf{y}) - \frac{n_R}{n} \times sd(\mathbf{D}_R, \mathbf{y}) \quad (6.6)$$

Para cada atributo, e para cada possível teste no valor do atributo, é calculada a redução da variância associada a esse teste. O teste que provoca uma maior redução em variância é escolhido como teste para o nó.

6.1.2 Estratégias de Poda

Poda é considerada a parte mais importante do processo de construção da árvore, pelo menos em domínios com ruídos. Dados ruidosos levantam dois problemas. O primeiro é que as árvores induzidas classificam novos objetos em um modo não confiável. Estatísticas calculadas nos nós mais profundos de uma árvore têm baixos níveis de importância devido ao pequeno número de exemplos que chegam nesses nós. Nós mais profundos refletem mais o conjunto de treinamento (superajustamento) e aumentam o erro devido à variância do classificador. O segundo é que a árvore induzida tende a ser grande e, portanto, difícil para compreender. Podar uma árvore, que é trocar nós profundos por folhas, ajuda a minimizar esses problemas.

Podar uma árvore de decisão quase certamente irá causar a classificação incorreta de alguns exemplos do conjunto de treinamento. A vantagem da poda torna-se aparente quando se classificam novos exemplos não usados no processo de construção da árvore. Podar, em geral, leva a erros de generalização menores. Métodos de poda podem ser divididos em dois grupos principais. Primeiro, métodos que param a construção da árvore quando algum critério é satisfeito, referenciados como *pré-poda*, e segundo, métodos que constroem uma árvore completa e a podam posteriormente, referenciados como *pós-poda*. Todos os métodos mantêm um *ponto de equilíbrio* entre o tamanho da árvore e uma estimativa da taxa de erro (ver Capítulo 9). Tal como as estimativas de erro, podemos diferenciar entre métodos que estimam o erro a partir do *conjunto de treinamento*, geralmente conhecido como *erro de ressubstituição*, e métodos que usam um *conjunto de poda* separado, não usado na construção da árvore, para estimar a taxa de erro.

Pré-poda

Pré-poda conta com regras de parada que previnem a construção daqueles ramos que não parecem melhorar a precisão preditiva da árvore. A pré-poda tem a vantagem de que o tempo não é perdido construindo uma estrutura que não é usada na árvore final. No Algoritmo 6.1, a condição de parada é testada no passo 2. Esposito et al. (1997) apresentam diversas regras de parada comumente usadas:

1. Todas as observações alcançando um nó pertencem à mesma classe.
2. Todas as observações alcançando um nó têm o mesmo vetor de características (mas não necessariamente pertencem à mesma classe).
3. O número de observações no nó é menor que um certo limiar.
4. O mérito atribuído a todos os possíveis testes que particionam o conjunto de observações no nó é muito baixo.

As regras 1 e 2 são universalmente aceitas. As outras regras foram criticadas porque os testes do limiar muitas vezes terminam o procedimento de divisão prematuramente. Pode ser que todos os atributos pareçam individualmente sem sentido, mas algumas combinações deles podem ser altamente discriminantes (Esposito et al., 1993). Breiman et al. (1984) também apontam que tais regras de parada não são fáceis de se obter diretamente: limiares muito altos podem terminar a divisão antes que seus benefícios se tornem evidentes, enquanto limiares muito baixos resultam em árvores grandes e com erro de generalização pobre.

Por essa razão, não exploraremos mais a pré-poda, embora na literatura se encontrem mais critérios de terminação, usando formas de parada não triviais.

Pós-poda

Esse é o método mais comum de poda de árvores de decisão. Ele foi exaustivamente descrito na literatura (Breiman et al., 1984; Quinlan, 1993). Uma árvore completa, superajustada aos dados de treinamento, é gerada e podada posteriormente. Quinlan (1988) aponta que “*Construir e podar uma árvore é mais lento, mas mais confiável*”. A decisão

chave na pós-poda é podar ou não uma subárvore. Um dos métodos mais simples é baseado em duas medidas (Bratko, 1984): o *erro estático* e o *erro de backed-up*. O *erro estático* é o número de classificações incorretas considerando que todos os exemplos que chegam nesse nó são classificados usando a classe majoritária da distribuição de classes desse nó. O *erro de backed-up* é a soma das classificações incorretas de todas as subárvores do nó corrente. Se o *erro de backed-up* é maior ou igual ao *erro estático*, então o nó é trocado por uma folha com a classe majoritária do nó.

A poda *custo de complexidade* foi apresentada por Breiman et al. (1984), e é um dos métodos mais utilizados. Inicialmente é gerada uma árvore completa. Com base nessa árvore é gerada uma sequência de árvores cada vez menores, sendo escolhida uma das subárvores. É baseada em dois parâmetros: a taxa de erro $R(T)$ e o tamanho da árvore, $|T|$, medido em termos das folhas. A medida de *custo-complexidade* para a árvore é:

$$R_\alpha(T) = R(T) + \alpha|T| \quad (6.7)$$

em que α é um parâmetro que pesa a importância relativa do tamanho da árvore em relação à taxa de erro. Para cada valor de α , o objetivo é encontrar a subárvore $|T_\alpha| \leq |T|$ que minimiza $R_\alpha(T)$. Se α é pequeno, a penalidade por ter um grande número de nós terminais é menor e T_α será grande. Quando α aumenta, a subárvore minimizada terá poucos nós terminais. Ainda que α execute através de valores contínuos, o número de subárvores de T é finito. Então o processo de poda produz uma sequência finita de subárvores aninhadas T_1, T_2, T_3, \dots com nós terminais progressivamente menores. A árvore podada selecionada é aquela que minimiza $R_\alpha(T)$ na sequência de subárvores. O livro CART (Breiman et al., 1984) apresenta um algoritmo eficiente para implementar esse processo de poda.

A poda pessimista (Quinlan, 1993) estima, para cada subárvore, um erro aparente, baseado na proporção dos exemplos de treinamento classificados incorretamente nas folhas. Ela usa o mesmo conjunto de treinamento para gerar e podar a árvore. A taxa de erro estimada no conjunto de treinamento é uma estimativa *otimista* e não provê o melhor critério para escolher a melhor árvore. Por essa razão, Quinlan introduz a continuidade da correção baseada na distribuição binomial dos erros. Uma subárvore é podada e substituída por uma folha quando a taxa de erro para a subárvore não é显著antemente menor que o erro da folha. Como uma poda pessimista usa uma estratégia de cima para baixo (*top-down*) e não requer um conjunto de poda separado, sua principal vantagem é a velocidade. A poda pessimista é o método usado no C4.5 (Quinlan, 1988). Ela usa a informação do conjunto de treinamento para construir e simplificar a árvore usando uma estratégia transversal pós-ordem de baixo para cima (*bottom-up*).

Considere um nó gerado a partir de n exemplos de treinamento. Assumindo a classe majoritária como representando esse nó, os exemplos que não são da classe majoritária seriam classificados em erro. Seja e o número de erros na amostra de n exemplos. Assuma que a verdadeira probabilidade de erro é q e que os n exemplos são gerados por um processo Bernoulli com parâmetro q . Não conseguimos calcular q , mas conseguimos obter um intervalo de confiança $[L_c, U_c]$ que, para um nível de confiança c , contém q .

Dado um nível de confiança c , encontramos os limites z , tais que: $P[X \geq z] = c$. Considerando o conjunto dos exemplos abrangidos por uma folha como uma amostra estatística, é possível estimar um intervalo de confiança $[L_c, U_c]$ relativo à probabilidade de classificação incorreta da folha. O limite superior do intervalo de confiança, U_c , é de

particular interesse para a análise do pior caso. Partindo do pressuposto de que os erros no conjunto de treinamento seguem uma distribuição binomial com probabilidade q em n experimentos, é possível calcular o valor de U_c :

$$P \left[\frac{e - q}{\sqrt{q(1-q)/n}} > z \right] = c$$

para um nível de confiança c fornecido pelo usuário, e em que z é o número de desvios padrões correspondentes ao nível de confiança c . Para um nível de confiança de 25%, z é um valor tabelado e igual a 0,69. O limite superior desse intervalo de confiança é dado pela expressão:

$$U_c = \frac{e + \frac{z^2}{2n} + z\sqrt{e/n - e^2/n + z^2/(4n)}}{1 + z^2/n} \quad (6.8)$$

Tendo encontrado o limite superior, a estimativa de erro para as folhas e subárvore são calculadas assumindo que serão usadas para classificar um conjunto de casos não conhecidos do mesmo tamanho que o conjunto de treinamento. Assim, a taxa de erro predita para uma folha será $\#exemplos \times U_c$. A soma das taxas de erro preditas para todas as folhas em um ramo é considerada uma estimativa da taxa de erro do próprio ramo. Dessa forma, comparando a taxa de erro predita para um dado nó, como se ele fosse trocado por uma folha, com a subárvore da raiz até esse nó, podemos decidir se é conveniente podar ou manter o nó. Além disso, comparando com o erro de predição do sub-ramo da raiz até os filhos desse nó, podemos decidir por inserir o sub-ramo.

Há duas suposições fortes subjacentes a esse método de poda. É difícil aceitar que os exemplos do conjunto de treinamento abrangidos por um nó representam uma amostra estatística, uma vez que a árvore foi construída para ajustar a esses dados da melhor forma possível. De acordo com Esposito et al. (1997), a suposição de que erros em um exemplo têm uma distribuição *binomial* é mais questionável.

Mingers (1989a) desenvolve uma comparação empírica de cinco métodos de poda para indução de uma árvore de decisão. Ele conclui que “*Os resultados mostram que dois métodos – complexidade do custo e erro reduzido – têm bom desempenho*”. O autor também mostra que não há interação significante entre o método usado para gerar um árvore e métodos de poda. Esposito et al. (1993) declaram e justificam que a metodologia experimental usada por Mingers é injusta. Tal como as regras de divisão, a poda é um domínio em que nenhuma proposta existente é a melhor para todos os casos. A poda é um envisagemento em direção à simplicidade. Se o domínio do problema admite soluções simples, então a poda é uma opção eficiente (Schaffer, 1993).

6.1.3 Valores desconhecidos

Quando se usa uma árvore como um classificador, o exemplo a ser classificado passa através da árvore. A cada nó um teste baseado nos valores dos atributos é executado. Se o valor de atributo testado não é conhecido (frequentemente em dados reais alguns valores de atributos são desconhecidos ou indeterminados), o procedimento pode não determinar o percurso a seguir. Uma vez que uma árvore de decisão constitui uma hierarquia de testes, o problema do valor desconhecido tem relevância especial nesse tipo de classificadores.

Várias soluções foram propostas na literatura. Quinlan (1986) examinou algumas delas. As mais comuns são:

- Uma estratégia simples consiste em trocar o valor não conhecido pelo valor mais comum para o atributo encontrado no conjunto de treinamento, também discutida na Capítulo 3.
- Outra estratégia consiste em considerar o valor desconhecido como outro valor possível do atributo (Kohavi et al., 1997). Quando se constrói a árvore, cada nó de decisão pode conter um ramo para o caso em que o atributo testado leva a um valor desconhecido.
- O C4.5 (Quinlan, 1993) usa uma estratégia mais complexa. Associa-se uma probabilidade a cada um dos possíveis valores do atributo. As probabilidades são estimadas com base nas frequências observadas dos valores para o atributo nos exemplos do nó corrente. Esses valores fracionários são usados para calcular o *ganho de informação* desse atributo. Quando se classifica um exemplo de teste, C4.5 passa o exemplo através de todos os ramos em que o valor do atributo desconhecido foi detectado. Cada ramo produz como saída um voto para a classe. A saída final é calculada como a classe majoritária de todas as saídas dos ramos.
- CART (Breiman et al., 1984) usa uma estratégia mais sofisticada conhecida como *divisão substituta*. Em vez de armazenar, para cada nó, somente o atributo que minimiza a função de impureza, CART armazena os atributos que produzem uma divisão similar, ordenados pelo critério de impureza. Quando se procede à classificação de um exemplo, se nesse exemplo o valor do atributo testado no nó da árvore é desconhecido, o CART irá procurar, na lista ordenada de atributos alternativos, o primeiro atributo cujo valor é conhecido no exemplo.

6.1.4 Discussão: Vantagens e Desvantagens

Árvores de decisão têm inúmeras vantagens. Elas são um dos algoritmos mais comumente usados, quer em aplicações do mundo real quer no meio acadêmico. Alguns dos pontos mais positivos referenciados na literatura são:

1. *Flexibilidade*

Árvores de decisão não assumem nenhuma distribuição para os dados. Elas são métodos não paramétricos. O espaço de objetos é dividido em subespaços, e cada subespaço é ajustado com diferentes modelos. Uma árvore de decisão fornece uma cobertura exaustiva do espaço de instâncias. Havendo exemplos suficientes, pode aproximar o *erro de Bayes* de qualquer função.

2. *Robustez*

Árvores univariáveis são invariantes a transformações (estritamente) monótonas de variáveis de entrada. Por exemplo, usar \mathbf{x}_j , $\log \mathbf{x}_j$, ou $e^{\mathbf{x}_j}$ como a j -ésima variável de entrada produz árvores com a mesma estrutura. Como uma consequência dessa invariância, a sensibilidade a distribuições com grande cauda e *outliers* é também reduzida (Friedman, 1999).

3. Seleção de atributos

O processo de construção de uma árvore de decisão seleciona os atributos a usar no modelo de decisão. Essa seleção de atributos produz modelos que tendem a ser bastante robustos contra a adição de atributos irrelevantes e redundantes.

4. Interpretabilidade

Decisões complexas e globais podem ser aproximadas por uma série de decisões mais simples e locais. Todas as decisões são baseadas nos valores dos atributos usados para descrever o problema. Ambos os aspectos contribuem para a popularidade das árvores de decisão.

5. Eficiência

O algoritmo para aprendizado de árvore de decisão é um algoritmo guloso que é construído de cima para baixo (*top-down*), usando uma estratégia dividir para conquistar sem *backtracking*. Sua *complexidade de tempo* é linear com o número de exemplos.

Apesar das vantagens já mencionadas, alguns problemas conhecidos referenciados na literatura de AM incluem:

1. Replicação

O termo refere-se à duplicação de uma sequência de testes em diferentes ramos de uma árvore de decisão, levando a uma representação não concisa, que também tende a ter baixa precisão preditiva (ver Capítulo 9). Por exemplo, para representar o conceito $(A \wedge B) \vee (C \wedge D)$, um dos subconceitos ($A \wedge B$ ou $C \wedge D$) tem de ser duplicado. Suponha que a árvore escolha para raiz um teste no atributo A , então o conceito $(C \wedge D)$ tem de aparecer nas subárvores descendentes. Pagallo e Haussler (1990) argumentam que a *replicação* é inerente à representação da árvore de decisão.

2. Valores ausentes

Uma árvore de decisão é uma hierarquia de testes. Se o valor de um atributo é desconhecido, isso causa problemas em decidir que ramo seguir. Algoritmos devem empregar mecanismos especiais para abordar falta de valores. Friedman et al. (1996) sustentam que “*Cerca de metade do código no CART e 80% dos esforços de programação foram desenvolvidos para falta de valores!*”.

3. Atributos contínuos

O gargalo do algoritmo é a presença de atributos *contínuos*. Nesse caso, uma operação de *ordenação* é solicitada para cada atributo *contínuo* de cada nó de decisão. Alguns autores estimam que a operação de ordenação consuma 70% do tempo necessário para induzir uma árvore de decisão em grandes conjuntos de dados com muitos atributos contínuos (Catlett, 1991). Devido a essa observação, alguns pesquisadores (Catlett, 1991; Fayyad e Irani, 1993) examinaram a possibilidade de discretização de atributos contínuos.

4. Instabilidade

Muitos pesquisadores, especialmente Breiman et al. (1984), Breiman (1996b) e Kohavi e Kunz (1997) apontaram que pequenas variações no conjunto de treinamento podem produzir grandes variações na árvore final. A cada nó, o critério de

mérito de divisão classifica os atributos, e o melhor atributo é escolhido para dividir os dados. Se dois ou mais atributos são classificados similarmente, pequenas variações da classificação dos dados podem alterar a classificação. Todas as subárvores abaixo desse nó mudam. A estratégia da partição recursiva implica que a cada divisão que é feita o dado é dividido com base no atributo de teste. Depois de algumas divisões, há usualmente muitos poucos dados nos quais a decisão se baseia. Há uma forte tendência a inferências feitas próximo das folhas serem menos confiáveis que aquelas feitas próximas da raiz.

6.2 Regras de Decisão

Uma regra de decisão é uma implicação da forma: **se A então B** . A parte condicional A é uma conjunção de condições. Cada condição é definida por uma relação entre um atributo e os valores do domínio. A relação pode ser $=, <, >, \leq, \geq$ ou \in . Por exemplo, pode assumir a forma de: $Atributo_i = valor_i, Atributo_i \leq valor_i, Atributo_i \in Conjunto$ etc, em que $valor_i$ pertence ao domínio do atributo. Um exemplo de uma regra de decisão é: $Tempo = \text{Ensolarado} \wedge \text{Umidade} \leq 75 \Rightarrow \text{Jogar} = \text{Sim}$. Tal como nas árvores de decisão, o conjunto de regras é disjunto (FND).

$$\text{regra}_1 \text{ ou } \text{regra}_2 \text{ ou } \dots \text{ ou } \text{regra}_n$$

Regras de decisão e árvores de decisão são bastante similares em suas formas de representação para expressar generalizações dos exemplos. Ambas definem superfícies de decisão similares. As superfícies de decisão definidas pelas regras de decisão correspondem a hiper-retângulos no espaço definido pelos atributos. Dois exemplos de superfícies de decisão esboçados por conjuntos de regras são apresentados na Figura 6.4.

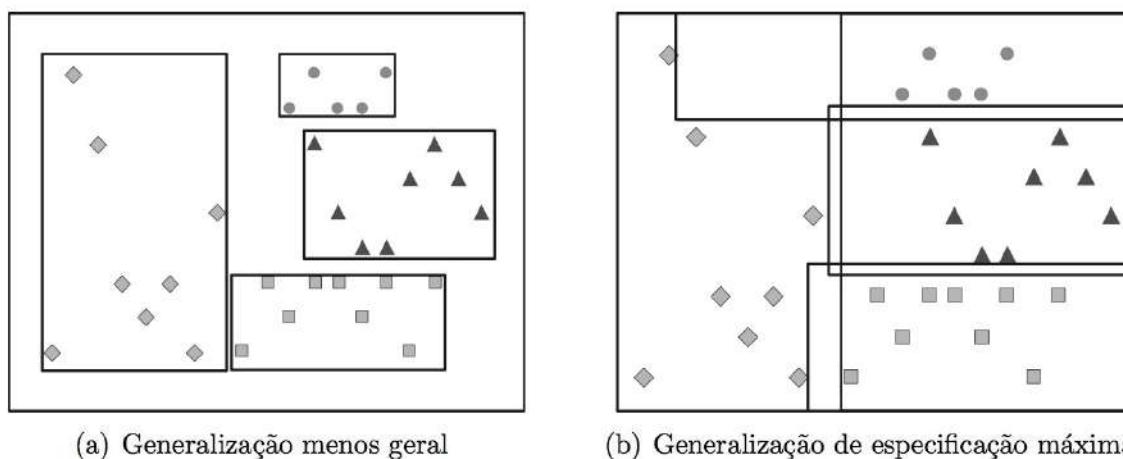


Figura 6.4 Dois exemplos da superfície de decisão desenhadas por um conjunto de regras.

6.2.1 Por que Regras de Decisão?

Como as árvores de decisão cobrem todo o espaço de instâncias, a vantagem é que qualquer exemplo é classificado por uma árvore de decisão. Entretanto, cada teste em

um nó tem um contexto definido por testes anteriores, definidos nos nós no caminho, que podem ser problemáticos se levarmos em conta a interpretabilidade. Por outro lado, as regras são modulares, ou seja, podem ser interpretadas isoladamente.

Cada regra cobre uma região específica do espaço de instâncias. A união de todas as regras pode ser menor que o Universo. As características das representações das regras são ilustradas na Figura 6.4. Em comparação com as árvores, as regras de decisão:

- Removem condições em uma regra sem remover uma outra regra.
- Perdem a distinção entre testes perto da raiz e perto das folhas.

Um exemplo claro das desvantagens da hierarquia de testes imposta por uma árvore de decisão é conhecido como *fragmentação do conceito*. Deixamos como exercício ao leitor obter uma árvore de decisão para representar o conceito: $(A \wedge B) \vee (C \wedge D)$. Como se pode verificar facilmente, qualquer árvore de decisão duplica um dos conceitos, $(A \wedge B)$ ou $(C \wedge D)$.

O Algoritmo OneR

Holte (1993) propôs um algoritmo, OneR (do inglês *One Rule*), que gera regras baseadas num único atributo. Seu trabalho experimental revela que as regras muito simples, como as geradas pelo OneR, são competitivas com métodos muito mais sofisticados.

O OneR induz uma árvore de decisão de um nível, ou seja, regras que testam um único atributo. O algoritmo básico consiste em, para cada atributo, considerar uma regra por cada valor desse atributo. Considerando os exemplos de treinamento, cada uma dessas regras prevê a classe majoritária dos exemplos para os quais o atributo toma esse valor. Ou seja, cada atributo define um conjunto de regras, uma regra para cada valor desse atributo. Para cada atributo, é calculada a taxa de erro, e o atributo com menor taxa de erro é escolhido.

6.2.2 De Árvores de Decisão às Regras de Decisão

Árvores de decisão extensas são de difícil compreensão porque o teste de decisão em cada nó aparece em um contexto específico, definido pelo resultado de todos os testes nos nós antecedentes. O trabalho desenvolvido por Rivest (1987) apresenta as *listas de decisão*, uma nova representação para a generalização de exemplos que estende as árvores de decisão. A grande vantagem dessa representação é a modularidade do modelo de decisão e, consequentemente, a sua interpretabilidade: cada regra é independente das outras regras, e pode ser interpretada isoladamente das outras regras. Como consequência, a representação utilizando regras de decisão permite eliminar um teste em uma regra, mas reter o teste em outra regra. Além disso, como a conjunção de condições é comutativa, a distinção entre testes perto da raiz e testes perto das folhas desaparece.

Indução de Listas de Regras de Decisão. Existem vários algoritmos para indução de regras de decisão (Rivest, 1987; Clark e Niblett, 1989; Cohen, 1995; Domingos, 1996; Weiss e Indurkhya, 1998). Nesta seção, referimos os algoritmos que geram regras de decisão a partir de árvores de decisão, tal como é feito em Quinlan (1993).

Qualquer árvore de decisão pode ser facilmente reescrita em um conjunto de regras de decisão. Cada regra corresponde a um percurso desde a raiz da árvore até uma folha. Existem tantas regras quantas as folhas da árvore de decisão. Esse processo gera um conjunto de regras com a mesma complexidade da árvore de decisão. Contudo, alguns antecedentes em regras consideradas individualmente podem conter condições irrelevantes. **C4.5rules** (Quinlan, 1993, 1995) usa um processo de otimização para simplificar o conjunto de regras, removendo condições irrelevantes.

O processo de otimização consiste em duas fases. Primeiro, cada regra é generalizada pela eliminação de condições que não contribuem para discriminar as classes. É utilizada uma procura gulosa, em que em cada passo a regra é avaliada removendo uma das condições. A condição que produz um menor aumento da estimativa pessimista da taxa de erro é eliminada. Para obter a estimativa pessimista da taxa de erro, é utilizado um processo semelhante ao mecanismo de poda usado pelo C4.5. Após a generalização individual das regras, são removidas regras idênticas e as regras sem parte condicional. Em uma segunda fase, as regras são agrupadas pela classe que preveem. Para cada classe, o conjunto de regras é simplificado, eliminando as regras que não contribuem para a taxa de acerto do conjunto. O estudo experimental apresentado em Quinlan (1993) mostra que as regras de decisão são mais simples e com menor taxa de erro do que a árvore de decisão a partir da qual foram geradas.

Frank e Witten (1998) apresentam um método para gerar regras por transformação de árvores de decisão que não necessita recorrer a um processo de otimização global. A ideia base consiste em crescer uma árvore em largura, em vez de crescer a árvore em profundidade. Quando uma folha é encontrada, a regra de decisão correspondente a essa folha é extraída. Os exemplos cobertos pela folha são removidos, e iterativamente são geradas mais regras com os exemplos não cobertos pelas regras anteriores.

6.2.3 O Algoritmo da Cobertura

Nesta seção estudamos um dos algoritmos mais divulgados para aprender regras de decisão a partir de exemplos. O *algoritmo da cobertura* define o processo de aprendizado como um processo de procura: dados um conjunto de exemplos classificados e uma linguagem para representar generalizações dos exemplos, o algoritmo procede, para cada classe, uma procura heurística. Tipicamente, o algoritmo procura regras da forma: **se $Atributo_i = valor_j$ e $Atributo_l = valor_k \dots$ então $Classe_z$** . A procura pode proceder quer a partir da regra mais geral, ou seja, uma regra sem parte condicional, para regras mais específicas, acrescentando condições; ou a partir de regras muito específicas (por exemplo, um exemplo com restrições em todos os atributos) para regras mais gerais, eliminando restrições. O processo de procura é guiado por uma função de avaliação das hipóteses. Essa função estima a qualidade das regras que são geradas durante o processo. Uma possível função de avaliação é a taxa de erro, com os problemas que já referimos (Breiman et al., 1984). Outra função de avaliação é a informação mútua ou entropia condicional (Clark e Niblett, 1989).

Ideia Básica

Dado um conjunto de exemplos de classes diferentes, o *algoritmo de cobertura* consiste em aprender uma regra para uma das classes, removendo o conjunto de exemplos cobertos

pela regra (ou o conjunto de exemplos positivos), e repetir o processo. O processo termina quando só há exemplos de uma única classe. O algoritmo de cobertura é apresentado no Algoritmo 6.2.

Algoritmo 6.2 Algoritmo de Cobertura: construção de um conjunto de regras

Entrada: Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
Saída: Um conjunto de regras: **Regras**

- 1 **Regras** $\leftarrow \{\}$;
- 2 Seja \mathbf{Y} o conjunto das classes em \mathbf{D} ;
- 3 **para cada** $y_i \in \mathbf{Y}$ **faça**
- 4 **repita**
- 5 $Regra = \text{Aprende_Uma_Regra}(\mathbf{D}, y_i)$;
- 6 **Regras** $\leftarrow \text{Regras} \cup \{Regra\}$;
- 7 $\mathbf{D} \leftarrow \text{Remove exemplos cobertos pela Regra em } \mathbf{D}$;
- 8 **até** não haver exemplos de y_i ;
- 9 **fim**
- 10 **Retorna:** **Regras** ;

Encontrar uma regra é um problema de procura. Duas estratégias básicas de procura têm sido usadas. A primeira inicia a busca da regra mais geral, $\{\} \rightarrow Classe$, e aplica operadores de especificação, acrescentando condições à parte condicional da regra. É uma busca *top-down* e é orientada pelo modelo. O algoritmo para encontrar uma regra está apresentado no Algoritmo 6.3. É usada em sistemas como CN2 (Clark e Niblett, 1989). A segunda proposta se inicia pela regra mais específica (é escolhido um dos exemplos aleatoriamente, o que implica restrições em todos os atributos) e aplica operadores de generalização, removendo restrições. Essa proposta é *bottom-up* e orientada pelos dados. O algoritmo para encontrar uma regra está apresentado no Algoritmo 6.4. É usada, por exemplo, no algoritmo AQ (Michalski et al., 1986; Wnek e Michalski, 1994). A Figura 6.5 apresenta uma ilustração comparativa do funcionamento dos dois algoritmos.

O algoritmo de procura pode ser implementado quer por um algoritmo de subida em colina (*hill-climbing*), quer por um algoritmo de *beam-search*. Essa última alternativa requer mais recursos computacionais, mas tem revelado melhor desempenho em termos de taxa de acerto.

A qualidade de uma regra pode ser medida tendo em conta:

- **ncover:** número de exemplos cobertos pela regra;
- **ncorreto:** número de exemplos cobertos pela regra corretamente classificados por ela;
- **cobertura:** definida como $ncover/n$;
- **taxa de acerto:** definida como $ncorreto/ncover$.

Quando é acrescentada uma condição a uma regra, a regra torna-se mais específica e **ncover** diminui. Por outro lado, quando se remove uma condição, a regra fica mais geral, e **ncover** aumenta. Qualquer função cujo valor aumente com a cobertura e taxa de acerto

pode ser utilizada. Por exemplo, o sistema AQ (Wnek e Michalski, 1994) usa a seguinte função de avaliação: $AQ = \frac{ncorreto+1}{ncover+k}$, em que k é o número de classes.

Algoritmo 6.3 Algoritmo *top-down* para encontrar uma regra

Entrada: Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
 y : classe da regra

Saída: *Regra*: Uma Regra de classificação

1 Seja *Avs* o conjunto de *atributo_valores* em \mathbf{D} ;
2 *Regra* $\leftarrow \{\}$;
3 $v \leftarrow Avalia(Regra, \mathbf{D}, y)$;
4 *melhor* $\leftarrow v$;
5 *continua* \leftarrow Verdadeiro ;
6 **enquanto** *continua* **faça**
7 *continua* \leftarrow Falso ;
8 **para** **cada** $av_i \in Avs$ **faça**
9 $val \leftarrow Avalia(Regra \cup av_i, \mathbf{D}, y)$;
10 **se** $val < melhor$ **então**
11 *melhor* $\leftarrow val$;
12 *Cond* $\leftarrow av_i$;
13 *continua* \leftarrow Verdadeiro ;
14 **fim**
15 **fim**
16 **se** *continua* **então**
17 *Regra* $\leftarrow Regra \cup Cond$;
18 **fim**
19 **fim**
20 **Retorna:** *Regra* ;

Um dos sistemas de indução de regras de decisão mais populares é o sistema CN2, desenvolvido por Clark e Niblett (1989). Ele usa o algoritmo de cobertura, e a indução de uma regra utiliza o processo *top-down* descrito nesta seção. A função de avaliação de hipóteses usa o conceito de entropia.

Exemplo Ilustrativo: Considere novamente o conjunto de exemplos da Tabela 6.1, em que o objetivo é encontrar uma regra para *Joga* = Sim. Neste exemplo ilustrativo vamos usar como função de avaliação de hipóteses a taxa de erro. Obviamente, queremos minimizar essa função.

A procura inicia com a regra mais geral $\{\} \rightarrow \text{Sim}$. A regra não tem restrições, ou seja, tudo é da classe Sim. A taxa de erro é de 5/14. Introduzindo uma restrição, o conjunto de hipóteses é:

- Atributo *Tempo*
 - $\text{Tempo} = \text{Ensolarado} \rightarrow \text{Sim}; (3/5)$
 - $\text{Tempo} = \text{Nublado} \rightarrow \text{Sim}; (0/4)$

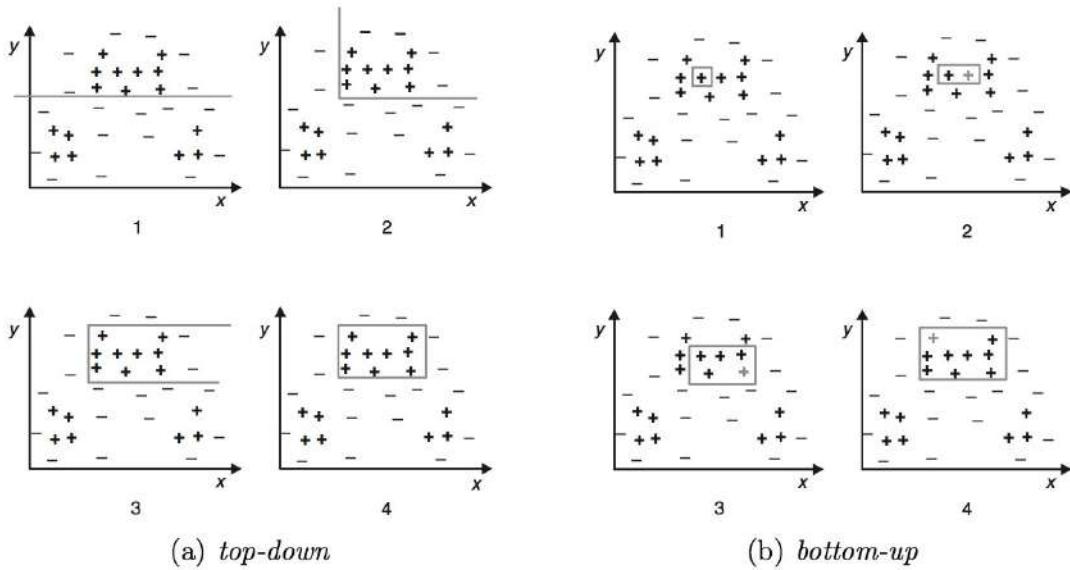


Figura 6.5 Exemplo ilustrativo do funcionamento do algoritmo para induzir uma regra.

- *Tempo* = Chuvoso → Sim; (2/5)
- Atributo *Temperatura*
 - *Temperatura* = Quente → Sim; (3/4)
 - *Temperatura* = Fresco → Sim; (1/6)
 - *Temperatura* = Frio → Sim; (1/4)
- Atributo *Umidade*
 - *Umidade* = Alta → Sim; (3/7)
 - *Umidade* = Normal → Sim; (2/7)
- Atributo *Vento*
 - *Vento* = Sim → Sim; (3/6)
 - *Vento* = Não → Sim; (2/8)

Como foi encontrada uma regra com taxa de erro 0, *Tempo* = Nublado → Sim, o processo de encontrar uma regra termina. O algoritmo de cobertura remove os exemplos cobertas pela regra² e retorna o processo de encontrar uma nova regra a partir do subconjunto de exemplos.

No sistema AQ (Michalski et al., 1986; Wnek e Michalski, 1994), que segue uma estratégia *bottom-up*, a procura é efetuada do mais específico para o geral (Algoritmo 6.4). Para construir uma regra para uma classe, o ponto de partida é um exemplo dessa classe. O operador de generalização remove condições da regra atual. No caso do sistema AQ, a função de avaliação é: $AQ = (n_{correto} + 1)/(n_{cover} + k)$, ou seja, usa a correção de Laplace para a taxa de acerto. Um exemplo ilustrativo da aplicação desse algoritmo é apresentado na Figura 6.6.

²Em algumas variantes, são removidos apenas os exemplos corretamente classificados pela regra.

Algoritmo 6.4 Algoritmo *Bottom-Up* para encontrar uma regra

Entrada: Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
 y : classe da regra

Saída: *Regra*: Uma Regra de classificação

1 Escolhe, aleatoriamente, um exemplo da classe y em \mathbf{D} ;
2 Seja *Regra* o conjunto de *atributo_valor* desse exemplo ;
3 $v \leftarrow Avalia(Regra, \mathbf{D}, y)$;
4 *melhor* $\leftarrow v$;
5 *continua* \leftarrow Verdadeiro ;
6 **enquanto** *continua* **faça**
7 *continua* \leftarrow Falso ;
8 **para cada** $av_i \in Regra$ **faça**
9 $val \leftarrow Avalia(Regra \setminus av_i, \mathbf{D}, y)$;
10 **se** $val < melhor$ **então**
11 $melhor \leftarrow val$;
12 $Cond \leftarrow av_i$;
13 *continua* \leftarrow Verdadeiro ;
14 **fim**
15 **fim**
16 **se** *continua* **então**
17 $Regra \leftarrow Regra \setminus Cond$;
18 **fim**
19 **fim**
20 **Retorna:** *Regra* ;

Aplicando Regras

A aplicação de um conjunto de regras para classificar um exemplo de teste consiste em verificar qual a regra cuja parte condicional é verificada e decidir de acordo com a conclusão da regra. No entanto, quando da classificação de um exemplo de teste, podemos ter as situações: *i*) nenhuma regra dispara; *ii*) apenas uma regra dispara; ou *iii*) mais que uma regra dispara. O primeiro caso pode ser evitado, acrescentando uma regra sem parte condicional e cuja conclusão seja, por exemplo, a classe majoritária. Se mais que uma regra dispara, poderemos ter situações de conflito. As soluções mais usuais para esses casos consiste em ordenar as regras por prevalência, ou qualquer critério de mérito.

Há uma diferença fundamental entre os dois algoritmos descritos anteriormente no que se refere ao seu uso na classificação de exemplos de teste. Enquanto o método *top-down* gera regras ordenadas pela ordem em que são induzidas, o método *bottom-up* gera um conjunto não ordenado de regras. Na aplicação do conjunto de regras a exemplos não classificados, há duas estratégias básicas. No caso de conjuntos ordenados de regras, cada exemplo é classificado pela primeira regra cuja parte condicional é satisfeita. Neste contexto é frequente adicionar uma regra *default* sem parte condicional, que se aplica quando nenhuma das regras dispara. Como vimos, o algoritmo de cobertura termina quando existem apenas exemplos de uma classe. A regra *default* tem como conclusão essa classe. No caso de conjuntos de regras não ordenadas, todas as regras cuja parte

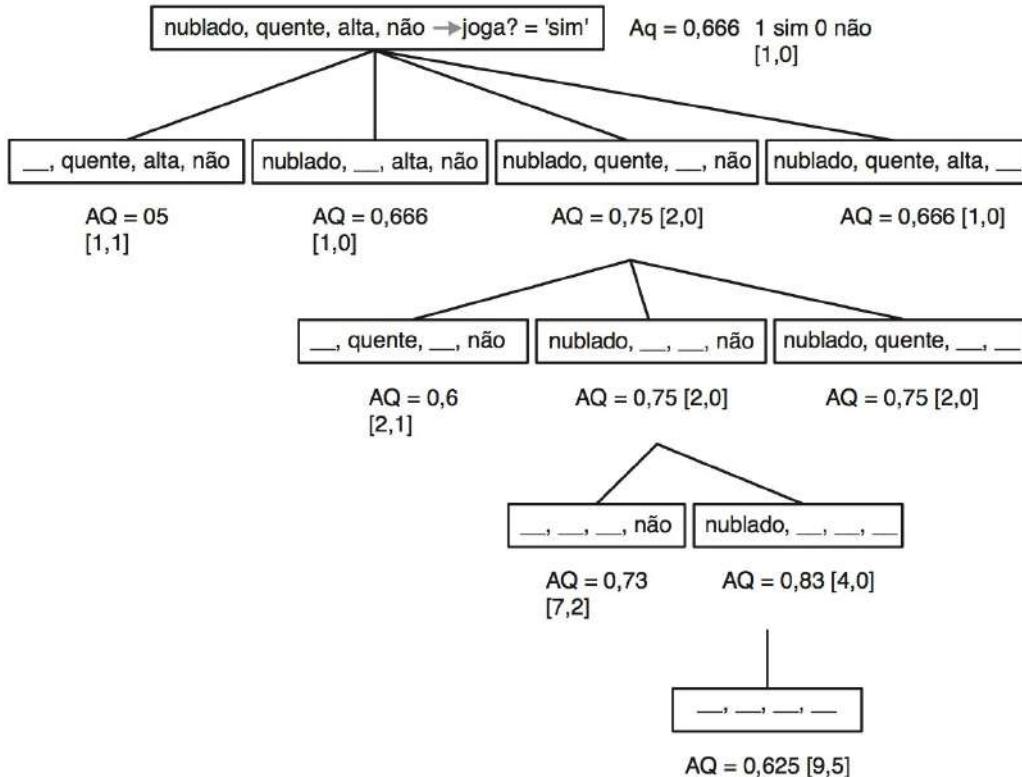


Figura 6.6 Exemplo do processo *bottom-up* de construção de uma regra.

condicional é verificada são utilizadas para classificar o exemplo, tipicamente por votação pesada pela qualidade da regra.

6.3 Outros Modelos para Árvores de Previsão

6.3.1 Árvores de Modelos

Árvores de regressão muitas vezes geram um grande número de nós, dificultando a interpretação dos dados. Apesar disso, o seu desempenho em muitos problemas costuma ser melhor do que uma simples equação de regressão. No livro de referência de Breiman et al. (1984), os autores escrevem:

“Na regressão, o uso de divisões baseadas em combinações lineares não tem o apelo que tem em classificação. (...) Uma alternativa promissora para a melhoria da precisão é crescer uma árvore pequena com apenas alguns dos nós mais significativos. Em seguida, fazer regressão linear múltipla em cada um dos nós terminais.”

Uma árvore de modelos (do inglês *model tree*) (Quinlan, 1992; Wang e Witten, 1997; Frank et al., 1998) é uma árvore que combina árvore de regressão com equações de regressão. Esse tipo de árvore funciona da mesma maneira que uma árvore de regressão, porém os nós folha contêm expressões lineares em vez de valores agregados (médias ou medianas). A estrutura da árvore divide o espaço dos atributos em subespaços, e os exemplos em cada um dos subespaços são aproximados por uma função linear. A *model tree* é

menor e mais compreensível que uma árvore de regressão e, mesmo assim, apresenta um erro médio menor na predição.

O M5 (Quinlan, 1992) é um indutor para *model tree*. Posteriormente, foi desenvolvido o sistema Cubist (Quinlan, 1998), uma ferramenta para geração de modelos preditivos baseados em regras. O algoritmo do Cubist usa o algoritmo do M5 para gerar uma árvore que é posteriormente transformada em regras. O modelo do Cubist consiste em uma coleção de regras não ordenadas da forma: **se** condições **então** modelo_linear. A regra indica que, se um caso satisfaz todas as condições, o modelo linear é apropriado para prever o valor do atributo objetivo. Se duas ou mais regras são aplicadas para um caso, é calculada uma média dos valores definidos pelos respectivos modelos lineares para chegar a uma previsão final. O Cubist utiliza um método não muito comum para a combinação. Inicialmente, são encontrados os cinco casos mais similares (*k*-vizinhos mais próximos) do exemplo de teste. Posteriormente, é calculada a média utilizando esses valores. Finalmente, o Cubist ajusta esses valores usando o modelo baseado em regras. Os modelos gerados pelo Cubist geralmente fornecem resultados melhores que os produzidos por técnicas como regressão linear multivariada, ao mesmo tempo que são mais fáceis de entender do que as redes neurais, por exemplo.

Por exemplo, no problema *Machine-Cpu* disponível no repositório da UCI (Blake et al., 1999), o problema consiste em estimar o desempenho de processadores. O modelo gerado pelo Cubist é:

```
IF MMAZ <= 14000 THEN
    ERP = 0,55 * CACH + 1,14 * CHMIN + 0,114 CHMAX + 3,15
```

em que o **ERP**, desempenho relativo estimado, é a variável objetivo, e os atributos de entrada são: **MMAZ**, a memória RAM, **CACH**, a memória *cache* em kbytes, **CHMIN**, o número mínimo de canais, **CHMAX**, o número máximo de canais.

6.3.2 Árvores de Opção

Os modelos usuais de árvores de decisão incorporam um único teste em cada nó, e cada exemplo segue um único caminho, correspondendo ao resultado do teste, até que uma folha seja alcançada e uma predição é então realizada. *Árvores de opção* foram introduzidas por Buntine (1990) como uma generalização das árvores de decisão. Árvores de opção podem incluir *nós de opção*, que trocam o usual teste no valor de um atributo por um conjunto de testes, cada um dos quais sobre o valor de um atributo. Um nó de opção é como um nó *ou* em árvores *e-ou*. Na construção da árvore, em vez de selecionar o *melhor* atributo, são selecionados todos os atributos promissores, aqueles com maior valor do ganho de informação. Para cada atributo selecionado, uma árvore de decisão é construída. É de salientar que uma árvore de opção pode ter três tipos de nós: nós com somente um atributo teste – *nós de decisão*; nós com disjunções dos atributos de teste – *nós de opção*; e nós folhas.

A classificação de um exemplo **x** usando uma árvore de opção é um processo recursivo:

- Para um nó folha, devolva o rótulo da classe predito pela folha.
- Para um nó de decisão, o exemplo segue o único nó filho.

- Para um nó de opção, o exemplo segue todas as subárvores ligadas ao atributo teste. As previsões do teste disjuntivo são agregadas por um esquema de votação.

Kohavi e Kunz (1997) mostram que a redução de erro verificada nessas árvores é devida à redução na componente da variância (ver Seção 9.5). Afirmando ainda que é possível alcançar uma redução significativa das taxas de erro (em comparação com árvores regulares) usando *árvores de opção* restritas a dois níveis de nós de opção a partir do topo. Em comparação com o *bagging*, descrito no Capítulo 8, uma *árvore de opção* é determinística e não se baseia em técnicas de amostragem como as expostas no Capítulo 9. As árvores de opção usam todos os dados disponíveis para construir a árvore. A principal desvantagem das *árvores de opção* é que elas requerem muito mais memória e espaço em disco do que árvores normais.

6.4 Considerações Finais

Os modelos de decisão estudados neste capítulo, árvores e regras de decisão, são dos modelos mais utilizados em aprendizado de máquina. Hoje em dia existem algoritmos eficientes para indução de árvores de decisão ou conjuntos de regras e de aplicação eficiente, com um desempenho equivalente ao de outros modelos (como redes neurais e SVM), mas com maior grau de interpretabilidade. No entanto, as regras de decisão são de alguma forma mais expressivas e flexíveis para representar conhecimento.

Capítulo 7

Métodos Baseados em Otimização

Algumas técnicas de AM realizam a busca pela hipótese que descreve os dados recorrendo à otimização de alguma função. Nesse processo, um problema de aprendizado é formulado como um problema de otimização em que o objetivo consiste em minimizar (ou maximizar) uma função objetivo. No caso de problemas supervisionados, o rótulo dos objetos é normalmente considerado de alguma forma na formulação realizada.

Neste capítulo são descritas duas técnicas populares em AM que recorrem à otimização de uma função em seu treinamento: as redes neurais artificiais (RNAs) (Braga et al., 2007; Haykin, 1999) e as máquinas de vetores de suporte (SVMs, do inglês *support vector machines*) (Cristianini e Shawe-Taylor, 2000). As RNAs têm inspiração nas redes neurais biológicas presentes no cérebro, enquanto as SVMs têm suas origens na aplicação de conceitos da Teoria de Aprendizado Estatístico (Vapnik, 1995). O algoritmo de treinamento mais comum das RNAs envolve uma regra de correção de erros, na qual se recorre à otimização de uma função quadrática do erro entre as respostas da RNA e os rótulos dos exemplos. O treinamento das SVMs envolve a solução de um problema de otimização quadrática, formulado com o objetivo de maximizar a margem de separação entre os objetos de diferentes classes.

Os principais conceitos de RNAs são introduzidos na Seção 7.1, seguidos por uma introdução às SVMs na Seção 7.2. Em conformidade com a estrutura adotada neste livro, nessas seções essas técnicas são apresentadas no contexto de aprendizado supervisionado. Seu uso em problemas não supervisionados exige adaptações (Braga et al., 2007; Ben-Hur et al., 2000) que não serão discutidas neste capítulo (no caso das RNAs, um modelo não supervisionado é apresentado no Capítulo 12). Considerações finais são apresentadas na Seção 7.3.

7.1 Redes Neurais Artificiais

Na busca pela construção de máquinas inteligentes, ou com comportamento intelectual, um modelo que ocorre naturalmente é o do cérebro humano. Em nosso cotidiano, realizamos diversas tarefas que requerem atenção a diferentes eventos ao mesmo tempo e o processamento de informações variadas, de forma a tomarmos ações convenientes. Tarefas consideradas simples, como pegar um objeto ou mesmo caminhar, envolvem a ação de diversos componentes, tais como memória, aprendizado e coordenação física. A complexidade de tais ações simples para a maioria das pessoas é evidenciada pela dificuldade encontrada em ensinar robôs a realizá-las. A realização aparentemente simples de tarefas como essas e muitas outras é possível graças à nossa complexa estrutura biológica,

e o grande responsável pelo citado processamento de diversas informações e geração de respostas a elas é o cérebro humano.

A partir dessas motivações, o desenvolvimento das redes neurais artificiais (RNAs) tomou como inspiração a estrutura e o funcionamento do sistema nervoso, com o objetivo de simular a capacidade de aprendizado do cérebro humano na aquisição de conhecimento.

A procura por modelos computacionais ou matemáticos do sistema nervoso teve início na mesma época em que foram desenvolvidos os primeiros computadores eletrônicos, na década de 1940. Devem-se a McCulloch e Pitts (1943) alguns dos primeiros estudos na área. Em 1943 eles propuseram um modelo matemático de neurônio artificial em que os neurônios executavam funções lógicas simples e cada um podia executar uma função diferente. Esses neurônios artificiais são conhecidos como unidades lógicas com limiar (LTU, do inglês *Logic Threshold Unit*). McCulloch e Pitts mostraram que a combinação de vários neurônios artificiais em sistemas neurais produz um elevado poder computacional, uma vez que qualquer função que pudesse ser representada por uma combinação de funções lógicas poderia ser modelada por uma rede formada por esses neurônios. Essas redes iniciais não possuíam capacidade de aprendizado.

Outros pesquisadores deram importantes contribuições nos anos seguintes, entre eles Hebb (1949), com estudos sobre aprendizado, Rosenblatt (1958), com sua teoria sobre perceptrons, entre outros. As pesquisas na área de RNAs, entretanto, foram interrompidas na década de 1970 por diversos fatores. O principal foi a publicação do livro de Minsky e Papert (1969), no qual os autores apontaram a limitação da rede perceptron a problemas linearmente separáveis. Na década de 1980 houve um ressurgimento de interesse na área. Fatores contribuintes para essa retomada foram o aparecimento de computadores mais rápidos, o interesse da construção de computadores paralelos e, principalmente, a proposta de novas arquiteturas de RNAs com maior capacidade de representação e de algoritmos de aprendizado mais sofisticados.

Os trabalhos iniciais em RNAs tinham por objetivo compreender o cérebro e utilizar o conhecimento obtido para desenvolver sistemas de aprendizado biologicamente plausíveis. Dessa forma, as RNAs são baseadas em modelos abstratos de como pensamos que o cérebro (e os neurônios) funciona. Como muitos conceitos em RNAs foram inspirados em estudos sobre o sistema nervoso, a próxima seção descreve brevemente os seus principais aspectos.

7.1.1 Sistema Nervoso

O sistema nervoso, do qual faz parte o cérebro, é um conjunto complexo de células que determinam o funcionamento e o comportamento dos seres vivos. Ele está presente em todos os seres vivos vertebrados e na maioria dos invertebrados. A unidade fundamental do sistema nervoso é a célula nervosa, o neurônio, que se distingue das outras células por apresentar excitabilidade, que lhe permite responder a estímulos externos e internos. Isso possibilita a transmissão de impulsos nervosos a outros neurônios e a células musculares e glandulares.

O principal bloco de construção do cérebro é o neurônio. Os principais componentes de um neurônio são: dendritos, corpo celular e axônio. Um esquema de um neurônio simplificado pode ser visualizado na Figura 7.1. Diferentes tipos de neurônios podem assumir diferentes estruturas. Por estar fora do escopo do livro, essas variações não serão discutidas.

Os dendritos são prolongamentos dos neurônios especializados na recepção de estímulos nervosos provenientes de outros neurônios ou do ambiente. Esses estímulos são então transmitidos para o corpo celular ou soma. O soma coleta as informações recebidas dos dendritos, as combina e processa. De acordo com a intensidade e frequência dos estímulos recebidos, o corpo celular gera um novo impulso, que é enviado para o axônio. O axônio é um prolongamento dos neurônios, responsável pela condução dos impulsos elétricos produzidos no corpo celular até outro local mais distante (usualmente até outros neurônios). Alguns axônios de um adulto humano podem chegar a mais de um metro de comprimento. O sinal no neurônio flui então da esquerda para a direita, ou seja, dos dendritos para o corpo celular e em seguida para o axônio. O contato entre a terminação de um axônio e o dendrito de outro neurônio é denominado sinapse. As sinapses são, portanto, as unidades que medeiam as interações entre os neurônios (Haykin, 1999), e podem ser excitatórias ou inibitórias.

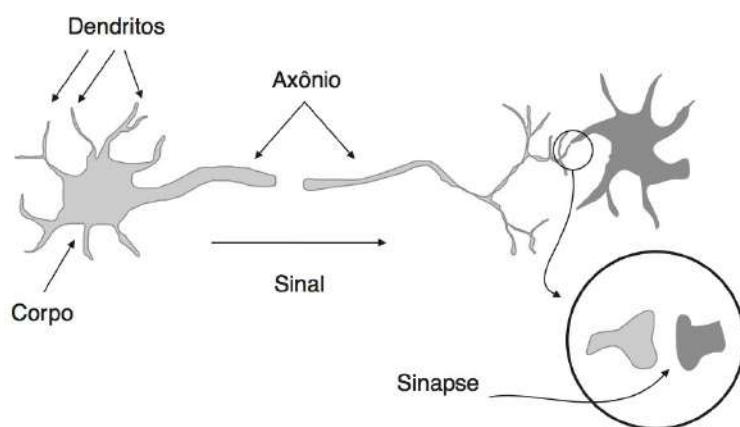


Figura 7.1 Neurônio biológico simplificado.

O cérebro humano possui um grande número de neurônios, da ordem de 10 a 500 bilhões. De acordo com estimativas, eles encontram-se organizados em aproximadamente 1000 módulos principais, cada um com 500 redes neurais. Além disso, cada neurônio pode estar conectado a centenas ou até mesmo milhares de outros neurônios. Essas redes biológicas trabalham de forma massivamente paralela, provendo uma grande rapidez de processamento. Essa característica é evidenciada pelo fato de que, apesar de os neurônios biológicos possuírem um tempo de execução normalmente da ordem de 10^{-3} segundos, o cérebro é capaz de realizar diversas tarefas (como reconhecimento de padrões, percepção e controle motor) várias vezes mais rapidamente que o mais rápido computador digital existente na atualidade.

7.1.2 Componentes Básicos das RNAs

As RNAs são sistemas computacionais distribuídos compostos de unidades de processamento simples, densamente interconectadas. Essas unidades, conhecidas como neurônios artificiais, computam funções matemáticas. As unidades são dispostas em uma ou mais camadas e interligadas por um grande número de conexões, geralmente unidirecionais. Na maioria das arquiteturas, essas conexões, que simulam as sinapses biológicas, possuem pesos associados, que ponderam a entrada recebida por cada neurônio da rede. Os pesos podem assumir valores positivos ou negativos, dependendo de o comportamento da conexão ser excitatório ou inibitório, respectivamente. Os pesos têm seus valores ajustados em

um processo de aprendizado e codificam o conhecimento adquirido pela rede (Braga et al., 2007).

Uma RNA é portanto caracterizada por dois aspectos básicos: arquitetura e aprendizado. Enquanto a arquitetura está relacionada ao tipo e número de unidades de processamento e à forma como os neurônios estão conectados, o aprendizado diz respeito às regras utilizadas para o ajuste dos pesos da rede e que informação é utilizada pelas regras.

Arquitetura

O neurônio é a unidade de processamento fundamental de uma RNA. Na Figura 7.2 é apresentado um modelo simples de neurônio artificial (Haykin, 1999). As unidades de processamento desempenham um papel muito simples. Cada terminal de entrada do neurônio, simulando os dendritos, recebe um valor. Os valores recebidos são ponderados e combinados por uma função matemática f_a , equivalendo ao processamento realizado pelo soma. A saída da função é a resposta do neurônio para a entrada. Várias funções diferentes podem ser utilizadas. Para apresentar as funções, vamos primeiro supor um objeto \mathbf{x} com d atributos representado na forma de vetor como $\mathbf{x} = [x_1, x_2, \dots, x_d]^t$ e um neurônio com d terminais de entrada cujos pesos são w_1, w_2, \dots, w_d , que podem ser representados na forma vetorial como $\mathbf{w} = [w_1, w_2, \dots, w_d]$. A entrada total recebida pelo neurônio, u , pode ser definida pela Equação 7.1:

$$u = \sum_{j=1}^d x_j w_j \quad (7.1)$$

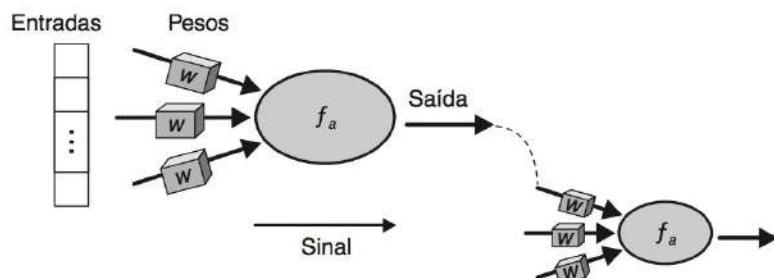


Figura 7.2 Neurônio artificial.

Conforme já mencionado, os neurônios podem apresentar conexões de entrada negativas ($w_j < 0$) ou positivas ($w_j > 0$). Um valor de peso igual a zero equivale a ausência da conexão associada.

A saída de um neurônio é definida por meio da aplicação de uma função de ativação à entrada total, conforme ilustrado na Figura 7.3. Várias funções de ativação têm sido propostas na literatura. A Figura 7.4 mostra a equação e o formato de três dessas funções, as funções linear, limiar e sigmoidal. O uso da função linear identidade (Figura 7.4(a)) implica retornar como saída o valor de u . Na função limiar (Figura 7.4(b)), empregada no modelo de neurônio artificial de McCulloch e Pitts (1943), o valor do limiar define quando o resultado da função limiar será igual a 1 ou 0 (alternativamente, pode-se empregar o valor -1). Quando a soma das entradas recebidas ultrapassa o limiar estabelecido, o neurônio torna-se ativo (saída $+1$). Quanto maior o valor do limiar, maior tem que ser o valor da entrada total para que o valor de saída do neurônio seja igual a 1. Na função

sigmoidal (Figura 7.4(c)), diferentes inclinações podem ser utilizadas. A função sigmoidal representa uma aproximação contínua e diferenciável da função limiar.

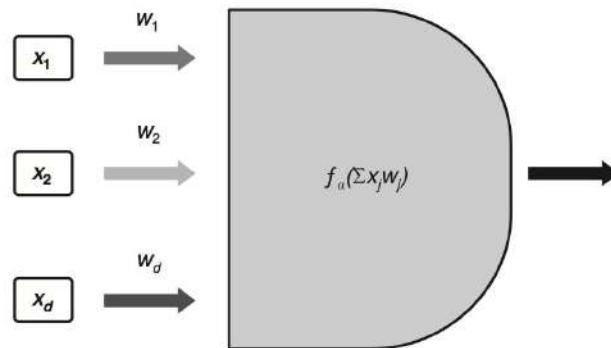


Figura 7.3 Entrada total em um neurônio artificial.

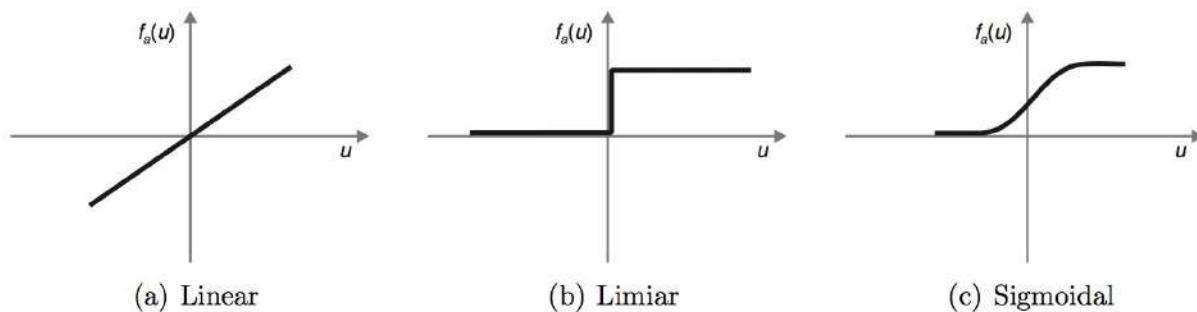


Figura 7.4 Exemplos de funções de ativação.

Em uma RNA, os neurônios podem estar dispostos em uma ou mais camadas. Quando duas ou mais camadas são utilizadas, um neurônio pode receber em seus terminais de entrada valores de saída de neurônios da camada anterior e/ou enviar seu valor de saída para terminais de entrada de neurônios da camada seguinte. A Figura 7.5 ilustra um exemplo de RNA com três camadas. Essa rede recebe como entrada valores de dois atributos de entrada e gera dois valores em sua saída.

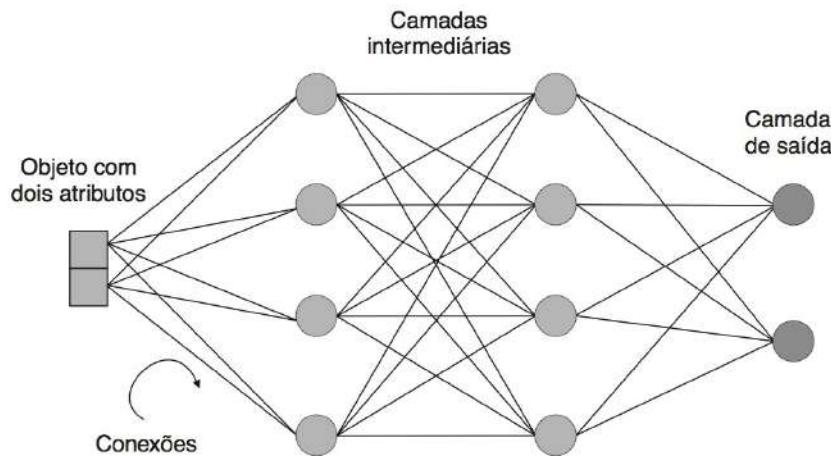


Figura 7.5 Exemplo de RNA multicamadas típica.

Uma rede com mais de uma camada de neurônios recebe o nome de rede multicamadas. A camada de neurônios que gera os valores de saída é chamada de camada de saída. As demais camadas são denominadas camadas intermediárias, escondidas ou ocultas.

Em uma rede multicamadas, as conexões entre os neurônios podem apresentar diferentes padrões de conexão. De acordo com esses padrões, a rede pode ser classificada em:

- Completamente conectada: quando os neurônios da rede estão conectados a todos os neurônios da camada anterior e/ou seguinte.
- Parcialmente conectada: quando os neurônios estão conectados a apenas alguns dos neurônios da camada anterior e/ou seguinte.
- Localmente conectada: são redes parcialmente conectadas, em que os neurônios conectados a um neurônio se encontram em uma região bem definida.

Esses três padrões de conectividade são mostrados na Figura 7.6.

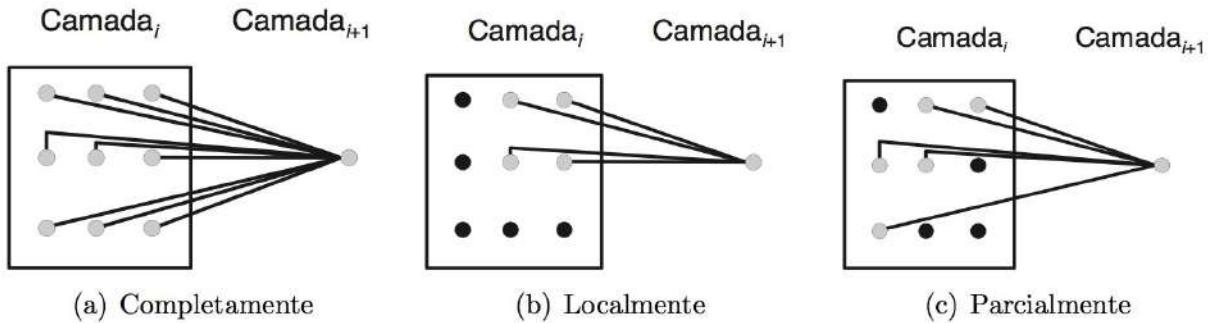


Figura 7.6 Diferentes padrões de conexão em uma RNA multicamadas.

Além do grau de conectividade, as RNAs podem apresentar ou não conexões de retroalimentação, ou *feedback*. A informação em uma rede neural geralmente flui da camada de entrada da rede para os neurônios da camada de saída. Para redes multicamadas, esse fluxo ocorre camada a camada. As conexões de retroalimentação permitem que um neurônio receba em seus terminais de entrada a saída de um neurônio da mesma camada ou de uma camada posterior. O neurônio pode inclusive receber sua própria saída em um de seus terminais de entrada. As redes com retropropagação, conhecidas como redes recorrentes, são indicadas para aplicações em que é necessário processar informações sequenciais e na simulação de sistemas dinâmicos. Exemplos de aplicações desses tipos incluem o processamento de língua natural e o controle de braços robóticos. Redes sem conexões de retropropagação, que são mais comumente utilizadas na prática, são denominadas RNAs *feedforward*. A Figura 7.7 ilustra exemplos dessas duas redes, uma recorrente e uma *feedforward*.

O número de camadas, o número de neurônios em cada camada, o grau de conectividade e a presença ou não de conexões de retropropagação definem a topologia de uma RNA.

Aprendizado

Vários algoritmos têm sido propostos na literatura para o ajuste dos parâmetros de uma RNA. Por ajuste de parâmetros entende-se principalmente a definição dos valores dos pesos associados às conexões da rede que fazem com que o modelo obtenha melhor desempenho, geralmente medido pela acurácia preditiva. Esses algoritmos, referenciados

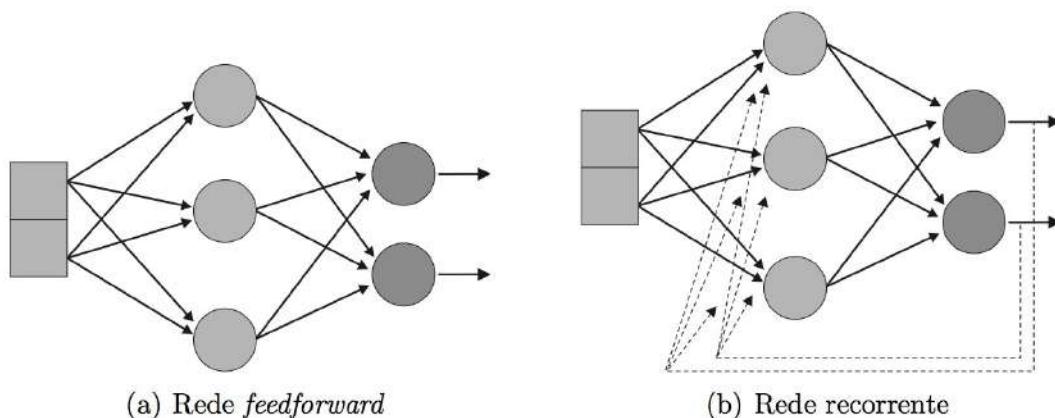


Figura 7.7 Redes neurais *feedforward* e *recorrente*.

como algoritmos de treinamento, são formados por um conjunto de regras bem definidas que especificam quando e como deve ser alterado o valor de cada peso. Diversos autores propuseram algoritmos de treinamento para RNAs seguindo os paradigmas de aprendizado supervisionado, não supervisionado e por reforço. Esses algoritmos podem ser divididos em quatro grupos:

1. Correção de erro: geralmente utilizados em aprendizado supervisionado, procuram ajustar os pesos da RNA de forma a reduzir os erros cometidos pela rede.
2. Hebbiano: frequentemente usados em aprendizado não supervisionado, são baseados na regra de Hebb, que diz que, se dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada.
3. Competitivo: utilizados em aprendizado não supervisionado, promovem uma competição entre neurônios para definir qual ou quais devem ter seus pesos ajustados. Os neurônios que vencem a competição em geral são os que respondem mais fortemente ao objeto apresentado aos seus terminais de entrada.
4. Termodinâmico (Boltzmann): algoritmos estocásticos baseados em princípios observados na metalurgia.

Nas últimas décadas, foi desenvolvido um grande número de arquiteturas de RNAs e algoritmos de treinamento para treiná-las. A seguir são apresentados algumas das principais arquiteturas existentes e os algoritmos de treinamento mais utilizados para elas. O foco da apresentação é em modelos e algoritmos do paradigma de aprendizado supervisionado.

7.1.3 Redes Perceptron e Adaline

A primeira RNA a ser implementada foi a rede perceptron, desenvolvida por Rosenblatt (1958). Essa rede, que utiliza modelo de McCulloch-Pitts como neurônio, introduziu o processo de treinamento de RNAs. Embora essa rede seja simples, apresentando apenas uma camada de neurônios, ela apresentou uma boa acurácia preditiva em diversos problemas de classificação. Conforme ilustrado na Figura 7.8, a rede perceptron possuía uma

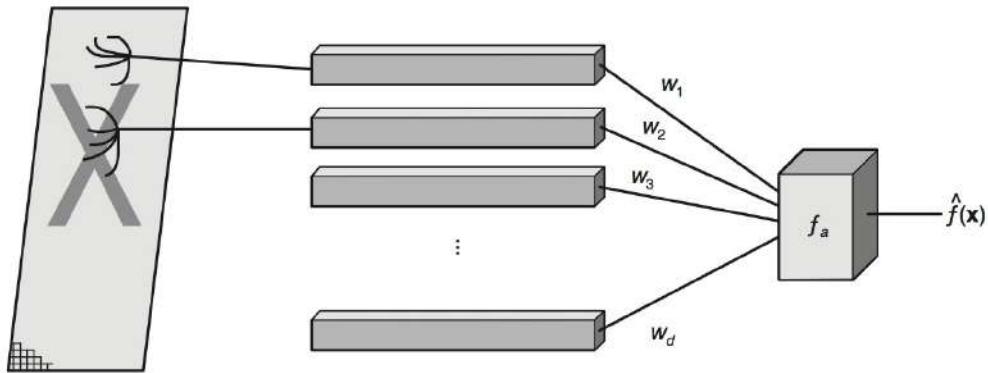


Figura 7.8 Rede perceptron.

máscara ou retina para receber os objetos de entrada, que eram pré-processados e então apresentados à rede, que possui apenas um neurônio.

A rede perceptron é treinada por um algoritmo supervisionado de correção de erro e usa a função de ativação do tipo limiar. Durante o seu treinamento, para um objeto \mathbf{x}_i , os pesos são ajustados de acordo com a Equação 7.2:

$$w_j(t+1) = w_j(t) + \eta x_i^j (y_i - \hat{f}(\mathbf{x}_i)) \quad (7.2)$$

em que $w_j(t)$ é o peso da j -ésima conexão de entrada no instante de tempo t , η é uma taxa de aprendizado, x_i^j é o valor do j -ésimo atributo do vetor de entrada \mathbf{x}_i , $\hat{f}(\mathbf{x}_i)$ é a saída produzida pela rede no instante de tempo t e y_i é a saída desejada para a rede (o rótulo de \mathbf{x}_i). O valor da taxa de aprendizado define a magnitude do ajuste feito no valor de cada peso. Valores altos fazem com que as variações sejam grandes, enquanto taxas pequenas implicam poucas variações nos pesos. Essa magnitude vai definir a velocidade de convergência da rede. O algoritmo de treinamento utilizado para a rede perceptron é descrito pelo Algoritmo 7.1.

Algoritmo 7.1 Algoritmo de treinamento da rede perceptron

Entrada: Um conjunto de n objetos de treinamento

Saída: Rede perceptron com valores dos pesos ajustados

1 Inicializar pesos da rede com valores baixos

2 repita

3 para cada objeto \mathbf{x}_i do conjunto de treinamento faça

4 Calcular valor da saída produzida pelo neurônio, $\hat{f}(\mathbf{x}_i)$

5 Calcular erro = $y_i - \hat{f}(\mathbf{x}_i)$

6 se erro > 0 então

7 Ajustar pesos do neurônio utilizando Equação 7.2

8 fim

9 fim

10 até erro = 0;

Alguns anos após propor a rede perceptron, Rosenblatt provou o teorema de convergência da rede perceptron, que diz que, se é possível classificar um conjunto de entradas linearmente, uma rede perceptron fará a classificação.

Uma outra rede neural com apenas uma camada que surgiu na mesma época da rede perceptron é a rede adaline (formada pelas primeiras sílabas de *Adaptive Linear*) (Widrow e Hoff, 1960). As principais diferenças entre as duas redes é que a rede adaline utiliza uma função de ativação linear e, assim, leva a magnitude do erro em consideração na hora de ajustar os pesos da rede. Para isso, a rede adaline utiliza uma regra de ajuste denominada Regra Delta, proposta por Widrow e Hoff (1960), cujo nome vem do uso da diferença entre os valores da saída desejada e da saída produzida. A equação utilizada para o ajuste dos pesos é similar à Equação 7.2 utilizada pela rede perceptron. A diferença está na forma como o valor da saída produzida, $\hat{f}(\mathbf{x}_i)$, é definido, sendo contínuo no caso das redes adaline. Com isso, essas redes são comumente utilizadas em problemas supervisionados de regressão. Em problemas de classificação, as saídas dos neurônios devem ser discretizadas. As redes perceptron, por outro lado, foram propostas para a solução de problemas de classificação.

Uma limitação das redes de uma camada, como as redes perceptron e adaline, é que elas conseguem classificar apenas objetos que são linearmente separáveis. Vamos supor que os objetos do conjunto de dados possuam apenas dois atributos de entrada. Os objetos de duas classes, com dois atributos cada, são linearmente separáveis se após plotar cada objeto em um espaço bidimensional, utilizando o valor de cada atributo para definir a posição do objeto, existe uma reta que separa os objetos de uma classe dos objetos da outra classe. Um exemplo de dados com dois atributos que são linearmente separáveis pode ser visto na Figura 7.9. Se em vez de dois os objetos apresentarem d atributos, o espaço de soluções será d -dimensional. Os objetos de duas classes serão linearmente separáveis se houver um hiperplano que separe os dados das duas classes.

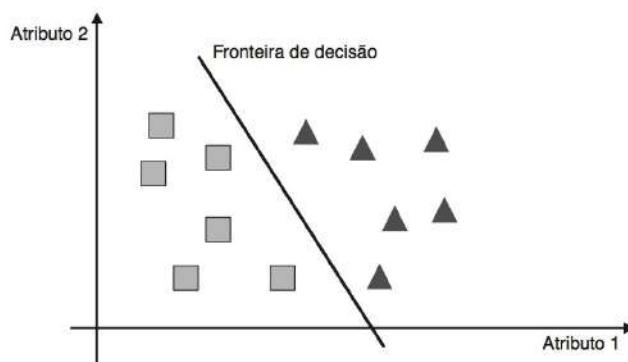


Figura 7.9 Objetos linearmente separáveis.

7.1.4 Perceptron Multicamadas

Para resolver problemas não linearmente separáveis utilizando RNAs, a alternativa mais utilizada é adicionar uma ou mais camadas intermediárias. Segundo Cybenko (1989), uma rede com uma camada intermediária pode implementar qualquer função contínua. A utilização de duas camadas intermediárias permite a aproximação de qualquer função.

As redes do tipo perceptron multicamadas (MLP, do inglês *multilayer perceptron*) apresentam uma ou mais camadas intermediárias de neurônios e uma camada de saída. A arquitetura mais comum para uma rede MLP é a completamente conectada, de forma que os neurônios de uma camada l estão conectados a todos os neurônios da camada $l+1$. Caso a camada l seja a primeira camada intermediária, cada um de seus neurônios estará

conectado a todos os atributos de entrada $x^j, j = 1 \dots, d$, para um objeto de entrada \mathbf{x} . A Figura 7.5 ilustra uma típica rede MLP.

Redes multicamadas utilizam nas camadas intermediárias funções de ativação não lineares, como a função sigmoidal. Pode ser facilmente mostrado utilizando conceitos de operações com matrizes, que uma rede multicamadas com funções de ativação lineares nos neurônios das camadas intermediárias é equivalente a uma rede de uma só camada.

Em uma MLP, cada neurônio realiza uma função específica. A função implementada por um neurônio de uma dada camada é uma combinação das funções realizadas pelos neurônios da camada anterior que estão conectados a ele. À medida que o processamento avança de uma camada intermediária para a camada seguinte, o processamento realizado (e a função correspondente) se torna mais complexo. Na primeira camada, cada neurônio aprende uma função que define um hiperplano, o qual divide o espaço de entrada em duas partes. Cada neurônio da camada seguinte combina um grupo de hiperplanos definidos pelos neurônios da camada anterior, formando regiões convexas. Os neurônios da camada seguinte combinam um subconjunto das regiões convexas em regiões de formato arbitrário. A Figura 7.10 exemplifica o papel de cada neurônio para a definição das fronteiras de decisão que permitirão à rede classificar novos exemplos. É a combinação das funções desempenhadas por cada neurônio da rede que define a função associada à RNA como um todo.

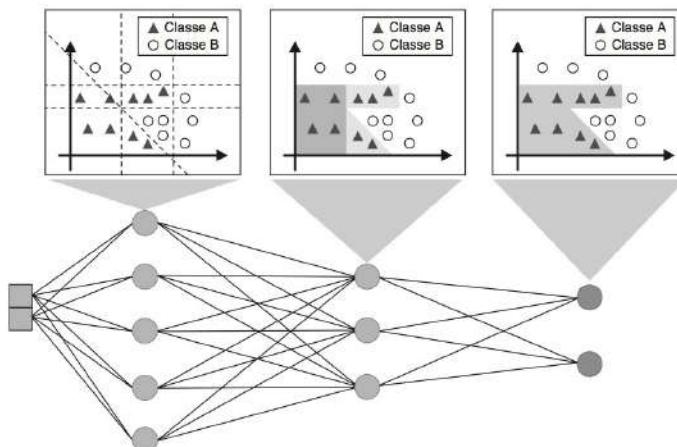


Figura 7.10 Papel desempenhado pelos neurônios das diferentes camadas da rede MLP.

Cada neurônio da camada de saída está associado a uma das classes presentes no conjunto de dados. Assim, os valores gerados pelos neurônios da camada de saída para um dado objeto de entrada podem ser representados por um vetor $\mathbf{y} = [y_1, y_2, \dots, y_k]^t$, em que k é o número de neurônios da camada de saída (e o número de classes do problema). Para o treinamento da rede, o vetor de respostas desejadas para cada objeto de entrada tem o valor 1 na posição associada à classe do objeto e 0 nas demais posições. O erro cometido pela rede para a classificação de um dado objeto é então definido pela comparação entre o vetor de saída dos neurônios da camada de saída e o vetor de valores desejados para essas saídas. A rede classifica corretamente um objeto quando o valor de saída mais elevado produzido pela rede é aquele gerado pelo neurônio de saída que corresponde à classe correta do objeto. Um erro de classificação ocorre quando o neurônio de uma outra classe é o que produz o valor de saída mais elevado. Quando nenhum neurônio produz um valor elevado ou o valor elevado é produzido por mais de um neurônio, a rede não tem condições de prever a classe do objeto. Deve-se observar que o uso das MLP também

pode ser diretamente estendido a problemas de regressão, mas nesse caso não se tem a discretização imposta pela escolha do neurônio com maior saída na predição.

7.1.5 Algoritmo *Back-propagation*

Um obstáculo que havia para utilizar redes multicamadas era a ausência de um algoritmo para o treinamento dessas redes, o que foi transposto com a proposta de um algoritmo de treinamento baseado em gradiente descendente denominado *back-propagation* (Rumelhart et al., 1986). Para que esse algoritmo seja utilizado, a função de ativação precisa ser contínua, diferenciável e, de preferência, não decrescente. A função de ativação do tipo sigmoidal obedece a esses requisitos.

Processo de Treinamento

O algoritmo *back-propagation* é baseado na regra delta utilizada na rede adaline, e também é conhecido como regra delta generalizada. Ele é constituído da iteração de duas fases, uma fase para a frente (*forward*) e uma fase para trás (*backward*). Na fase *forward*, cada objeto de entrada é apresentado à rede. O objeto é primeiramente recebido por cada um dos neurônios da primeira camada intermediária da rede, quando é ponderado pelo peso associado a suas conexões de entrada correspondentes. Cada neurônio nessa camada aplica a função de ativação a sua entrada total e produz um valor de saída, que é utilizado como valor de entrada pelos neurônios da camada seguinte. Esse processo continua até que os neurônios da camada de saída produzam cada um seu valor de saída, que é então comparado ao valor desejado para a saída desse neurônio. A diferença entre os valores de saída produzidos e desejados para cada neurônio da camada de saída indica o erro cometido pela rede para o objeto apresentado.

O valor do erro de cada neurônio da camada de saída é então utilizado na fase *backward* para ajustar seus pesos de entrada. O ajuste prossegue da camada de saída até a primeira camada intermediária. A Equação 7.3 ilustra como é feito o ajuste dos pesos de uma rede MLP pelo algoritmo *back-propagation*.

$$w_{jl}(t+1) = w_{jl}(t) + \eta x^j \delta_l \quad (7.3)$$

Nessa equação, w_{jl} representa o peso entre um neurônio l e o j -ésimo atributo de entrada ou a saída do j -ésimo neurônio da camada anterior, δ_l indica o erro associado ao l -ésimo neurônio e x^j indica a entrada recebida por esse neurônio (o j -ésimo atributo de entrada ou a saída do j -ésimo neurônio da camada anterior).

Como os valores dos erros são conhecidos apenas para os neurônios da camada de saída, o erro para os neurônios das camadas intermediárias precisa ser estimado. O algoritmo *back-propagation* propõe uma maneira de estimar o erro dos neurônios das camadas intermediárias utilizando os erros observados nos neurônios da camada posterior. O erro de um neurônio de uma dada camada intermediária é estimado como a soma dos erros dos neurônios da camada seguinte, cujos terminais de entrada estão conectados a ele, ponderados pelo valor do peso associado a essas conexões. Assim, a forma de calcular o erro depende da camada em que se encontra o neurônio, como mostra a Equação 7.4.

$$\delta_l = \begin{cases} f'_a e_l, & \text{se } n_l \in c_{sai} \\ f'_a \sum w_{lk} \delta_k, & \text{se } n_l \in c_{int} \end{cases} \quad (7.4)$$

Na Equação 7.4, n_l é o l -ésimo neurônio, c_{sai} representa a camada de saída, c_{int} representa uma camada intermediária, f'_a é a derivada parcial da função de ativação do neurônio e e_l é o erro quadrático cometido pelo neurônio de saída quando sua resposta é comparada à desejada, que é definido pela Equação 7.5.

$$e_l = \frac{1}{2} \sum_{q=1}^k (y_q - \hat{f}_q)^2 \quad (7.5)$$

A derivada parcial define o ajuste dos pesos, utilizando o gradiente descendente da função de ativação. Essa derivada mede a contribuição de cada peso no erro da rede para a classificação de um dado objeto \mathbf{x} . Se essa derivada para um dado peso for positiva, o peso está provocando um aumento da diferença entre a saída da rede e a saída desejada. Assim, sua magnitude deve ser reduzida para baixar o erro. Se a derivada for negativa, o peso está contribuindo para que a saída produzida pela rede seja mais próxima da desejada. Dessa forma, seu valor deve ser aumentado. O Algoritmo 7.2 ilustra os principais passos do algoritmo *back-propagation*. Nesse algoritmo, denomina-se por ciclo a apresentação de todos os objetos do conjunto de treinamento.

Algoritmo 7.2 Algoritmo de treinamento *back-propagation*

Entrada: Um conjunto de n objetos de treinamento
Saída: Rede MLP com valores dos pesos ajustados

- 1 Inicializar pesos da rede com valores aleatórios
- 2 Inicializar $erro_{total} = 0$
- 3 **repita**
- 4 **para cada** objeto \mathbf{x}_i do conjunto de treinamento **faça**
- 5 **para cada** camada da rede, a partir da primeira camada intermediária **faça**
- 6 **para cada** cada neurônio n_{jl} da camada atual **faça**
- 7 Calcular valor da saída produzida pelo neurônio, \hat{f}
- 8 **fim**
- 9 **fim**
- 10 Calcular $erro_{parcial} = y - \hat{f}$
- 11 **para cada** camada da rede, a partir da camada de saída **faça**
- 12 **para cada** cada neurônio n_{jl} da camada atual **faça**
- 13 Ajustar pesos do neurônio utilizando Equação 7.3
- 14 **fim**
- 15 **fim**
- 16 Calcular $erro_{total} = erro_{total} + erro_{parcial}$
- 17 **fim**
- 18 **até** $erro_{total} < \xi$;

Ajuste de Parâmetros

O valor da taxa de aprendizado η tem uma forte influência no tempo necessário à convergência da rede. Se a taxa de aprendizado for muito pequena, muitos ciclos podem

ser necessários para induzir um bom modelo. Por outro lado, a escolha de uma taxa elevada pode provocar oscilações que dificultam a convergência. Uma possível medida para amenizar esse problema é a introdução do termo *momentum* α , que quantifica o grau de importância da variação de peso do ciclo anterior ao ciclo atual. É isso que faz o algoritmo *back-propagation* com termo *momentum* (Rumelhart e McClelland, 1986). Nesse algoritmo, o ajuste dos pesos da rede utiliza a Equação 7.6, tornando o aprendizado mais estável e acelerando a convergência em regiões planas da função de erro.

$$w_{jl}(t+1) = w_{jl}(t) + \eta x^j \delta_l + \alpha(w_{jl}(t) - w_{jl}(t-1)) \quad (7.6)$$

Nessa equação, x^j indica o valor do j -ésimo atributo do objeto \mathbf{x} ou a saída do j -ésimo neurônio da camada anterior.

Variações

A versão padrão do algoritmo *back-propagation* ajusta os pesos de uma RNA para cada objeto ou exemplo apresentado individualmente. Existe uma variação denominada modo *batch*, em que os pesos de cada conexão são ajustados uma única vez para cada ciclo. Todos os exemplos do conjunto de treinamento são apresentados à rede, que tem seu erro médio calculado. O erro médio é utilizado para o ajuste dos pesos.

Outras variações muito utilizadas do algoritmo *back-propagation* são: Quickprop (Fahlman, 1988), Levenberg-Marquardt (Hagan e Menhaj, 1994), *momentum* de segunda ordem (Pearlmutter, 1992), Newton (Battiti, 1991) e Rprop (Riedmiller e Braun, 1994).

Critério de Parada

Os ciclos de apresentação dos dados de treinamento e eventuais ajustes de pesos no *back-propagation* são iterados até que seja atingido um critério de parada. Diferentes critérios de parada podem ser utilizados, como, por exemplo, um número máximo de ciclos ou uma taxa máxima de erro. Para reduzir a ocorrência de *overfitting*, parte do conjunto de treinamento é usualmente separada, formando um conjunto de validação. Os dados do conjunto de validação são apresentados à rede a cada n_v ciclos, para avaliar a taxa de erro da rede para dados que não fazem parte do conjunto de treinamento. Se as taxas de erro para os dados de treinamento e de validação forem plotadas em um gráfico, vai ser observado que no início do treinamento as duas taxas tendem a cair. Em um dado momento, a taxa de erro de validação pode começar a subir. Isso é um indício de que a rede parou de aprender e está se tornando superajustada aos dados de treinamento. Ou seja, está ocorrendo *overfitting*. Nesse ponto, o treinamento da rede deve ser finalizado. Esse processo de encerrar o treinamento da rede quando a taxa de erro para o conjunto de validação começa a subir é conhecido como validação cruzada com *early stop*.

Convergência do Algoritmo

A superfície de erro minimizada no *back-propagation*, apresenta regiões de mínimos locais e de mínimo global para problemas complexos. O objetivo do treinamento é atingir o mínimo global. Entretanto, é possível que o treinamento com *back-propagation* fique preso a uma região de mínimo local, fazendo com que a rede possua uma baixa acurácia

preditiva. Diferentemente das redes perceptron, não existe um teorema de convergência para algoritmos de aprendizado para o treinamento de redes multicamadas. A depender da distribuição estatística dos objetos, a rede pode convergir para um mínimo local ou demorar muito para encontrar uma solução adequada.

Uma crítica feita ao algoritmo *back-propagation* é sua lentidão na convergência para um bom conjunto de pesos e a sua queda de desempenho quando utilizado em grandes conjuntos de dados e problemas complexos. Mesmo para problemas simples, pode ser necessário apresentar o conjunto de treinamento centenas ou milhares de vezes, o que limita seu uso ao treinamento de pequenas redes, com no máximo poucos milhares de pesos. Embora alguns problemas práticos possam ser tratados com redes desse tamanho, há problemas reais que demandam redes maiores e mais complexas.

7.1.6 Projeto da Arquitetura de uma RNA

O número adequado de neurônios na camada intermediária de uma RNA depende de vários fatores, como:

- Número de exemplos de treinamento;
- Quantidade de ruído presente nos exemplos;
- Complexidade da função a ser aprendida;
- Distribuição estatística dos dados de treinamento.

O primeiro passo para que as RNAs possam induzir um modelo para o conjunto de dados é a definição de sua arquitetura, que engloba a escolha das funções de ativação e da topologia da rede. Conforme explicado anteriormente, a topologia diz respeito ao número de camadas e ao número de neurônios e, para redes multicamadas, qual o padrão de conexões e se existem conexões de retropropagação. A escolha da arquitetura mais promissora para um conjunto de dados não é uma tarefa simples. Ela é geralmente realizada por um processo de tentativa e erro, quando diferentes configurações são avaliadas antes de escolher uma delas. Nesse processo de busca, cada arquitetura investigada é treinada e avaliada de acordo com sua acurácia preditiva para o conjunto de dados de treinamento. Geralmente a arquitetura é definida por um processo de busca exaustiva, que pode ser realizada por diferentes abordagens. A seguir são apresentadas as abordagens mais utilizadas:

- Empírica: consiste na realização de uma busca cega no espaço de possíveis arquiteturas. Assim, diversas arquiteturas são testadas e comparadas até que se encontre uma RNA cuja acurácia preditiva seja adequada. Embora seja a abordagem mais utilizada, a busca cega normalmente apresenta um elevado custo de tempo e esforço. Algumas heurísticas são utilizadas na tentativa de acelerar o tempo de busca por uma RNA apropriada. Por exemplo, explorar apenas RNAs de uma camada intermediária, pois estas já possuem um considerável poder expressivo.
- Meta-heurística: essa abordagem gera um conjunto de variações de RNAs e combina as características das que apresentam melhores resultados, gerando assim um novo conjunto de RNAs. Essa técnica utiliza meta-heurísticas, geralmente Algoritmos

Genéticos (Holland, 1975), para realizar uma busca global por RNAs eficientes. Como um grande número de RNAs diferentes precisa ser treinado, essa abordagem possui um elevado custo computacional.

- Poda (ou *Pruning*): nessa abordagem, uma RNA com um grande número de neurônios é treinada até que seja alcançada a precisão desejada. Um algoritmo de poda é utilizado durante ou após o treinamento para remover conexões ou neurônios redundantes ou irrelevantes das camadas intermediárias da rede. Espera-se como resultado melhorar a capacidade de generalização da rede (Karnin, 1990). No fim do processo, uma rede mais compacta é em geral obtida.
- Construtiva: a abordagem construtiva gradualmente insere novos neurônios e conexões em uma RNA inicialmente sem neurônios intermediários, procurando melhorar seu desempenho diante do problema em questão.

Alternativamente, pode ser utilizada uma rede superdimensionada e evitar o *overfitting* (que ocorreria por causa da superespecialização que está associada a redes muito maiores que o necessário) utilizando validação cruzada ou controlando a norma dos pesos, para que os pesos não assumam valores positivos ou negativos muito elevados.

7.1.7 Discussão: Vantagens e Desvantagens

Mesmo com todo o progresso observado no desenvolvimento de RNAs, tanto em diversidade de arquiteturas e algoritmos de treinamento como em melhorias de precisão, a capacidade preditiva dessas redes, como das demais técnicas de AM, ainda está muito aquém da capacidade do cérebro humano. Essa diferença é ainda mais notável se for observado que o cérebro ocupa um volume de 1400 cm^3 e consome apenas 20 W de energia. Embora o tempo de processamento associado a um neurônio seja elevado, da ordem de milissegundos, cerca de 20 milhões de vezes mais lento que um processador de aproximadamente 2 GHz, essa lentidão é compensada pelo processamento paralelo de um grande número de neurônios densamente conectados a outros neurônios.

Contudo, o constante aumento no número de aplicações que incorporam RNAs confirma a sua importância para a resolução de problemas práticos. As RNAs possuem várias características que levam à sua popularidade, como, por exemplo, generalização e tolerância a falhas e ruídos (Braga et al., 2007; Haykin, 1999). Essas características fazem com que as RNAs apresentem um bom desempenho (baixa taxa de erros) quando utilizadas em um grande número de aplicações, destacando-se principalmente tarefas de percepção e controle, como visão computacional e robótica.

Contudo, para várias dessas aplicações, não apenas o desempenho obtido é importante, mas também a facilidade de o usuário compreender como a rede chega a suas decisões. Uma das críticas mais frequentes ao uso de RNAs é a dificuldade de entender como e por que as RNAs tomam suas decisões. A principal dificuldade de entender os conceitos representados pelas RNAs está no fato de o conhecimento estar armazenado na forma de uma grande quantidade de parâmetros e esses parâmetros serem manipulados através de complicadas fórmulas matemáticas. Devido a essa limitação, RNAs são comumente referenciadas como “caixas-pretas”.

Em contraste com as RNAs, o conhecimento representado por algoritmos simbólicos de Inteligência Artificial, como as regras e árvores de decisão, é geralmente mais amigável e de

mais fácil compreensão. Procurando unir o melhor dos dois mundos, vários pesquisadores estão se dedicando ao estudo de técnicas para extração de conhecimento das RNAs que traduzam o conhecimento adquirido pela rede para um formato tão amigável e compreensível quanto aquele gerado por técnicas simbólicas. Entre os vários algoritmos que têm sido propostos para a extração de conhecimento de RNAs, podem ser citados: EN (*Explanation Facility*) (Pau e Gotzche, 1992), KT (*Knowledgetron*) (Fu, 1994), LAP (*Language Processing*) (Hayward et al., 1995), *M-of-N* (Towell e Shavlik, 1993), OLS (Tickle et al., 1995), RULEX (*RULEs Extraction*) (Andrews et al., 1996), RuleNeg (Andrews et al., 1995), e TREPAN (*TREEs PArroting Networks*) (Craven e Shavlik, 1994). O conhecimento extraído de RNAs geralmente se dá através da geração de um conjunto de regras ou uma árvore de decisão. É importante ressaltar que a extração de conhecimento de uma RNA é uma tarefa que exige recursos e esforços adicionais. Por isso, se não for bem justificada, pode apresentar efeitos negativos.

Outra dificuldade das RNAs é a escolha do melhor conjunto de parâmetros para a arquitetura da rede, que faz com que o projeto de redes seja algumas vezes definido como “magia negra” (“*black art*”).

7.2 Máquinas de Vetores de Suporte

As máquinas de vetores de suporte (*support vector machines* – SVMs) (Cristianini e Shawe-Taylor, 2000) vêm recebendo crescente atenção da comunidade de AM nos últimos anos. Os resultados da aplicação dessa técnica são comparáveis e muitas vezes superiores aos obtidos por outros algoritmos populares de aprendizado, tal como as RNAs. Exemplos de aplicações de sucesso podem ser encontrados em diversos domínios, com destaque na categorização de textos (Joachims, 2002) e em Bioinformática (Schölkopf et al., 2003; Noble, 2004).

As SVMs são embasadas pela teoria de aprendizado estatístico, desenvolvida por Vapnik (1995) a partir de estudos iniciados em Vapnik e Chervonenkis (1971). Essa teoria estabelece uma série de princípios que devem ser seguidos na obtenção de classificadores com boa capacidade de generalização. A seguir é apresentada uma breve introdução aos principais conceitos dessa teoria, focando em aspectos a partir dos quais as SVMs são formuladas. As discussões apresentadas referem-se ao uso de SVMs na solução de problemas de classificação. Uma formulação de SVMs para problemas de regressão é discutida na Seção 7.2.4.

7.2.1 Teoria de Aprendizado Estatístico

Sejam h um classificador e H o conjunto de todos os classificadores que um determinado algoritmo de AM pode gerar. Esse algoritmo, durante o processo de aprendizado, utiliza um conjunto de treinamento \mathbf{X} , composto de n pares (\mathbf{x}_i, y_i) , para gerar um classificador particular $\hat{h} \in H$.

A Teoria de Aprendizado Estatístico (TAE) estabelece condições matemáticas que auxiliam na escolha de um classificador particular \hat{h} a partir de um conjunto de dados de treinamento. Essas condições levam em conta o desempenho do classificador no conjunto de treinamento e a sua complexidade, com o objetivo de obter um bom desempenho também para novos dados do mesmo domínio.

Considerações sobre a Escolha do Classificador

Na aplicação da TAE, assume-se inicialmente que os dados do domínio em que o aprendizado está ocorrendo são gerados de forma independente e identicamente distribuída (i.i.d.) de acordo com uma distribuição de probabilidade $P(\mathbf{x}, y)$ que descreve a relação entre os objetos e os seus rótulos (Burges, 1998). O erro (também denominado risco) esperado de um classificador h para todos os dados desse domínio pode então ser quantificado pela Equação 7.7 (Müller et al., 2001), medindo a capacidade de generalização de h . Na Equação 7.7, $c(h(\mathbf{x}), y)$ é uma função de custo relacionando a previsão $h(\mathbf{x})$ à saída desejada y , em que $y_i \in \{-1, +1\}$. Um tipo de função de custo comumente empregada em problemas de classificação é a 0–1, definida pela Equação 7.8. Essa função retorna o valor 0 se \mathbf{x} é classificado corretamente e 1 em caso contrário.

$$R(h) = \int c(h(\mathbf{x}), y) dP(\mathbf{x}, y) \quad (7.7)$$

$$c(h(\mathbf{x}), y) = \frac{1}{2} |y - h(\mathbf{x})| \quad (7.8)$$

Infelizmente, não é possível minimizar o risco esperado apresentado na Equação 7.7 diretamente, uma vez que em geral a distribuição de probabilidade $P(\mathbf{x}, y)$ é desconhecida. Tem-se unicamente a informação dos dados de treinamento, também amostrados de $P(\mathbf{x}, y)$. Normalmente utiliza-se o princípio da indução para inferir uma função \hat{h} que minimize o erro sobre os dados de treinamento e espera-se que esse procedimento leve também a um menor erro sobre novos dados. Essa é a estratégia adotada pela maioria das técnicas de AM supervisionadas. O risco empírico de h , fornecido pela Equação 7.9, mede o desempenho do classificador nos dados de treinamento, por meio da taxa de classificações incorretas obtidas em \mathbf{X} .

$$R_{emp}(h) = \frac{1}{n} \sum_{i=1}^n c(h(\mathbf{x}_i), y_i) \quad (7.9)$$

Esse processo de indução com base nos dados de treinamento conhecidos constitui o princípio de minimização do risco empírico (Smola e Schölkopf, 2002). Assintoticamente, com $n \rightarrow \infty$, é possível estabelecer condições para o algoritmo de aprendizado que garantam a obtenção de classificadores cujos valores de risco empírico convergem para o risco esperado (Müller et al., 2001). Para conjuntos de dados menores, porém, geralmente não é possível determinar esse tipo de garantia. Embora a minimização do risco empírico possa levar a um menor risco esperado, nem sempre isso ocorre. Considere, por exemplo, um classificador binário (para duas classes) que memoriza todos os objetos de treinamento e gera classificações aleatórias para outros exemplos (Smola et al., 1999). Embora seu risco empírico seja nulo, seu risco esperado é 0,5.

A noção expressa nesses argumentos é a de que, ao permitir que \hat{h} seja escolhida a partir de um conjunto de funções amplo H , é sempre possível encontrar uma h com pequeno risco empírico. Porém, nesse caso os exemplos de treinamento podem se tornar pouco informativos para a tarefa de aprendizado, pois o classificador induzido pode se superajustar a eles. Deve-se então restringir a classe de funções da qual \hat{h} é extraída. Existem diversas abordagens para tal. A TAE lida com essa questão considerando a complexidade (também referenciada por capacidade) da classe de funções que o algoritmo

de aprendizado é capaz de induzir (Smola e Schölkopf, 2002). Nessa direção, a TAE provê diversos limites no risco esperado de uma função de classificação, os quais podem ser empregados na escolha do classificador. A seguir são relacionados alguns dos principais limites sobre os quais as SVMs se baseiam.

Limites no Risco Esperado

Um limite importante fornecido pela TAE relaciona o risco esperado de uma função ao seu risco empírico e a um termo de capacidade. Esse limite, apresentado na Inequação 7.10, é garantido com probabilidade $1 - \theta$, em que $\theta \in [0, 1]$ (Burges, 1998).

$$R(h) \leq R_{emp}(h) + \sqrt{\frac{VC \left(\ln \left(2n/VC \right) + 1 \right) - \ln \left(\theta/4 \right)}{n}} \quad (7.10)$$

Nessa inequação, VC denota a dimensão Vapnik-Chervonenkis (Vapnik, 1995) da classe de funções H à qual h pertence, n representa a quantidade de exemplos no conjunto de treinamento \mathbf{X} e a parcela de raiz na soma é referenciada como termo de capacidade. A dimensão VC mede a capacidade do conjunto de funções H (Burges, 1998). Quanto maior o seu valor, mais complexas são as funções de classificação que podem ser induzidas a partir de H .

A contribuição principal da Inequação 7.10 está em afirmar a importância de se controlar a capacidade do conjunto de funções H do qual o classificador é extraído. Interpretando-a em termos práticos, tem-se que o risco esperado pode ser minimizado pela escolha adequada, por parte do algoritmo de aprendizado, de um classificador \hat{h} que minimize o risco empírico e que pertença a uma classe de funções H com baixa dimensão VC. Com esses objetivos, definiu-se um princípio de indução denominado minimização do risco estrutural (Vapnik, 1998), que busca a função de menor complexidade possível que tenha um baixo erro para os dados de treinamento.

Embora o limite representado na Inequação 7.10 tenha sido útil na definição do procedimento de minimização do risco estrutural, na prática surgem alguns problemas. Em primeiro lugar, computar a dimensão VC de uma classe de funções geralmente não é uma tarefa trivial. Soma-se a isso o fato de que o valor dela pode ser desconhecido ou infinito (Müller et al., 2001).

Para funções de decisão lineares do tipo $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ (em que \mathbf{w} é o vetor normal a h), entretanto, existem resultados alternativos que relacionam o risco esperado ao conceito de margem (Smola et al., 1999). A margem de um exemplo tem relação com sua distância à fronteira de decisão induzida, e é uma medida da confiança da previsão do classificador. Para um problema binário, em que $y_i \in \{-1, +1\}$, dados uma função h e um exemplo \mathbf{x}_i , a margem $\varrho(h(\mathbf{x}_i), y_i)$ com que esse objeto é classificado por h pode ser calculada pela Equação 7.11. Nessa equação, um valor negativo de $\varrho(\mathbf{x}_i, y_i)$ denota uma classificação incorreta.

$$\varrho(h(\mathbf{x}_i), y_i) = y_i h(\mathbf{x}_i) \quad (7.11)$$

Para obter a margem geométrica de um objeto \mathbf{x}_i , a qual mede efetivamente a distância de \mathbf{x}_i à fronteira de decisão, divide-se o termo à direita da Equação 7.11 pela norma de \mathbf{w} , ou seja, por $\|\mathbf{w}\|$ (Smola e Schölkopf, 2002). Para exemplos classificados incorretamente,

o valor obtido equivale à distância com sinal negativo. Para realizar uma diferenciação, a margem da Equação 7.11 será referenciada como margem de confiança.

A partir do conceito introduzido, é possível definir o risco ou erro marginal de uma função h ($R_\rho(h)$) sobre um conjunto de treinamento. Esse erro fornece a proporção de exemplos de treinamento cuja margem de confiança é inferior a uma determinada constante $\rho > 0$ (Equação 7.12) (Smola et al., 1999).

$$R_\rho(h) = \frac{1}{n} \sum_{i=1}^n I(y_i h(\mathbf{x}_i) < \rho) \quad (7.12)$$

Na Equação 7.12, $I(q) = 1$ se q é verdadeiro e $I(q) = 0$ se q é falso.

Existe uma constante c tal que, com probabilidade $1 - \theta \in [0, 1]$, para todo $\rho > 0$ e H correspondendo à classe de funções lineares $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ com $\|\mathbf{x}\| \leq R$ e $\|\mathbf{w}\| \leq 1$, o seguinte limite se aplica (Smola et al., 1999):

$$R(h) \leq R_\rho(h) + \sqrt{\frac{c}{n} \left(\frac{R^2}{\rho^2} \log^2 \left(\frac{n}{\rho} \right) + \log \left(\frac{1}{\theta} \right) \right)} \quad (7.13)$$

Como na Inequação 7.10, tem-se na Expressão 7.13 novamente o erro esperado limitado pela soma de uma medida de erro no conjunto de treinamento, nesse caso o erro marginal, a um termo de capacidade. A interpretação do presente limite é de que uma maior margem ρ implica um menor termo de capacidade. Entretanto, a maximização da margem pode levar a um aumento na taxa de erro marginal, pois torna-se mais difícil obedecer à restrição de todos os dados de treinamento estarem distantes de uma margem maior em relação ao hiperplano separador. Um baixo valor de ρ , em contrapartida, leva a um erro marginal menor, porém aumenta o termo de capacidade. Deve-se então buscar um compromisso entre a maximização da margem e a obtenção de um erro marginal baixo.

Como conclusão tem-se que, na geração de um classificador linear, deve-se buscar um hiperplano que tenha margem ρ elevada e cometa poucos erros marginais, minimizando assim o erro sobre os dados de treinamento e também sobre novos dados. Esse hiperplano é usualmente denominado ótimo.

Existem diversos outros limites reportados na literatura, assim como outros tipos de medida de complexidade de uma classe de funções. Um exemplo é a dimensão *fat-shattering*, que caracteriza o poder de um conjunto de funções em separar os objetos com uma margem ρ (Shawe-Taylor et al., 1998). Contudo, os limites apresentados anteriormente proveem uma base teórica suficiente à compreensão das SVMs.

7.2.2 SVMs Lineares

As SVMs surgiram pelo emprego direto dos resultados fornecidos pela TAE. Esta seção apresenta o uso de SVMs na obtenção de fronteiras lineares para a separação de objetos pertencentes a duas classes. A primeira formulação, mais simples, lida com problemas linearmente separáveis (Boser et al., 1992). Essa formulação foi posteriormente estendida para definir fronteiras lineares sobre conjuntos de dados mais gerais (Cortes e Vapnik, 1995). A partir desses conceitos iniciais, na Seção 7.2.3 descreve-se a obtenção de fronteiras não lineares com SVMs.

SVMs com Margens Rígidas

As SVMs lineares com margens rígidas definem fronteiras lineares a partir de dados linearmente separáveis. Seja \mathbf{X} um conjunto de treinamento com n objetos $\mathbf{x}_i \in X$ e seus respectivos rótulos $y_i \in Y$, em que X constitui o espaço de entrada e $Y = \{-1, +1\}$ são as possíveis classes. \mathbf{X} é linearmente separável se é possível separar os objetos das classes $+1$ e -1 por um hiperplano.

Classificadores que separam os dados por meio de um hiperplano são denominados lineares. A equação de um hiperplano é apresentada na Equação 7.14, em que $\mathbf{w} \cdot \mathbf{x}$ é o produto escalar entre os vetores \mathbf{w} e \mathbf{x} , $\mathbf{w} \in X$ é o vetor normal ao hiperplano descrito e $\frac{b}{\|\mathbf{w}\|}$ corresponde à distância do hiperplano em relação à origem, com $b \in \mathbb{R}$.

$$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \quad (7.14)$$

Essa equação pode ser usada para dividir o espaço de entrada X em duas regiões: $\mathbf{w} \cdot \mathbf{x} + b > 0$ e $\mathbf{w} \cdot \mathbf{x} + b < 0$. Uma função sinal $g(\mathbf{x}) = \text{sgn}(h(\mathbf{x}))$ pode então ser empregada na obtenção das classificações, conforme ilustrado na Equação 7.15.

$$g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \begin{cases} +1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases} \quad (7.15)$$

A partir de $h(\mathbf{x})$, é possível obter um número infinito de hiperplanos equivalentes, pela multiplicação de \mathbf{w} e b por uma mesma constante. Define-se o hiperplano canônico em relação ao conjunto \mathbf{X} como aquele em que \mathbf{w} e b são escalados de forma que os exemplos mais próximos ao hiperplano $\mathbf{w} \cdot \mathbf{x} + b = 0$ satisfazam a Equação 7.16 (Müller et al., 2001).

$$|\mathbf{w} \cdot \mathbf{x}_i + b| = 1 \quad (7.16)$$

Essa forma implica as Inequações 7.17, resumidas na Equação 7.18 e ilustradas na Figura 7.11.

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq +1 & \text{se } y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 & \text{se } y_i = -1 \end{cases} \quad (7.17)$$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad \forall (\mathbf{x}_i, y_i) \in \mathbf{X} \quad (7.18)$$

Manipulações algébricas simples usando pontos sobre os hiperplanos canônicos H_1 e H_2 (Figura 7.11) permitem deduzir que a maximização da margem de separação dos objetos em relação a $\mathbf{w} \cdot \mathbf{x} + b = 0$ pode ser obtida pela minimização de $\|\mathbf{w}\|$ (Campbell, 2000). Dessa forma, recorre-se ao seguinte problema de otimização:

$$\underset{\mathbf{w}, b}{\text{Minimizar}} \quad \frac{1}{2} \|\mathbf{w}\|^2 \quad (7.19)$$

$$\text{Com as restrições: } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad \forall i = 1, \dots, n \quad (7.20)$$

As restrições são impostas de maneira a assegurar que não haja dados de treinamento entre as margens de separação das classes. Por esse motivo, a SVM obtida possui também a nomenclatura de SVM com margens rígidas.

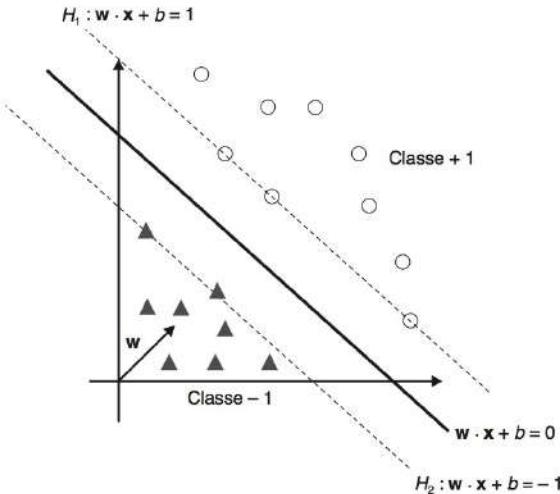


Figura 7.11 Ilustração de hiperplanos canônicos e separador.

O problema de otimização obtido é quadrático, e sua solução possui uma ampla e estabelecida teoria matemática. Como a função objetivo sendo minimizada é convexa e os pontos que satisfazem as restrições formam um conjunto convexo, esse problema possui um único mínimo global. Problemas desse tipo podem ser solucionados com a introdução de uma função lagrangiana, que engloba as restrições à função objetivo, associadas a parâmetros denominados multiplicadores de Lagrange α_i (Equação 7.21).

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1) \quad (7.21)$$

A função lagrangiana deve ser minimizada, o que implica maximizar as variáveis α_i , enquanto \mathbf{w} e b devem ser minimizados. Derivando L em relação a b e \mathbf{w} e igualando o resultado a 0, obtém-se as Equações 7.22 e 7.23.

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (7.22)$$

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (7.23)$$

Substituindo as Equações 7.22 e 7.23 na Equação 7.21, obtém-se o seguinte problema de otimização:

$$\underset{\boldsymbol{\alpha}}{\text{Maximizar}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (7.24)$$

$$\text{Com as restrições: } \begin{cases} \alpha_i \geq 0, \quad \forall i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad (7.25)$$

Essa formulação é denominada forma dual, enquanto o problema original é referenciado como forma primal. A forma dual possui os atrativos de apresentar restrições mais simples e permitir a representação do problema de otimização em termos de produtos internos

entre objetos, o que será útil na posterior não linearização das SVMs (Seção 7.2.3). É interessante observar também que o problema dual é formulado utilizando apenas os dados de treinamento e os seus rótulos.

Sejam α^* a solução do problema dual e \mathbf{w}^* e b^* as soluções da forma primal. Obtido o valor de α^* , \mathbf{w}^* pode ser determinado pela Equação 7.23. O parâmetro b^* é definido por α^* e por condições de Kuhn-Tucker, provenientes da teoria de otimização com restrições e que devem ser satisfeitas no ponto ótimo. Elas afirmam que no ponto ótimo o produto entre as variáveis duais (de Lagrange) e as restrições deve ser nulo (Smola e Schölkopf, 1998). Para o problema dual formulado, tem-se:

$$\alpha_i^* (y_i (\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1) = 0, \quad \forall i = 1, \dots, n \quad (7.26)$$

Observa-se nessa equação que α_i^* pode ser diferente de 0 somente para os objetos que se encontram sobre os hiperplanos H_1 e H_2 . Esses são os exemplos que se situam mais próximos ao hiperplano separador, exatamente sobre as margens. Para os outros casos, a condição apresentada na Equação 7.26 é obedecida apenas com $\alpha_i^* = 0$. Esses pontos não participam então do cálculo de \mathbf{w}^* (Equação 7.23). Os exemplos que possuem $\alpha_i^* > 0$ são denominados vetores de suporte (*support vectors* - SVs) e podem ser considerados os objetos mais informativos do conjunto de treinamento (Burges, 1998). O valor de b^* é calculado a partir dos SVs e das condições representadas na Equação 7.26. Computa-se a média apresentada na Equação 7.27 sobre todos \mathbf{x}_j tal que $\alpha_j^* > 0$, ou seja, todos os SVs. Nessa equação, n_{SV} denota o número de SVs e SV representa o conjunto dos SVs.

$$b^* = \frac{1}{n_{SV}} \sum_{\mathbf{x}_j \in SV} \frac{1}{y_j} - \mathbf{w}^* \cdot \mathbf{x}_j = \frac{1}{n_{SV}} \sum_{\mathbf{x}_j \in SV} \left(\frac{1}{y_j} - \sum_{\mathbf{x}_i \in SV} \alpha_i^* y_i \mathbf{x}_i \cdot \mathbf{x}_j \right) \quad (7.27)$$

Como resultado final, tem-se o classificador $g(\mathbf{x})$ apresentado na Equação 7.28.

$$g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \text{sgn} \left(\sum_{\mathbf{x}_i \in SV} y_i \alpha_i^* \mathbf{x}_i \cdot \mathbf{x} + b^* \right) \quad (7.28)$$

SVMs com Margens Suaves

Em situações reais, é difícil encontrar aplicações cujos dados sejam linearmente separáveis. Isso se deve a diversos fatores, entre eles a presença de ruídos e *outliers* nos objetos ou à própria natureza do problema, que pode ser não linear. As SVMs lineares apresentadas anteriormente podem ser estendidas para lidar com conjuntos de treinamento mais gerais. Para realizar essa tarefa, permite-se que alguns objetos possam violar a restrição da Equação 7.20. Isso é feito com a introdução de variáveis de folga ξ_i , para todo $i = 1, \dots, n$. Essas variáveis relaxam as restrições impostas ao problema de otimização primal, que se tornam (Smola e Schölkopf, 2002):

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \dots, n \quad (7.29)$$

A aplicação desse procedimento suaviza as margens do classificador linear, permitindo que alguns objetos permaneçam entre os hiperplanos H_1 e H_2 e também a ocorrência de alguns erros de classificação. Por esse motivo, as SVMs obtidas nesse caso também podem ser referenciadas como SVMs com margens suaves.

Um erro no conjunto de treinamento é indicado por um valor de ξ_i maior que 1. Logo, a soma dos ξ_i representa um limite no número de erros de treinamento. Para levar em consideração esse termo, minimizando assim o erro sobre os dados de treinamento, a função objetivo da Equação 7.19 é reformulada como (Burges, 1998):

$$\underset{\mathbf{w}, b, \boldsymbol{\xi}}{\text{Minimizar}} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \left(\sum_{i=1}^n \xi_i \right) \quad (7.30)$$

A constante C é um termo de regularização que impõe um peso à minimização dos erros no conjunto de treinamento em relação à minimização da complexidade do modelo. A presença do termo $\sum_{i=1}^n \xi_i$ no problema de otimização também pode ser vista como uma minimização de erros marginais, pois um valor de $\xi_i \in (0, 1]$ indica um objeto entre as margens. Tem-se então uma formulação de acordo com os princípios da TAE discutidos na Seção 7.2.1.

Novamente o problema de otimização gerado é quadrático, com as restrições lineares apresentadas na Equação 7.29. Sua solução envolve passos matemáticos semelhantes aos apresentados anteriormente, com a introdução de uma função lagrangiana e tornando suas derivadas parciais nulas. Tem-se como resultado o seguinte problema dual:

$$\underset{\boldsymbol{\alpha}}{\text{Maximizar}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (7.31)$$

$$\text{Com as restrições: } \begin{cases} 0 \leq \alpha_i \leq C, \quad \forall i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad (7.32)$$

Pode-se observar que essa formulação é igual à apresentada para as SVMs de margens rígidas, a não ser pela restrição nos α_i , que agora são limitados pelo valor de C .

Seja $\boldsymbol{\alpha}^*$ a solução do problema dual, enquanto \mathbf{w}^* , b^* e $\boldsymbol{\xi}^*$ denotam as soluções da forma primal. O vetor \mathbf{w}^* continua sendo determinado pela Equação 7.23. As variáveis ξ_i^* podem ser calculadas pela Equação 7.33 (Cristianini e Shawe-Taylor, 2000).

$$\xi_i^* = \max \left\{ 0, 1 - y_i \sum_{j=1}^n y_j \alpha_j^* \mathbf{x}_j \cdot \mathbf{x}_i + b^* \right\} \quad (7.33)$$

A variável b^* provém novamente de $\boldsymbol{\alpha}^*$ e de condições de Kuhn-Tucker, que nesse caso são:

$$\alpha_i^* (y_i (\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1 + \xi_i^*) = 0 \quad (7.34)$$

$$(C - \alpha_i^*) \xi_i^* = 0 \quad (7.35)$$

Como nas SVMs de margens rígidas, os pontos \mathbf{x}_i para os quais $\alpha_i^* > 0$ são denominados vetores de suporte (SVs), e são os objetos que participam da formação do hiperplano separador. Porém, nesse caso, pode-se distinguir tipos distintos de SVs. Se $\alpha_i^* < C$, pela Equação 7.35, $\xi_i^* = 0$ e então, da Equação 7.34, esses SVs encontram-se sobre as margens e também são denominados livres. Os SVs para os quais $\alpha_i^* = C$ podem representar três

casos: erros, se $\xi_i^* > 1$; pontos corretamente classificados, porém entre as margens, se $0 < \xi_i^* \leq 1$; ou pontos sobre as margens, se $\xi_i^* = 0$. O último caso ocorre raramente, e os SVs anteriores são denominados limitados. Na Figura 7.12 são ilustrados os possíveis tipos de SVs. Pontos na cor cinza representam SVs livres. SVs limitados são ilustrados em preto. Pontos pretos com bordas extras correspondem a SVs limitados que são erros de treinamento. Todos os outros objetos, em branco, são corretamente classificados e encontram-se fora das margens, possuindo $\xi_i^* = 0$ e $\alpha_i^* = 0$.

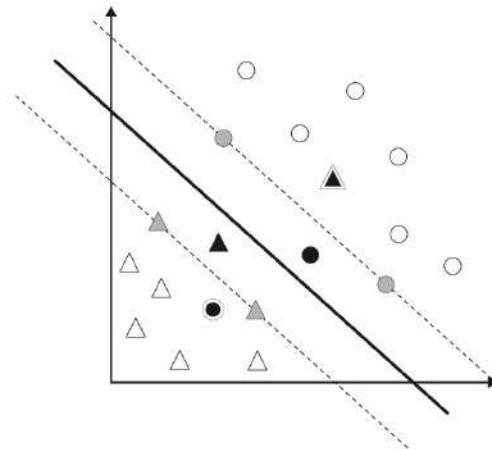


Figura 7.12 Tipos de SVs: livres (cor cinza) e limitados (cor preta).

Para calcular b^* , computa-se a média da Equação 7.27 sobre todos SVs \mathbf{x}_j entre as margens, ou seja, com $\alpha_j^* < C$.

Tem-se como resultado final a mesma função de classificação representada na Equação 7.28, porém nesse caso as variáveis α_i^* são determinadas pela solução da Equação 7.31.

7.2.3 SVMs Não Lineares

As SVMs lineares são eficazes na classificação de conjuntos de dados linearmente separáveis ou que possuam uma distribuição aproximadamente linear, e a versão de margens suaves tolera a presença de alguns ruídos e *outliers*. Porém, há muitos casos em que não é possível dividir satisfatoriamente os dados de treinamento por um hiperplano. Um exemplo é apresentado na Figura 7.13, em que o uso de uma fronteira curva seria mais adequada na separação das classes.

As SVMs lidam com problemas não lineares mapeando o conjunto de treinamento de seu espaço original, referenciado como de entradas, para um novo espaço de maior dimensão, denominado espaço de características (*feature space*) (Hearst et al., 1998). Seja $\Phi : X \rightarrow \mathfrak{X}$ um mapeamento, em que X é o espaço de entradas e \mathfrak{X} denota o espaço de características. A escolha apropriada de Φ faz com que o conjunto de treinamento mapeado em \mathfrak{X} possa ser separado por uma SVM linear.

O uso desse procedimento é motivado pelo teorema de Cover (Haykin, 1999). Dado um conjunto de dados não linear no espaço de entradas X , esse teorema afirma que X pode ser transformado em um espaço de características \mathfrak{X} no qual com alta probabilidade os objetos são linearmente separáveis. Para isso, duas condições devem ser satisfeitas. A primeira é que a transformação seja não linear, enquanto a segunda é que a dimensão do espaço de características seja suficientemente alta.

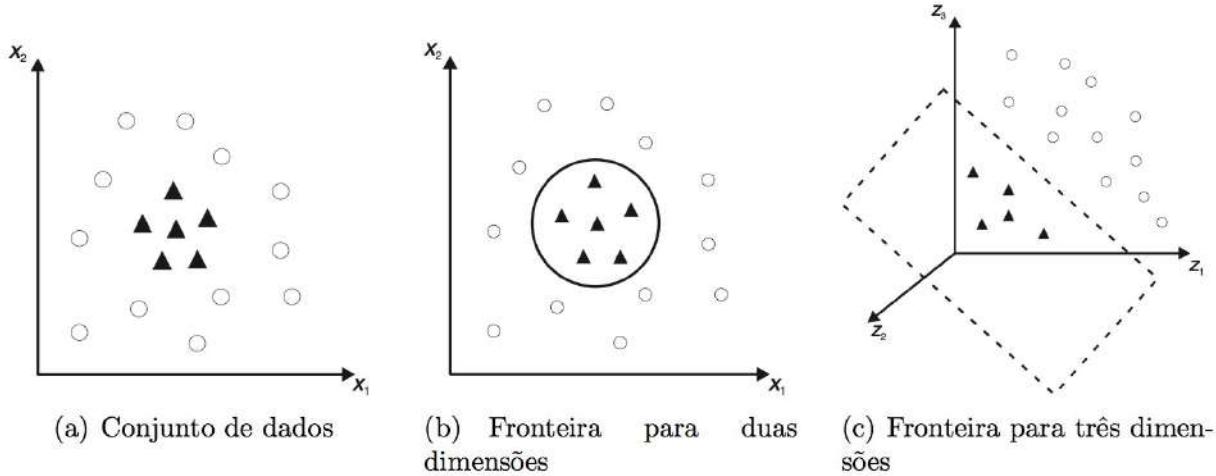


Figura 7.13 Exemplo de transformação realizada em conjunto de dados não linear para espaço de características.

Para ilustrar esses conceitos, considere o conjunto de dados apresentado na Figura 7.13(a) (Müller et al., 2001). Transformando os objetos de \mathbb{R}^2 para \mathbb{R}^3 com o mapeamento representado na Equação 7.36, o conjunto de dados não linear em \mathbb{R}^2 torna-se linearmente separável em \mathbb{R}^3 (Figura 7.13(c)). É possível então encontrar um hiperplano capaz de separar esses objetos, descrito na Equação 7.37. Pode-se verificar que a função apresentada, embora linear em \mathbb{R}^3 (Figura 7.13(c)), corresponde a uma fronteira não linear em \mathbb{R}^2 (Figura 7.13(b)).

$$\Phi(\mathbf{x}) = \Phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \quad (7.36)$$

$$h(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + b = w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 + b = 0 \quad (7.37)$$

Logo, mapeiam-se inicialmente os objetos para um espaço de maior dimensão utilizando Φ e aplica-se a SVM linear sobre esse espaço. Ela encontra o hiperplano com maior margem de separação, garantindo assim uma boa generalização. Utiliza-se a versão de SVM linear com margens suaves, que permite lidar com ruídos e *outliers* presentes nos dados. Para realizar o mapeamento, basta aplicar Φ aos exemplos presentes no problema de otimização representado na Equação 7.31, conforme ilustrado a seguir:

$$\text{Maximizar}_{\boldsymbol{\alpha}} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) \quad (7.38)$$

De forma semelhante, o classificador extraído se torna:

$$g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \text{sgn} \left(\sum_{\mathbf{x}_i \in \text{SV}} \alpha_i^* y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b^* \right) \quad (7.39)$$

em que b^* é adaptado da Equação 7.27 também aplicando o mapeamento aos objetos:

$$b^* = \frac{1}{n_{\text{SV}: \alpha^* < C}} \sum_{\mathbf{x}_j \in \text{SV}: \alpha_j^* < C} \left(\frac{1}{y_j} - \sum_{\mathbf{x}_i \in \text{SV}} \alpha_i^* y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \right) \quad (7.40)$$

Como \mathfrak{S} pode ter dimensão muito alta (até mesmo infinita), a computação de Φ pode ser extremamente custosa ou inviável. Porém, percebe-se, pelas Equações 7.38, 7.39 e 7.40, que a única informação necessária sobre o mapeamento é de como realizar o cálculo de produtos escalares entre os objetos no espaço de características. Isso é obtido com o uso de funções denominadas kernels.

Um kernel K é uma função que recebe dois pontos \mathbf{x}_i e \mathbf{x}_j no espaço de entradas e calcula o produto escalar desses objetos no espaço de características (Herbrich, 2001). Tem-se então:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \quad (7.41)$$

Para o mapeamento apresentado na Equação 7.36 e dois objetos \mathbf{x}_i e \mathbf{x}_j em \mathfrak{R}^2 , por exemplo, o kernel é dado por:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2 \quad (7.42)$$

É comum empregar a função kernel sem conhecer o mapeamento Φ , que é gerado implicitamente. A utilidade dos kernels está, portanto, na simplicidade de seu cálculo e em sua capacidade de representar espaços abstratos.

Para garantir a convexidade do problema de otimização formulado na Equação 7.38 e também que o kernel represente mapeamentos nos quais seja possível o cálculo de produtos escalares conforme a Equação 7.41, utilizam-se funções kernel que seguem as condições estabelecidas pelo teorema de Mercer (Mercer, 1909). De forma simplificada, um kernel que satisfaz as condições de Mercer é caracterizado por dar origem a matrizes positivas semidefinidas \mathbf{K} , em que cada elemento K_{ij} é definido por $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, para todo $i, j = 1, \dots, n$ (Herbrich, 2001).

Alguns dos kernels mais utilizados na prática são os polinomiais, os de função base radial (*radial basis function* - RBF) e os sigmoidais, listados na Tabela 7.1. Cada um deles apresenta parâmetros que devem ser determinados pelo usuário, indicados também na tabela. O kernel sigmoidal, em particular, satisfaz as condições de Mercer apenas para alguns valores de δ e κ . O kernel polinomial com $d = 1$, $\delta = 1$ e $\kappa = 0$ também é denominado linear, e seu uso implica não realizar um mapeamento dos dados, ou seja, a obtenção de uma SVM linear.

Tabela 7.1 Funções kernel mais comuns

Tipo de kernel	Função $K(\mathbf{x}_i, \mathbf{x}_j)$	Parâmetros
Polinomial	$(\delta(\mathbf{x}_i \cdot \mathbf{x}_j) + \kappa)^d$	δ , κ e d
RBF	$\exp(-\sigma \ \mathbf{x}_i - \mathbf{x}_j\ ^2)$	σ
Sigmoidal	$\tanh(\delta(\mathbf{x}_i \cdot \mathbf{x}_j) + \kappa)$	δ e κ

A obtenção de um classificador por meio do uso de SVMs envolve então a escolha de uma função kernel, além de parâmetros dessa função e do valor da constante de regularização C . A escolha do kernel e dos parâmetros considerados afeta o desempenho do classificador obtido, pois eles definem a fronteira de decisão induzida. Segundo observado em Hsu et al. (2003), uma boa escolha inicial é empregar o kernel RBF, uma vez que o kernel linear é apontado como um caso especial de função RBF (Keerthi e Lin, 2003) e

o kernel sigmoidal pode ter comportamento semelhante ao RBF para certos parâmetros (Lin e Lin, 2003).

O Algoritmo 7.3 resume então a formulação final seguida pelas SVMs em seu treinamento.

Algoritmo 7.3 Algoritmo de treinamento de SVM

Entrada: Um conjunto de n objetos de treinamento

Saída: SVM treinada

1 Seja $\alpha^* = (\alpha_1^*, \dots, \alpha_n^*)$ a solução de:

2 Maximizar: $\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$

3 Sob as restrições: $\begin{cases} \sum_{i=1}^n y_i \alpha_i = 0 \\ 0 \leq \alpha_i \leq C, i = 1, \dots, n \end{cases}$

4 O classificador é dado por: $g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \text{sgn}\left(\sum_{\mathbf{x}_i \in \text{SV}} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^*\right)$

5 Em que: $b^* = \frac{1}{n_{\text{SV}: \alpha^* < C}} \sum_{\mathbf{x}_j \in \text{SV}: \alpha_j^* < C} \left(\frac{1}{y_j} - \sum_{\mathbf{x}_i \in \text{SV}} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}_j) \right)$

7.2.4 SVMs em Problemas de Regressão

Embora as descrições anteriores tenham sido focadas em problemas de classificação, as SVMs também podem ser aplicadas na solução de problemas de regressão e no agrupamento de dados (aprendizado não supervisionado). Contudo, o problema de otimização para o seu treinamento deve ser reformulado para lidar com as características e objetivos desses problemas. Nesta seção, o uso de SVMs em problemas de regressão é brevemente descrito, tomando como base o tutorial (Smola e Schölkopf, 1998).

O algoritmo ε -SVR (*support vector regression*) (Vapnik, 1995) tem como objetivo encontrar uma função $h(\mathbf{x})$ que produza saídas contínuas para os dados de treinamento que desviam no máximo de ε de seu rótulo desejado. Essa função deve também ser o mais uniforme e regular possível.

Seguindo a estrutura apresentada para o caso de classificação, consideremos primeiro o uso de funções lineares h (Equação 7.14). Nesse caso, a regularidade se reflete em buscar uma função com pequeno \mathbf{w} , o que pode ser conseguido pela minimização da norma $\|\mathbf{w}\|$. Tem-se então o problema de otimização:

$$\underset{\mathbf{w}, b}{\text{Minimizar}} \quad \frac{1}{2} \|\mathbf{w}\|^2 \quad (7.43)$$

$$\text{Com as restrições: } \begin{cases} y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon_i \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon_i \end{cases} \quad (7.44)$$

Procura-se então a função linear que aproxime os pares (\mathbf{x}_i, y_i) de treinamento com uma precisão de ε . Na Figura 7.14 é apresentada uma ilustração do procedimento realizado.

Busca-se a função linear tal que os dados de treinamento fiquem dentro de uma região ao redor de h , representada em sombreado na figura.

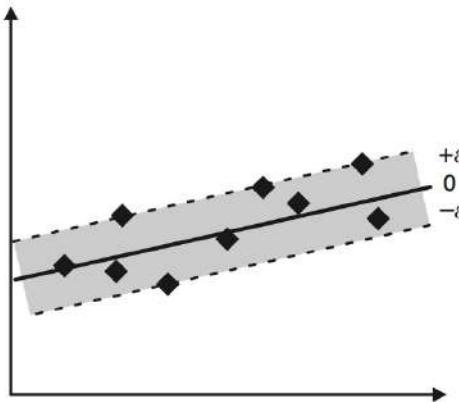


Figura 7.14 Ilustração simplificada de procedimento realizado por SVR.

Analogamente ao caso das SVMs de margens suaves, esse problema pode ser relaxado com a introdução de variáveis de folga, permitindo assim lidar com ruídos e *outliers* nos objetos. As variáveis de folga permitem que alguns exemplos fiquem fora da região entre $-\varepsilon$ e $+\varepsilon$. Tem-se então:

$$\underset{\mathbf{w}, b, \xi, \bar{\xi}}{\text{Minimizar}} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \left(\sum_{i=1}^n \xi_i + \bar{\xi}_i \right) \quad (7.45)$$

$$\text{Com as restrições: } \begin{cases} y_i - \mathbf{w} \cdot \mathbf{x}_i - b \leq \varepsilon + \xi \\ \mathbf{w} \cdot \mathbf{x}_i + b - y_i \leq \varepsilon + \bar{\xi} \\ \xi_i, \bar{\xi}_i \geq 0 \end{cases} \quad (7.46)$$

Nas equações apresentadas, ξ_i e $\bar{\xi}_i$ representam as variáveis de folga e C é uma constante que impõe um *trade-off* entre a regularidade de h e o quanto de desvios são tolerados. Como no caso das SVMs para classificação, monta-se o problema dual equivalente ao anterior pelo uso de uma lagrangiana, tornando nulo o resultado das derivações parciais e substituindo as expressões resultantes na equação lagrangiana inicial.

O problema dual obtido é descrito em termos de produtos internos entre os objetos. Pode-se então recorrer ao uso de kernels para realizar regressões não lineares. O uso do kernel implica o mapeamento dos objetos para um espaço de características, onde então a função linear mais regular e com baixo erro de treinamento é encontrada. O problema de otimização final solucionado é dado por:

$$\underset{\boldsymbol{\alpha}, \bar{\boldsymbol{\alpha}}}{\text{Maximizar}} \quad -\frac{1}{2} \sum_{i,j=1}^n (\alpha_i - \bar{\alpha}_i)(\alpha_j - \bar{\alpha}_j) K(\mathbf{x}_i, \mathbf{x}_j) - \varepsilon \sum_{i=1}^n (\alpha_i + \bar{\alpha}_i) + \sum_{i=1}^n y_i (\alpha_i - \bar{\alpha}_i) \quad (7.47)$$

$$\text{Com as restrições: } \begin{cases} \sum_{i=1}^n (\alpha_i - \bar{\alpha}_i) = 0 \\ \alpha_i, \bar{\alpha}_i \in [0, C] \end{cases} \quad (7.48)$$

Nas equações apresentadas, α_i e $\bar{\alpha}_i$ representam as variáveis de Lagrange e K é a função kernel, que deve satisfazer as condições de Mercer (os mesmos tipos de kernel da Tabela 7.1 podem ser empregados para SVR). As variáveis de Lagrange associadas a todos os exemplos que se encontram dentro da margem entre $-\varepsilon$ e $+\varepsilon$ são nulas. Os outros casos correspondem aos SVs.

7.2.5 Discussão: Vantagens e Desvantagens

Com princípios embasados na teoria de aprendizado estatístico, as SVMs caracterizam-se por apresentar uma boa capacidade de generalização. Elas também são robustas diante de objetos de grande dimensão, sobre os quais outras técnicas de aprendizado comumente obtêm classificadores super ou subajustados. Outra característica atrativa é a convexidade do problema de otimização formulado em seu treinamento, que implica a existência de um único mínimo global. Essa é uma vantagem das SVMs sobre, por exemplo, as RNAs MLP, em que há mínimos locais na função objetivo minimizada. Além disso, o uso de funções kernel na não linearização das SVMs torna o algoritmo eficiente, pois permite a construção de simples hiperplanos em um espaço de alta dimensão de forma tratável do ponto de vista computacional (Burges, 1998).

Entre as principais limitações das SVMs encontram-se a sua sensibilidade a escolhas de valores de parâmetros e a dificuldade de interpretação do modelo gerado por essa técnica. Esses problemas foram tratados em alguns trabalhos, tais como Chapelle et al. (2002); Duan et al. (2003) e Fu et al. (2004).

Observou-se, no decorrer desta seção, que o raciocínio empregado pelas SVMs na obtenção do classificador final leva a um problema de otimização dual em termos dos dados de treinamento. Porém, a forma de solução desse problema não foi apresentada. Existem diversos pacotes matemáticos capazes de solucionar problemas quadráticos com restrições. Contudo, eles geralmente não são adequados a aplicações de AM, que em geral se caracterizam pela necessidade de lidar com um grande volume de dados. Diversas técnicas e estratégias foram então propostas para adaptar a solução do problema de otimização das SVMs a aplicações de larga escala. Em geral, recorre-se a alguma estratégia decomposicional, em que subproblemas menores são otimizados a cada passo do algoritmo. Uma discussão mais detalhada a respeito dos métodos e algoritmos comumente empregados nesse processo pode ser encontrada em Cristianini e Shawe-Taylor (2000).

Nesta seção nos limitamos também a apresentar a formulação original das SVMs, a qual é capaz de lidar apenas com problemas de classificação binários. Existe uma série de técnicas que podem ser empregadas na generalização das SVMs para a solução de problemas multiclasse. Pode-se recorrer à decomposição do problema multiclasse em vários subproblemas binários ou reformular o algoritmo de treinamento das SVMs em versões multiclasse. Em geral, esse último procedimento leva a algoritmos computacionalmente custosos (Hsu e Lin, 2002). Por esse motivo, a estratégia decomposicional é empregada mais frequentemente. No Capítulo 17 é apresentada uma revisão de técnicas para a decomposição de problemas multiclasse em múltiplos subproblemas binários.

7.3 Considerações Finais

Este capítulo apresentou duas técnicas de AM que se valem de um processo de otimização para a obtenção de um modelo preditivo a partir de um conjunto de dados: as RNAs e as SVMs.

As RNAs mais usualmente empregadas na prática são as redes MLP, cujo algoritmo de treinamento mais popular é o *back-propagation*. O *back-propagation* é baseado no método de otimização gradiente descendente e busca minimizar o erro quadrático entre os rótulos desejados dos objetos e os produzidos pela rede. No caso das SVMs, recorre-se a um problema de otimização quadrático em função dos dados de treinamento, o qual busca maximizar a margem de separação entre as classes. Esse problema apresenta uma única solução global, enquanto no caso das RNAs MLP a função de erro minimizada pode possuir mínimos locais e o método gradiente pode convergir para um deles, dependendo de seus parâmetros iniciais.

Ambas as técnicas são consideradas “caixas-pretas”, no sentido de que o conhecimento extraído dos objetos por elas se encontra codificado em equações de difícil interpretação, em contraste com os modelos gerados por técnicas simbólicas como as árvores de decisão. Ambas também possuem parâmetros a serem estimados pelo usuário que afetam diretamente o seu desempenho, embora o número de parâmetros seja maior no caso das RNAs.

Outra desvantagem das RNAs perante as SVMs é que seus resultados são estocásticos e dependem fortemente da ordem de apresentação dos objetos e dos pesos iniciais atribuídos a suas conexões. Dessa forma, é recomendável executá-la várias vezes para diferentes configurações dos dados e valores iniciais de pesos, obtendo uma média de desempenho. As SVMs, por outro lado, são determinísticas, e seus resultados não dependem da ordem em que os objetos se encontram no conjunto empregado em seu aprendizado. Por outro lado, foi observado que as SVMs podem ter tempos de predição maiores que as RNAs (LeCun et al., 1995).

É possível observar ainda que as RNAs e as SVMs lidam apenas com atributos com valores numéricos. No caso de dados categóricos, uma codificação numérica deve ser realizada. Em geral, a codificação mais usada é a um-para-n, em que se tem um bit para cada possível valor que o atributo categórico pode assumir. Além disso, é recomendável normalizar os atributos contínuos para evitar o domínio de atributos em intervalos numéricos maiores sobre aqueles em intervalos menores.

De forma geral, as SVMs e RNAs apresentam um bom desempenho preditivo em diversas tarefas de classificação, figurando entre as técnicas mais utilizadas em problemas que requerem alta precisão. Ambas são ainda consideravelmente tolerantes a ruídos. No caso das SVMs, tem-se também uma robustez a dados de alta dimensionalidade. As duas técnicas podem também ser adaptadas para gerar probabilidades de classificação dos exemplos (Haykin, 1999; Platt, 2000).

Capítulo 8

Modelos Múltiplos Preditivos

O termo *modelos múltiplos* é utilizado para identificar um conjunto de preditores cujas decisões individuais são combinadas ou agregadas de alguma forma para prever novos exemplos (Dietterich, 1997). Neste capítulo, iremos focar problemas de classificação. No entanto, todas as técnicas apresentadas são válidas para problemas de regressão com alterações triviais relacionadas com a função de custo utilizada.

Segundo Hoeting et al. (1999), a primeira referência a uma combinação de modelos na literatura estatística aparece em Barnard (1963), um trabalho que estuda os dados de passageiros de companhias aéreas. Entretanto, a maior parte dos trabalhos anteriores na área de combinação de modelos não está presente em revistas científicas de Estatística. O artigo original sobre previsões feito por Granger e Newbold (1976) estimulou, nos anos 1970, uma onda de artigos na literatura de Economia sobre combinação de previsões de diferentes modelos de previsão.

A ideia principal por trás de qualquer modelo múltiplo é baseada na observação de que diferentes algoritmos de aprendizado exploram:

- Diferentes linguagens de representação;
- Diferentes espaços de procura;
- Diferentes funções de avaliação de hipóteses.

Como é possível explorar essas diferenças? É possível desenvolver um conjunto de classificadores que, trabalhando juntos, obtêm um melhor desempenho do que cada classificador trabalhando individualmente? Essas são algumas questões que vamos abordar neste capítulo.

Sabe-se que não existe um algoritmo que seja melhor para todos os problemas de decisão. Essa observação é baseada em resultados teóricos (o teorema *no-free lunch*, Wolpert e Macready (1997)) e em resultados experimentais do projeto Statlog (Michie et al., 1994). A primeira observação que pode ser feita quando se trabalha com modelos múltiplos é que combinar avaliadores idênticos é inútil. Assim, uma condição necessária para que a proposta de combinação de classificadores seja útil é que avaliadores individuais tenham um nível substancial de desacordo.

Hansen e Salamon (1990) introduziram a hipótese de que modelos múltiplos são mais úteis quando os modelos constituintes cometem erros independentes. Os autores provaram que, quando *i*) todos os modelos têm a mesma taxa de erro, *ii*) a taxa de erro é menor que 0,5, *iii*) todos cometem erros completamente independentes, então o erro esperado do conjunto decresce linearmente com o número de modelos.

Considerando a tarefa de classificação, a Figura 8.1(a) mostra a evolução da taxa de erro obtida variando o número de classificadores no conjunto. Esse é um estudo de simulação, num problema de duas classes equiprováveis, ou seja, a probabilidade de observar cada classe é 50%. Nesse estudo, consideram-se conjuntos de classificadores com tamanhos que variam entre 3 e 24.

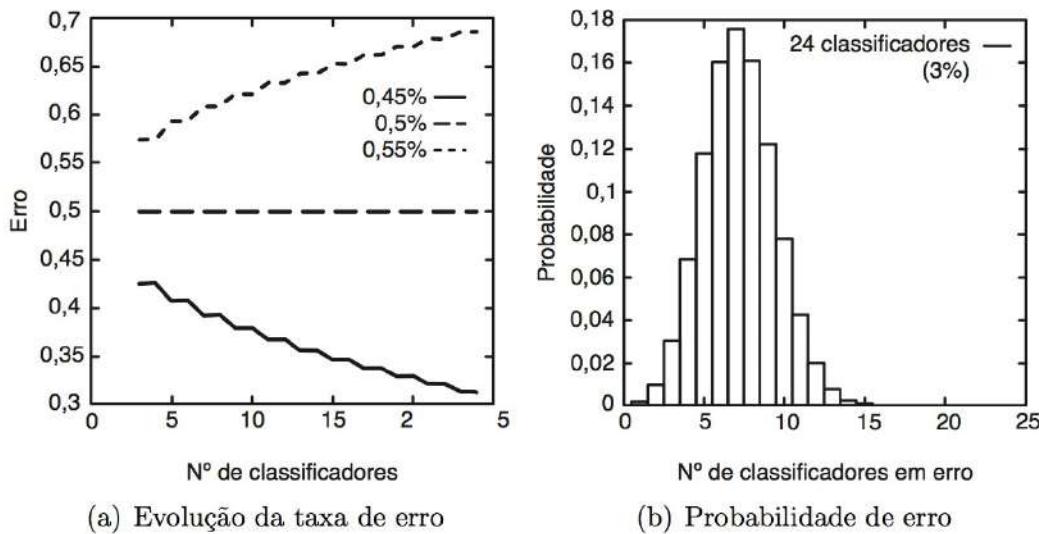


Figura 8.1 (a) Evolução da taxa de erro variando o número de classificadores em um modelo múltiplo. (b) Probabilidade de que exatamente i dos 24 classificadores cometem um erro.

Todos os classificadores classificam os exemplos de teste. A classificação final do conjunto é obtida agregando os votos dos classificadores por votação uniforme, isto é, o voto de classificador tem peso 1. Ou seja, para cada exemplo, a classificação do conjunto é a classe mais votada pelos classificadores individuais. Todos os classificadores têm a mesma probabilidade de cometer um erro, mas os erros são independentes uns dos outros. Quando essa probabilidade é 45%, a taxa de erro do conjunto decresce linearmente. Quando a probabilidade é 55%, a taxa de erro cresce linearmente, e quando a probabilidade é igual a 50%, a taxa de erro do conjunto permanece constante. Se as taxas de erro de nc classificadores são todas iguais a $p < 0,5$ e se os erros são independentes, então a probabilidade de a votação por maioria estar errada pode ser calculada como a área sob a curva da distribuição binomial para o caso em que mais de $nc/2$ classificadores estão errados (Dietterich, 1997). A Figura 8.1(b) mostra essa área para a simulação de 24 classificadores, cada um tendo um erro de 30%. A área abaixo da curva para mais de 12 classificadores é 0,02, que é bem menor que a taxa de erro de classificadores individuais (0,3).

Tumer e Ghosh (1995, 1996a,b) mostram como a taxa de erro obtida por um combinador de nc classificadores está relacionada à taxa de erro de um único classificador. Essa relação pode ser observada na Equação 8.1, em que ρ denota a correlação entre erros de classificadores e $Error_{Bayes}$ é a taxa de erro obtida usando a regra de Bayes, assumindo que todas as probabilidades condicionais são conhecidas. Se $\rho = 0$, o erro do conjunto decresce proporcionalmente ao número de classificadores. Quando $\rho = 1$, o erro do conjunto é igual ao erro de um único classificador.

$$Error_{conjunto} = \frac{1 + \rho(nc - 1)}{nc} Error + Error_{Bayes} \quad (8.1)$$

Ainda que os pressupostos da análise teórica difiram das aplicações do mundo real, duas ideias principais que surgiram a partir desse estudo são:

- Combinar modelos que cometam erros não correlacionados ou preferencialmente negativamente correlacionados;
- Cada modelo deve ter um melhor desempenho do que uma escolha aleatória.

Duas questões surgem quando se trabalha com *modelos múltiplos*. A primeira delas é – *como combinar as previsões de diferentes modelos?* A segunda questão é – *como gerar diferentes modelos?* Essas questões serão abordadas nas próximas seções.¹

8.1 Combinando Previsões de Classificadores

Suponha que para um determinado problema de aprendizado de classificação, temos disponível um conjunto de classificadores. Nesta seção, não importa como eles foram treinados. O conjunto de classificadores podem ser pessoas, especialistas do domínio ou um modelo de decisão treinado por algoritmos de aprendizado. Denotamos o conjunto de classificadores como *classificadores de base*.

Suponha agora um exemplo teste. Cada classificador de base faz uma previsão para esse exemplo. Como podemos combinar as previsões? Quais são as vantagens de fazer essa combinação? Por que, como e quando devemos usar um esquema de combinação? A primeira observação relevante é que os classificadores a combinar têm de fazer previsões de forma independente, ou seja, fazer previsões não correlacionadas.

Diz-se que dois modelos cometem um *erro correlacionado* quando *ambos* classificam um exemplo da classe y_a como pertencente à classe y_b , $y_a \neq y_b$. Ali e Pazzani (1996) apresentam uma definição precisa de erros correlacionados: Dado um conjunto de classificadores $F = \{\hat{f}_1(\mathbf{x}), \dots, \hat{f}_{nc}(\mathbf{x})\}$, $\hat{f}_i(\mathbf{x}) = y$ denota que o modelo i classificou o exemplo \mathbf{x} na classe y . $f(\mathbf{x})$ denota a verdadeira classe de \mathbf{x} . A *correlação de erro* entre dois classificadores i e j é definida como a probabilidade que os modelos \hat{f}_i e \hat{f}_j cometem o mesmo erro:

$$\phi_{ij} = p(\hat{f}_i(\mathbf{x}) = \hat{f}_j(\mathbf{x}), \hat{f}_i(\mathbf{x}) \neq f(\mathbf{x})). \quad (8.2)$$

O grau em que os erros em F são correlacionados, $\phi_e(F)$, possui a seguinte definição:

$$\phi_e(F) = \frac{1}{nc(nc-1)} \sum_{i=1}^{nc} \sum_{j \neq i}^{nc} \phi_{ij} \quad (8.3)$$

em que nc representa o número de modelos no conjunto. Essa definição de *correlação de erro* não satisfaz a propriedade de que a correlação entre um objeto e ele mesmo deve ser 1. Para ultrapassar essa dificuldade, definimos correlação de erro entre pares de classificadores como a probabilidade condicional de dois classificadores cometem o mesmo erro dado que um deles comete um erro. Essa definição de *correlação de erro* reside

¹ Os Capítulos 17, 18 e 19 também ilustram combinações de múltiplos modelos de classificação. Contudo, nesses casos é abordada a resolução de problemas de classificação com várias classes, e distintas relações entre elas, por meio da combinação de classificadores induzidos na solução de subproblemas mais simples.

no intervalo $[0, 1]$, e a correlação entre um classificador e ele mesmo é 1. A definição formal é:

$$\phi_{ij} = p(\hat{f}_i(\mathbf{x}) = \hat{f}_j(\mathbf{x}) | \hat{f}_i(\mathbf{x}) \neq f(\mathbf{x}) \vee \hat{f}_j(\mathbf{x}) \neq f(\mathbf{x})). \quad (8.4)$$

Levando em consideração que ϕ_{ij} é simétrico, a Equação 8.3 pode ser reescrita como:

$$\phi_e(F) = \frac{2}{nc(nc - 1)} \sum_{i=1}^{nc} \sum_{j>i}^{nc} \phi_{ij} \quad (8.5)$$

Quando combinamos as previsões dos classificadores, podemos diferenciar: **métodos de votação versus métodos de seriação**. A diferença está no tipo de saída do classificador individual. No primeiro método, o classificador de base produz um rótulo de classe. No segundo, a saída do classificador de base é probabilística, isto é, ela associa, para cada exemplo teste, uma probabilidade (ou um fator confiante) para cada possível classe. Noutra dimensão distinguimos **métodos dinâmicos versus métodos estáticos**. O esquema de combinação leva em consideração o exemplo de teste. Enquanto métodos estáticos combinam a predição de todos os classificadores no conjunto, métodos dinâmicos selecionam os classificadores mais apropriados para o exemplo de teste. Para diferentes exemplos de teste, métodos dinâmicos podem usar diferentes conjuntos de classificadores para fazer a previsão final.

8.1.1 Métodos de Votação *versus* Métodos de Seriação

Na literatura de reconhecimento de padrões, os métodos discutidos nesta seção aparecem com o nome de *fusão de classificadores* (Kittler, 1998). Distinguimos entre classificadores cuja previsão é uma etiqueta para a classe e classificadores cuja previsão assume a forma de distribuição de probabilidades para todos os possíveis valores da classe.

Métodos de Votação

Votação é o método mais comumente usado para combinar classificadores. Como destacado por Ali e Pazzani (1996), essa estratégia é motivada pela teoria do aprendizado bayesiano, que determina que a fim de maximizar a precisão da predição, em vez de usar apenas um único modelo de decisão, idealmente devem ser usados todos os modelos aceitáveis no espaço da hipótese. O voto de cada hipótese deve ser ponderado pela probabilidade posterior daquela hipótese, dado o conjunto de treinamento. Muitas variações dos métodos de votação podem ser encontradas na literatura de Aprendizado de Máquina. Uma delas é a *votação uniforme*, em que a opinião de todos os classificadores de base contribui igualmente para a classificação final. Outra variação comum é a *votação com peso*, em que cada classificador de base tem um peso associado, que pode mudar ao longo do tempo, e reforçando assim a classificação atribuída a um *bom* classificador.²

Métodos de Seriação

Uma melhoria na votação uniforme é obtida quando cada classificador pode produzir uma estimativa da *probabilidade* de o exemplo pertencer a uma classe, em vez de produzir

²Domingos (1997a) argumenta que o sucesso da proposta de modelos múltiplos é devido primeiramente à redução da variação e não pelo fato de ser uma melhor aproximação da teoria de aprendizado bayesiano.

uma única etiqueta. Dado um exemplo de teste \mathbf{x} , cada classificador probabilístico reporta a probabilidade de o exemplo pertencer a cada uma das classes: p_1, \dots, p_m . Dado um conjunto de m classificadores probabilísticos, as probabilidades de classe de todos os modelos podem ser combinadas, por exemplo, usando a fórmula: $\frac{1}{m} \sum_{i=1}^m p_i$. Esse método é conhecido na literatura como *soma da distribuição*. Kittler (1998) apresentou uma perspectiva teórica nesse problema. Ele examinou a sensibilidade de várias combinações de regras para estimar erros, e apresentou um enquadramento unificando algumas das regras mais usuais, como a *regra do produto*, a *regra da soma*, a *regra do mínimo*, a *regra do máximo* e a *regra da mediana*.

Kittler (1998) estudou várias estratégias para a fusão de m classificadores probabilísticos em problemas com k classes. Assumindo que representamos por P_{ik} a probabilidade dada pelo classificador i de o exemplo ser da classe k , então:

- Regra da soma: $S_k = \sum_{i=1}^m P_{ik}$
- Regra da média: $S_k = \sum_{i=1}^m P_{ik}/m$
- Regra da média geométrica: $S_j = \sqrt[m]{\prod_{i=1}^m P_{ik}}$
- Regra do produto: $S_k = \prod_{i=1}^m P_{ik}$
- Regra do máximo: $S_k = \max_i P_{ik}$
- Regra do mínimo: $S_k = \min_i P_{ik}$

Após a aplicação de uma das estratégias de fusão enumeradas, cada exemplo é classificado na classe que maximize S_k .

A análise apresentada em Kittler (1998) conclui que a regra da soma é conservadora, mas largamente usada, enquanto a regra do produto é mais arriscada, mas pode produzir melhores resultados. Uma alternativa interessante consiste em tentar reduzir o conjunto de possíveis rótulos (classes) para cada exemplo de teste.

8.1.2 Métodos Dinâmicos *versus* Métodos Estáticos

A distribuição da taxa de erros sobre o espaço de atributos geralmente não é homogênea. Dependendo do classificador, a taxa de erro será mais concentrada em certas regiões do espaço de objetos do que em outras. Os métodos estáticos têm em conta as previsões de todos os elementos do conjunto, enquanto os métodos dinâmicos levam em consideração o exemplo de teste e realizam uma *seleção do modelo* para classificar o dado exemplo de teste. A seguir serão detalhados dois métodos dinâmicos: MAI e SCANN.

MAI: *Model Applicability Induction*

Ortega (1995) apresenta a proposta chamada *Model Applicability Induction* para combinar previsões de múltiplos modelos. A proposta consiste em caracterizar as situações em que cada modelo é capaz de fazer previsões corretas. Isso é feito através do aprendizado de um *metaclassificador* para cada modelo disponível da base. O objetivo desse metaclassificador é predizer onde o modelo de base classificará corretamente o exemplo de teste.

O algoritmo genérico para construir um *metaclassificador* é apresentado no Algoritmo 8.1. Nesse caso, usamos uma estratégia de avaliação deixar-um-de-fora. Como alternativa, por exemplo, no caso de grandes bancos de dados, pode ser usada validação cruzada (ver Capítulo 9). Nos dados de $Nível_0$, os exemplos positivos são os exemplos corretamente classificados pelo algoritmo de aprendizado de base ϕ , e os exemplos negativos são os incorretamente classificados. Os dados de $Nível_1$ representam sempre um problema de duas classes. As regras de classificação geradas usando os dados $Nível_1$ caracterizam os exemplos que o classificador gerado com ϕ classificou corretamente. A Tabela 8.1 ilustra os conjuntos de dados de $Nível_0$ e $Nível_1$. A tabela mostra o conjunto de dados original e o conjunto de dados de $Nível_1$. Nesse último conjunto de dados, a coluna **Erro** indica se o classificador de base classificou corretamente (ou não) o exemplo de treinamento. A Figura 8.2 apresenta as árvores de decisão aprendidas a partir dos dados de $Nível_0$ (8.2(a)) e $Nível_1$ (8.2(b)). A árvore da Figura 8.2(b) caracteriza as regiões do espaço onde a árvore da Figura 8.2(a) classifica corretamente os exemplos.

Algoritmo 8.1 *Model Applicability Induction:* o algoritmo para gerar metaclassificadores

Entrada: Algoritmo de aprendizado de base ϕ
 Algoritmo de aprendizado de nível meta \mathfrak{S}
 Um conjunto de treinamento $D = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$

Saída: Metamodelo para ϕ em D

```

1 /* Gerar dados de  $Nível_1$  data */
2 Dados  $Nível_1 \leftarrow \{\}$ 
3 para cada exemplo  $\mathbf{x}_i \in D$  faça
4    $\hat{f} \leftarrow \phi(D - \{(\mathbf{x}_i, y_i)\})$ 
5   se  $(y_i = \hat{f}(\mathbf{x}_i))$  então
6     Inserir  $\{(\mathbf{x}_i, +)\}$  nos dados de  $Nível_1$ 
7   fim
8   senão
9     Inserir  $\{(\mathbf{x}_i, -)\}$  nos dados de  $Nível_1$ 
10  fim
11 fim
12 Metamodelo  $\leftarrow \mathfrak{S}(Nível_1)$ 
13 Retorna: (Metamodelo)
  
```

Para objetos novos, esses metaclassificadores são primeiramente consultados para selecionar o modelo de predição mais apropriado, e a predição do modelo selecionado é então devolvida.

Repare que os atributos dos dois problemas são os mesmos. Um aspecto interessante dessa arquitetura é que o modelo de metaclassificador é definido em termos dos atributos originais. Ele define as regiões do espaço de objetos onde os classificadores de base são mais (ou menos) propensos a erro.

Um estudo experimental do *Model Applicability Induction* foi apresentado em Seewald e Fürnkranz (2001). Nesse estudo, os autores usam o termo classificador *grading* para MAI. Os autores concluem:

Tabela 8.1 Exemplo ilustrativo do MAI: tabela mostra o conjunto de dados original e o conjunto de dados de Nível₁

V1	V2	V3	V4	V5	Classe	V1	V2	V3	V4	V5	Erro
t	a	c	t	a	membro	t	a	c	t	a	+
t	g	c	t	a	membro	t	g	c	t	a	-
g	t	a	c	t	não membro	g	t	a	c	t	+
a	a	t	t	g	membro	a	a	t	t	g	+
t	c	g	a	t	não membro	t	c	g	a	t	-
a	g	g	g	g	membro	a	g	g	g	g	+

Conjunto de dados original

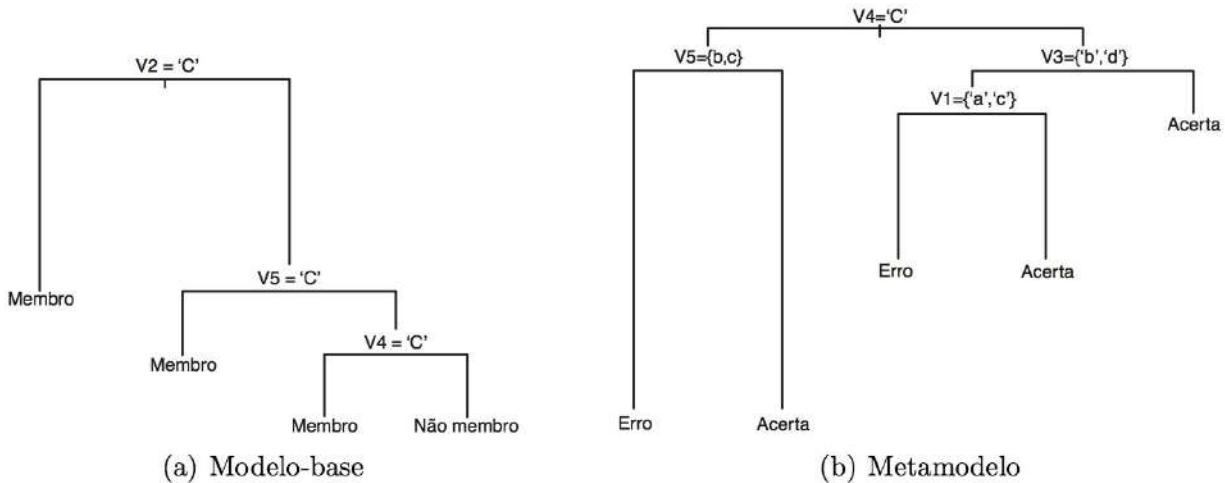
Conjunto de dados Nível₁

Figura 8.2 Exemplo de uma árvore de decisão originada a partir dos dados originais 8.2(a) e um metaclassificador 8.2(b), também na forma de uma árvore de decisão, gerado a partir dos dados de Nível₁.

“Essa proposta pode ser vista como uma generalização direta da seleção por validação cruzada, que sempre seleciona o classificador de base que corresponde ao metaconjunto de dados com a maior precisão padrão. Grading supera o desempenho de estratégias de votação e a seleção por validação cruzada.”

SCANN – Análise da Correspondência Seguida por um Vizinho Mais Próximo

Merz (1998) apresenta um método para combinar modelos para problemas de classificação e regressão. O método consiste em transformar a matriz de previsões ($M \times N$), em que cada linha corresponde a um exemplo de treinamento e cada coluna corresponde a um modelo aprendido, em um novo espaço de objetos de menor dimensão. Essa transformação é feita usando técnicas de *Análise de Correspondência*. É usada a *decomposição singular de valor* (SVD) para decompor a matriz de previsões em um conjunto de *modelos básicos*, não correlacionados. A decomposição pode detectar modelos redundantes, e caracteriza as áreas do espaço de exemplos onde cada modelo é superior. A representação baseada em SVD ajuda a evitar os problemas associados a previsões correlacionadas sem

descartar nenhum modelo aprendido. A estratégia do vizinho mais próximo é então usada para classificar exemplos de teste nesse novo espaço de representação.

8.2 Combinando Classificadores Homogêneos

Nesta seção são analisados métodos que combinam modelos gerados por um único algoritmo. *Diversidade* é um dos requisitos quando são usados modelos múltiplos. Várias estratégias foram propostas para geração de classificadores diferentes usando o mesmo algoritmo de aprendizado. A maioria delas manipula o conjunto de treinamento para gerar múltiplas hipóteses. O algoritmo de aprendizado executa várias vezes, utilizando cada vez uma distribuição diferente de exemplos de treinamento. Essa técnica funciona especialmente bem para algoritmos de aprendizado *instáveis* – algoritmos cuja saída do classificador sofre grandes mudanças em resposta a pequenas mudanças nos dados do treinamento. Os métodos para combinar classificadores homogêneos podem ser agrupados pela maneira como geram diversidade nos classificadores de base: por amostragem dos objetos, amostragem dos atributos, injeção de aleatoriedade e perturbação dos exemplos de teste. Métodos baseados em cada uma dessas estratégias são apresentados a seguir.

8.2.1 Métodos Baseados em Amostragem dos Exemplos de Treinamento

Bootstrap Aggregating - Bagging.

Breiman (1996a) descreve a técnica chamada *Bootstrap Aggregating - bagging*, que produz replicações do conjunto de treinamento por amostragem com reposição. Cada réplica do conjunto de treinamento tem o mesmo tamanho que os dados originais. Alguns exemplos não aparecem na amostra, enquanto outros podem aparecer mais de uma vez. Tal conjunto de treinamento é chamado *bootstrap replicado* do conjunto de dados original, e a técnica é chamada *agregação bootstrap* (da qual o termo *bagging* é derivado). Para um conjunto de treinamento com n exemplos, a probabilidade de um exemplo ser selecionado é $1 - (1 - 1/n)^n$. Para um valor de n grande, a probabilidade é cerca de $1 - 1/e$, em que e é a base de logaritmos naturais. Cada amostra contém, em média, 36,8% ($1/e$) de exemplos duplicados. Para cada réplica do conjunto de treinamento um classificador é gerado. Todos os classificadores são usados para classificar cada exemplo no conjunto de teste, e a classificação final do exemplo é feita geralmente usando um esquema de voto uniforme. O Algoritmo 8.2 detalha o funcionamento da técnica *bagging*.

Por que o *bagging* funciona? Intuitivamente, considerando o voto majoritário de diferentes hipóteses, a variabilidade aleatória dos classificadores individuais é reduzida. As árvores de decisão são um algoritmo instável, exatamente o tipo de algoritmo para o qual podemos obter uma grande melhora com *bagging*. Quando se induz uma árvore de decisão, há pelo menos duas situações afetadas pelo *bagging*. A primeira delas é a escolha do atributo que divide cada nó. Se dois ou mais atributos avaliam de forma similar, com relação a uma dada função de avaliação, uma pequena mudança nos dados de treinamento pode mudar o atributo escolhido. Todas as árvores descendentes também serão mudadas. *Bagging* afeta também a escolha do *ponto de corte*, para atributos *contínuos*.

Algoritmo 8.2 O algoritmo de *bagging*

Entrada: Um algoritmo de aprendizado ϕ
 Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
 Número de Iterações Nr
 Um conjunto de teste com nt exemplos $\mathbf{T} = \{(\mathbf{x}_j, ?), j = 1, \dots, nt\}$

Saída: Previsões para o conjunto de teste

```

1 /* Fase de Aprendizado */
2 para cada  $l = 1$  to  $Nr$  faça
3    $\mathbf{D}' \leftarrow$  amostra com reposição de  $\mathbf{D}$ 
4    $\hat{f}_l \leftarrow \phi(\mathbf{D}')$ 
5 fim
6 /* Fase de Classificação */
7 para cada  $j = 1$  to  $nt$  faça
8    $\hat{y}_j = \arg \max_{y \in Y} \sum_{l=1}^{Nr} \hat{f}_l(\mathbf{x}_j \in \mathbf{T})$ 
9 fim
10 Retorna: Vetor de previsões  $\hat{\mathbf{y}}$ 
```

C4.5 seleciona um valor no conjunto de valores que aparecem no conjunto de treinamento. Novamente, uma pequena mudança no conjunto de treinamento pode levar a um ponto de corte diferente. Agregar os classificadores por votação leva a uma superfície de decisão mais complexa. A Figura 8.3 ilustra essa última situação.

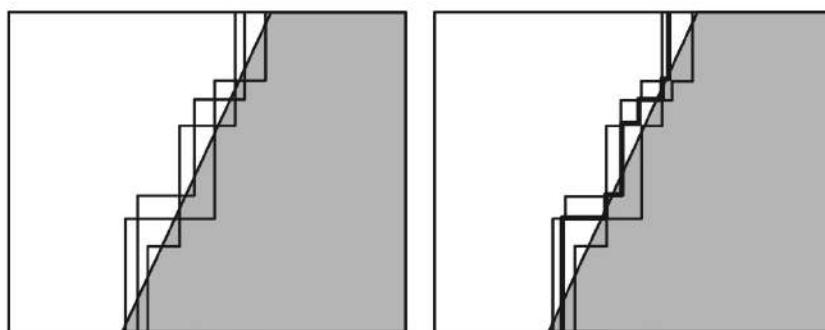


Figura 8.3 A figura da esquerda representa as superfícies de decisão de três classificadores num problema de duas classes. A figura da direita apresenta, em negrito, a superfície de decisão obtida por votação uniforme dos três classificadores.

Algumas questões são importantes de serem consideradas quando se aplica *bagging*. A primeira delas é: Quantas amostras são suficientes? A intuição de Breiman (1996a) é: “*mais replicações são necessárias com um aumento no número de classes*”. Breiman também nota que o “*bagging é quase um procedimento de sonho para a computação paralela*”, assumindo que os tempos de execução para um grande número de amostras não é tão relevante. Em um experimento de simulação, variando o número de amostras, verificou-se que um número de exemplos moderado, 10, é geralmente suficiente.

Outra questão a ser considerada é: Todos os modelos gerados por um *bagging* serão úteis para predição? Poderemos eliminar alguns desses modelos? Uma proposta comum

para esses problemas consiste em avaliar cada modelo em exemplos *out-of-bag*. Em amostragem com reposição, cerca de 1/3 dos exemplos não será usado para treinamento. Esses exemplos são chamados de exemplos *out-of-bag*. Eles podem ser usados para estimar o desempenho de um modelo gerado. Se o desempenho está abaixo de algum limiar, o modelo é deixado fora do conjunto e não é usado para fazer previsões no conjunto de teste. Um trabalho interessante nessa linha foi apresentado por Martínez-Muñoz e Suárez (2006). Os autores apresentam um método para ordenar os classificadores gerados no *bagging* e mostram que um subconjunto deles leva a um aumento de desempenho.

Concluindo, o *bagging* é um modo simples e fácil para melhorar qualquer método existente. Tudo que é necessário é primeiramente adicionar um ciclo para selecionar a amostra de treinamento e enviá-la para o algoritmo de aprendizado, e um procedimento posterior faz a soma dos votos. Em comparação com as árvores, o que se perde é a estrutura simples e interpretável; o que se ganha é um aumento de precisão.

Outro método amostral que pode ser usado para construir conjuntos de treinamento consiste em desconsiderar conjuntos disjuntos dos dados de treinamento (Dietterich, 1997; Heath et al., 1996). Por exemplo, o conjunto de treinamento pode ser dividido aleatoriamente em 10 subconjuntos disjuntos. Então, conjuntos de treinamento sobrepostos podem ser construídos descartando um conjunto diferente desses 10 subconjuntos. Esse procedimento é empregado para construir conjuntos de treinamento para 10-validação cruzada. Os conjuntos construídos desse modo são usualmente chamados de *comitês de validação cruzada*.

Classificadores Fracos e Fortes: *Boosting*

Nos anos 1980, Michael Kearns colocou uma pergunta para a comunidade científica:

Poderá um conjunto de aprendedores fracos gerar um aprendedor forte?

Um classificador *fraco* é definido como um classificador cuja capacidade de generalização é pouco melhor que a escolha aleatória. Por outro lado, um classificador *forte* pode aproximar qualquer distribuição com um erro arbitrariamente pequeno. Formalmente, a pergunta é assim enunciada:

Sendo dados uma margem de erro ϵ , e um nível de confiança $1 - \delta$, é possível construir um classificador que, com probabilidade $1 - \delta$, gera uma hipótese com erro ϵ para qualquer distribuição de exemplos gerados para um problema?

Schapire (1990) responde à pergunta propondo um método geral (*boosting*) para converter um *classificador fraco* em um que alcança uma precisão arbitrariamente alta. Um algoritmo de aprendizado *fraco* tem um desempenho ligeiramente melhor que a escolha aleatória. O trabalho de Schapire mostra como amplificar esses classificadores fracos de forma a obter uma precisão arbitrariamente alta. O algoritmo originalmente desenvolvido foi baseado no modelo teórico de aprendizado PAC (*Probably Approximately Correct*) (Mitchell, 1997).

A ideia principal por trás do algoritmo de *boosting* consiste em associar um peso para cada exemplo no conjunto de treinamento que reflete sua importância. O ajuste de pesos de maneira diferente faz com que o classificador foque em exemplos diferentes levando a diferentes classificadores. *Boosting* é um algoritmo iterativo. Em cada iteração é gerado um novo classificador. Esse classificador é treinado com a distribuição dos exemplos dado

pelos pesos associados. Os pesos são ajustados de acordo com o desempenho do conjunto de classificadores aprendidos até essa iteração. O peso dos exemplos classificados incorretamente aumenta, enquanto o peso dos exemplos corretamente classificados diminui. O classificador final agrupa os classificadores aprendidos em cada iteração pela votação pesada. O peso de cada classificador é uma função da sua precisão.

A pesquisa em técnicas de *boosting* é bastante ativa. Freund e Schapire (1996) apresentaram o algoritmo *AdaBoost* (de *Adaptive Boosting*), mostrado no Algoritmo 8.3. Como o *bagging*, esse algoritmo gera um conjunto de classificadores que participam na classificação de exemplos de teste por votação ponderada. *Boosting* gera classificadores sequencialmente. Em cada iteração, o algoritmo muda o peso dos exemplos do treinamento levando em consideração o erro do conjunto de classificadores construídos previamente.

Algoritmo 8.3 O algoritmo *ADABOOST*

Entrada: Um algoritmo de aprendizado ϕ
 Um conjunto de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
 Número de Iterações Nr
 Um conjunto de teste com nt exemplos $\mathbf{T} = \{(\mathbf{x}_j, ?), j = 1, \dots, nt\}$

Saída: Previsões para o conjunto de teste

```

1 /* Fase de Treinamento */ ;
2 para cada exemplo  $i \in \mathbf{D}$  faça
3    $w(i) \leftarrow 1/n;$ 
4 fim
5 para cada  $l = 1$  to  $Nr$  faça
6   para cada exemplo  $i \in \mathbf{D}$  faça
7      $p_l(i) \leftarrow w_l(i) / \sum_i w_l(i)$  ;
8   fim
9   /* Chamada ao Algoritmo de Aprendizado */ ;
10   $\hat{f}_l \leftarrow \phi(p_l)$  ;
11  /* Calcular o Erro */ ;
12   $e_l = \sum_i p_l(i)[\hat{f}_l(\mathbf{x}_i) \neq y_i]$  ;
13   $\beta_l \leftarrow e_l / (1 - e_l)$  ;
14  para cada exemplo  $i \in D$  faça
15     $w_{l+1}(i) := w_l(i) \beta_l^{1 - [\hat{f}_l(\mathbf{x}_i) \neq y_i]}$  ;
16  fim
17 fim
18 /* Fase de Teste */ ;
19 para cada  $j = 1$  to  $nt$  faça
20   $\hat{y}_j = \arg \max_{y \in Y} \sum_{l=1}^{Nr} (\log \frac{1}{\beta_l}) [\hat{f}_l(\mathbf{x}_j \in \mathbf{T}) = y]$  ;
21 fim
22 Retorna: Vetor de previsões  $\hat{\mathbf{y}}$ ;
```

Exemplo Ilustrativo. Os exemplos seguintes ilustram o processo de *boosting*. O *classificador fraco* desenha superfícies de decisão que consistem em um único hiperplano per-

pendicular a um dos eixos no espaço de entrada. A Figura 8.4 ilustra a primeira iteração. A distribuição dos pesos é uniforme, isto é, todos os exemplos têm o mesmo peso. Na figura, o peso dos exemplos é ilustrado pelo tamanho do círculo ao redor dos objetos. O classificador fraco encontra o hiperplano que minimiza o erro nessa distribuição.

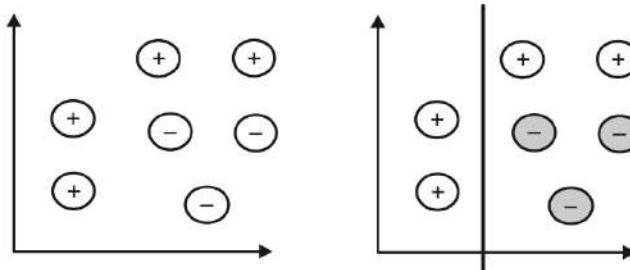


Figura 8.4 *Boosting: primeira iteração. A figura da esquerda mostra a distribuição uniforme original dos exemplos. A figura da direita mostra a superfície de decisão para essa distribuição.*

Na segunda iteração (Figura 8.5), a distribuição dos pesos muda de acordo com o erro dos modelos gerados na primeira iteração. O peso dos exemplos classificados incorretamente é aumentado, enquanto o peso dos exemplos classificados corretamente diminui. O novo conjunto de pesos define outra distribuição dos exemplos. O classificador fraco gera um modelo minimizando o erro da distribuição atual.

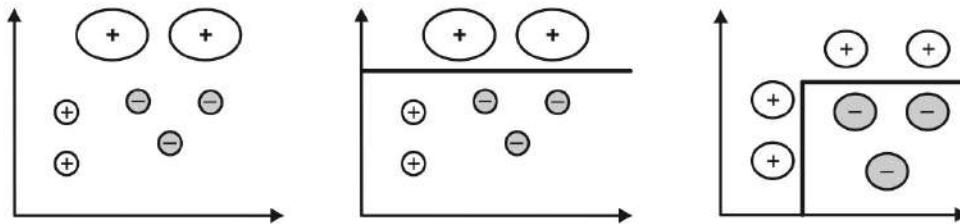


Figura 8.5 *Boosting: segunda iteração. O peso dos exemplos classificados incorretamente na primeira iteração aumentou. A figura do centro mostra a superfície para a nova distribuição dos exemplos. O modelo final, obtido conjugando os dois modelos anteriores, é apresentado na figura da direita.*

Discussão. O bom desempenho verificado com o algoritmo de *boosting* pode ser compreendido por duas observações. A primeira observação, que se ajusta a muitos problemas do mundo real, é que os exemplos observados tendem a ter níveis diversos de dificuldade de classificação. Por exemplo, os exemplos perto da superfície de decisão são mais difíceis de classificar do que os exemplos mais afastados. Para tais problemas, o algoritmo *boosting* tende a gerar distribuições que se concentram em exemplos difíceis de classificar. Por outro lado, é requerido que o algoritmo de aprendizado seja sensível à distribuição dos exemplos de treinamento, de forma a gerar hipóteses significativamente diferentes quando a distribuição do conjunto de treinamento altera. Essa propriedade está relacionada à segunda razão para a melhora verificada com o *boosting* – redução na variância. Intuitivamente, considerando a maioria ponderada sobre muitas hipóteses, tem o efeito de reduzir a variabilidade aleatória das hipóteses individuais.

8.2.2 Métodos Baseados na Amostragem do Conjunto de Atributos

A utilização de *bagging* ou *boosting* com algoritmos do tipo *naive Bayes* ou *k*-vizinhos mais próximos não é unânime. A utilização de *bagging* não é efetiva devido à instabilidade dos modelos de decisão quando se perturba o conjunto de exemplos de treinamento. Zheng (1998) aplicou *boosting* com um classificador *naive Bayes* e declara:

"Implementamos um algoritmo de boosting com um classificador naive Bayes usando um método similar àquele para uma árvore de decisão. Apesar de o algoritmo alcançar maior precisão que o classificador naive Bayes em alguns domínios, a melhoria na precisão sobre o classificador naive Bayes em um grande conjunto de domínios naturais é muito marginal. A razão deve ser que implicitamente boosting requer instabilidade do sistema de aprendizado que usou o boosting."

Melhoria efetiva de um comitê de classificadores *naive Bayes* foi obtida usando diferentes subconjuntos de atributos. Um classificador *naive Bayes* não é estável no sentido de que uma mudança no conjunto de atributos pode levar a muitos classificadores diferentes. Além disso, devido à suposição de independência do atributo, um classificador *naive Bayes* construído com um subconjunto de atributos pode apresentar melhor desempenho que um classificador *naive Bayes* criado usando todos os atributos. Essa técnica foi também usada por Skalak (1997) e Bay (1998) para comitês dos classificadores *k*-vizinhos mais próximos.

8.2.3 Métodos Baseados na Injeção de Aleatoriedade

Alguns algoritmos de aprendizado usam parâmetros inicializados aleatoriamente. Essa característica pode ser explorada no sentido de gerar diferentes modelos pela injeção de aleatoriedade nas entradas ou parâmetros do algoritmo de aprendizado.

Em redes neurais, por exemplo, um método comum para gerar diferentes redes usando o mesmo algoritmo e o mesmo conjunto de dados consiste em inicializar os pesos da rede com diferentes valores.

Breiman (2001) introduziu o algoritmo das *florestas aleatórias* (*random forests*). O modelo gera várias árvores de decisão cujas previsões são combinadas por votação uniforme. O algoritmo usa a amostragem de exemplos com reposição do *bagging* combinada à seleção aleatória de atributos. Considere um conjunto de treinamento \mathbf{D} com n exemplos e d atributos. O algoritmo das florestas aleatórias requer dois parâmetros: L é o número de árvores da floresta, e i ($i \ll d$) é o número de atributos a considerar para testes de decisão em cada nó da árvore de decisão. As florestas aleatórias geram L árvores de decisão. Cada árvore de decisão é induzida a partir de uma amostra com reposição do conjunto de treinamento. A amostra tem exatamente n exemplos, alguns dos quais repetidos. Os exemplos do conjunto de treinamento original \mathbf{D} que não estão na amostra vão ser utilizados para estimar o desempenho da árvore. O algoritmo para construir a árvore é em tudo semelhante ao algoritmo apresentado na Seção 6.1, com uma única diferença. Em cada nó da árvore, para escolher o atributo de teste, apenas são considerados i atributos escolhidos de forma aleatória. Nenhuma das árvores é podada.

Os processos de amostragem quer dos exemplos quer dos atributos não provocar um aumento da variabilidade das árvores induzidas. Resultados experimentais revelaram que esse algoritmo é dos mais competitivos.

8.2.4 Métodos Baseados na Perturbação dos Exemplos de Teste

Nesta seção abordamos técnicas que utilizam um único modelo e adiam para o estágio de predição a geração de múltiplas previsões pela perturbação do vetor de atributos correspondente ao caso de teste.

Um dos métodos mais ilustrativo dessa técnica foi apresentado por Geurts (2000, 2001). O autor apresenta o DPC (do inglês, *Dual Perturba e Combina*), que pode ser aplicado no topo de qualquer modelo produzido por qualquer algoritmo de aprendizado. Um único modelo é gerado a partir de um conjunto de treinamento. Na fase de predição, cada exemplo de teste sofre perturbações várias vezes. Para inserir perturbações no exemplo de teste, um ruído branco é adicionado ao valor do atributo. O modelo preditivo faz uma predição para cada versão do exemplo de teste com perturbações. A predição final é obtida pela agregação de previsões diferentes. Geurts (2000) apresenta evidências experimentais de que esse método é eficiente na redução da variação. O algoritmo principal é mostrado no Algoritmo 8.4.

Algoritmo 8.4 Algoritmo Dual Perturba e Combina (DPC)

Entrada:

Um conjunto de exemplos de teste $\mathbf{T} = \{(\mathbf{x}_i, ?), i = 1, \dots, n\}$

Um número de Iterações Nr

Um modelo de classificação (regressão) \hat{f}

Saída: Vetor de previsões para o conjunto de teste;

- 1 **para cada** $\mathbf{x}_i \in \mathbf{T}$ **faça**
 - 2 **para cada** $l = 1$ **to** Nr **faça**
 - 3 seja $\mathbf{x}_{i\epsilon^l}$ uma variante perturbada de \mathbf{x}_i , em que ϵ^l é obtido por uma distribuição gaussiana $N(0, \lambda_l \cdot \sigma_l)$;
 - 4 $p_l \leftarrow \hat{f}(\mathbf{x}_{i\epsilon^l})$;
 - 5 **fim**
 - 6 Calcula a previsão agregada, dada por: $\hat{y}_i = aggr_{l=1}^{Nr} p_l$, em que $aggr$ é um operador de agregação (média, mediana etc.);
 - 7 **fim**
 - 8 **Retorna:** Vetor de previsões $\hat{\mathbf{y}}$;
-

8.3 Combinando Classificadores Heterogêneos

Uma maneira de garantir a diversidade dos classificadores de base é com o uso de diferentes algoritmos para a produção dos classificadores. Nesse caso, temos um conjunto heterogêneo de classificadores para combinar.

8.3.1 Generalização em Pilha

Wolpert (1992) propôs o método *Generalização em Pilha*,³ que possui uma arquitetura de aprendizado em camadas. Os classificadores no *Nível₀* recebem como entrada os dados originais, e cada classificador produz uma predição. Camadas sucessivas recebem como entrada as predições das camadas imediatamente precedentes, e a saída é passada para a próxima camada. Um único classificador no nível mais alto produz a predição final.

Generalização em pilha é um processo para minimizar o erro de generalização usando classificadores nas camadas mais altas para aprender o tipo de erro cometido pelo classificador imediatamente abaixo. Nessa perspectiva, ela pode ser vista como uma extensão para métodos de seleção de modelo, nomeados *validação cruzada*, que usa uma estratégia de “o vencedor leva tudo”. Um único classificador com erro de validação cruzada mais baixo é selecionado. A ideia por trás da generalização em pilha é que pode haver um modo mais inteligente para usar o conjunto de classificadores. A regra dos classificadores dos níveis mais altos é aprender como os classificadores anteriores cometem erros, em qual classe eles concordam ou discordam, e usar o seu conhecimento para fazer predições.

A maioria dos trabalhos usando a arquitetura de pilha, por exemplo Wolpert (1992), Ting e Witten (1997), Skalak (1997) e Breiman (1996c), concentra-se na arquitetura duas camadas, que é ilustrada na Tabela 8.2 e na Figura 8.6. A Tabela 8.2 mostra o conjunto de dados original e o conjunto de dados de *Nível₁*. Nesse último caso, cada coluna, designada por P_{ik} , representa a probabilidade de o exemplo ser da classe k dada pelo classificador i . Nesse caso, há duas fases diferentes: a fase de treinamento ou aprendizado e a fase de aplicação. A fase de aprendizado consiste nos seguintes passos:

1. Treinar cada um dos classificadores *Nível₀* usando validação cruzada com o método deixar-um-de-fora da seguinte forma: para cada exemplo no conjunto de treinamento deixe um de fora e treine com os demais exemplos. Depois do treinamento, classifique o exemplo excluído. Crie um vetor a partir das predições de todos os classificadores *Nível₀* e a classe atual daquele exemplo.
2. Treinar o classificador *Nível₁*, usando como conjunto de treinamento a coleção de vetores gerados nos passos anteriores. O número de exemplos nos dados *Nível₁* é igual ao número de exemplos no conjunto de treinamento original.
3. No passo 1, classificadores são gerados usando um método deixar um de fora. Para explorar completamente o conjunto de treinamento, todos os classificadores *Nível₀* são retreinados usando o conjunto de treinamento inteiro. Os modelos gerados são usados para classificar os exemplos no conjunto de teste.

Na fase de aplicação, quando um novo exemplo é apresentado, este é classificado por todos os classificadores *Nível₀*. O vetor de predições é então classificado pelo classificador *Nível₁*, que produz como saída a predição final para o exemplo (ver Figura 8.6).

O enquadramento geral não é restrito ao modelo básico descrito. Por exemplo, Breiman (1996c) verificou que em alguns problemas de regressão foram obtidos melhores resultados usando *10-fold cross-validation* em vez de *leave-one-out*. Ting e Witten (1997) empiricamente observam que, usando distribuições de probabilidade de classe como atributos no

³Stacked Generalization ou Stacking.

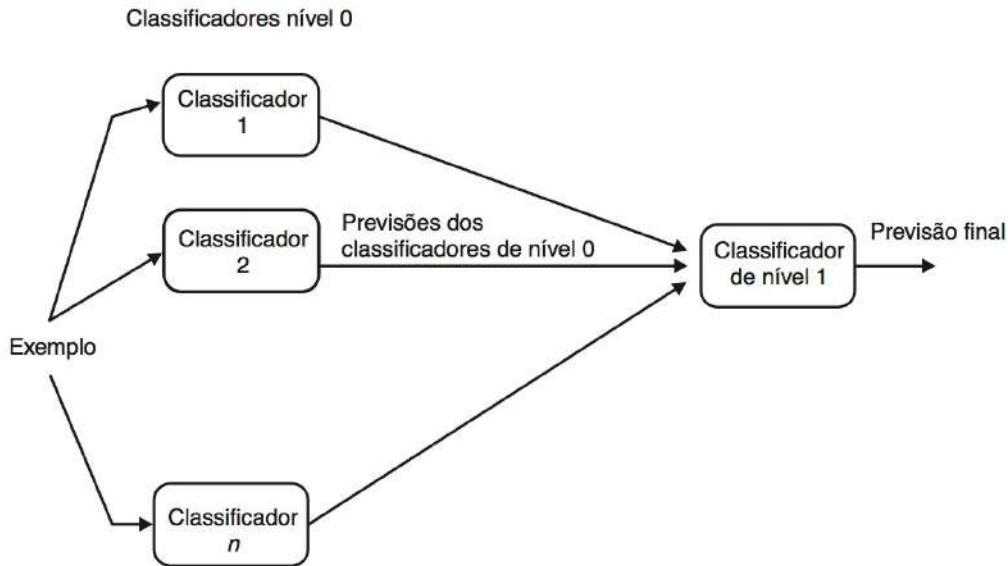


Figura 8.6 Arquitetura generalização em pilha.

Tabela 8.2 Exemplo ilustrativo de generalização em pilha: A tabela mostra o conjunto de dados original e o conjunto de dados de $Nível_1$

V1	V2	V3	V4	V5	Classe	P _{1,1}	P _{1,2}	P _{2,1}	P _{2,2}	P _{3,1}	P _{3,2}	Classe
t	a	c	t	a	Membro	0,51	0,49	0,13	0,87	0,12	0,88	Membro
t	g	c	t	a	Membro	0,19	0,81	0,07	0,93	0,81	0,19	Membro
g	t	a	c	t	Não Membro	0,68	0,32	0,55	0,45	0,69	0,31	Não Membro
a	a	t	t	g	Membro	0,74	0,26	0,66	0,34	0,94	0,06	Membro
t	c	g	a	t	Não Membro	0,62	0,38	0,01	0,99	0,78	0,22	Não Membro
a	g	g	g	g	Membro	0,65	0,35	0,90	0,10	0,55	0,45	Membro

Conjunto de dados original

Conjunto de dados de $Nível_1$

$Nível_1$ e um discriminante linear como classificador $Nível_1$, são obtidos melhores resultados. Em todos os casos, a generalização em pilha é uma técnica sofisticada para reduzir o erro devido à redução do viés (Breiman, 1996c; Skalak, 1997).

8.3.2 Generalização em Cascata

Generalização em cascata (Gama e Brazdil, 2000) é uma composição sequencial de classificadores que em cada nível de generalização aplica um operador construtivo. O operador construtivo constrói novos atributos. Dados um conjunto de treinamento D , um conjunto de teste T e dois algoritmos \mathfrak{S}_1 e \mathfrak{S}_2 , a generalização em cascata procede como se segue: o algoritmo \mathfrak{S}_1 gera um classificador, \hat{f}_1 , usando o conjunto de treinamento D . O modelo gerado, \hat{f}_1 , classifica todos os exemplos de treinamento e teste. Assume-se que o resultado de aplicar o modelo \hat{f}_1 a um exemplo é uma distribuição de probabilidade da classe. Ou seja, um vetor, \mathbf{c} , com a dimensão igual ao número de classes, em que cada elemento do vetor é a probabilidade de esse exemplo pertencer a uma das classes. O operador construtivo concatena cada exemplo \mathbf{x} com o vetor \mathbf{c} . O resultado da aplicação do operador construtivo é um novo conjunto de dados, com o mesmo número de exemplos do conjunto original, mas em que cada exemplo é acrescido de novos atributos, um novo

atributo para cada classe. Cada novo atributo é a probabilidade de o exemplo pertencer a uma das classes dado pelo modelo \hat{f}_1 . A aplicação do operador construtivo gera os conjuntos de dados ditos de $Nível_1$. O classificador \mathfrak{S}_2 aprende com os dados de treinamento $Nível_1$ e classifica os dados de teste $Nível_1$.

Estes passos representam a sequência básica da generalização em cascata. A composição dos dois modelos é representada formalmente pela expressão: $\mathfrak{S}_2 \nabla \mathfrak{S}_1$. Essa é a fórmula mais simples da generalização em cascata. Algumas possíveis extensões incluem a composição de nc classificadores e a composição paralela de classificadores. Uma composição de nc classificadores é representada por:

$$\mathfrak{S}_{nc} \nabla \mathfrak{S}_{nc-1} \nabla \mathfrak{S}_{nc-2} \dots \nabla \mathfrak{S}_1$$

Nesse caso, a generalização em cascata gera $nc - 1$ níveis de dados. O modelo final é aquele dado pelo classificador \mathfrak{S}_{nc} . Esse modelo pode conter termos na forma de condições baseados nos atributos construídos pelos classificadores anteriormente construídos.

Exemplo Ilustrativo Nesse exemplo, considera-se o conjunto de dados UCI **Monks-2** (Thrun et al., 1991). Os conjuntos de dados **Monks** descrevem o domínio de um robô artificial e são muito conhecidos na comunidade de Aprendizado de Máquina. Os robôs são descritos por seis atributos e classificados em uma das duas classes. O problema **Monks-2** foi escolhido aqui porque se sabe que essa é uma tarefa difícil para sistemas que aprendem usando árvores de decisão. A regra de decisão para esse problema é: *O robô é amigo, se exatamente 2 dos seis atributos assumem o 1º valor do domínio.* A regra de decisão combina os diferentes atributos de um modo que se torna complicado descrever na Forma Normal Disjuntiva usando somente os atributos originais.

Alguns exemplos dos dados de treinamento são apresentados na Tabela 8.3.

Tabela 8.3 Dois exemplos do conjunto de dados de $Nível_0$ no problema **Monks-2**

Cabeça	Corpo	Sorriso	Objeto	Cor	Gravata	Classe
redonda	redondo	sim	espada	vermelho	sim	inimigo
redonda	redondo	não	balão	azul	não	amigo

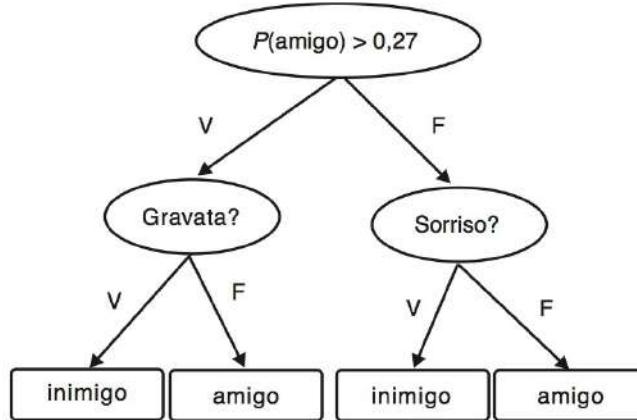
Usando *10-fold cross-validation*, a taxa de erro do C4.5 é 32,9%, e para o *naive Bayes* é 34,2%. O modelo composto C4.5 depois de *naive Bayes*, $C4.5 \nabla \text{naive Bayes}$ opera como se segue. O *naive Bayes* aprende um classificador usando o conjunto de treinamento de $Nível_0$. Esse classificador classifica os exemplos de treinamento e de teste de $Nível_0$, ou seja, para cada exemplo retorna a probabilidade de o exemplo pertencer a cada classe. Como existem duas classes, tem-se duas probabilidades. Cada exemplo é estendido com dois novos atributos dando origem aos conjuntos de dados de $Nível_1$. O exemplo mostrado anteriormente tem a forma ilustrada na Tabela 8.4. Os novos atributos $P(\text{amigo})$ e $P(\text{inimigo})$ representam a probabilidade de que o exemplo pertença às classes *amigo* e *inimigo*, respectivamente.

C4.5 é treinado nos dados de treinamento do $Nível_1$ e classifica os dados de teste do $Nível_1$. A composição C4.5 ∇ *naive Bayes* obtém uma taxa de erro de 8,9%, que é substancialmente menor que a taxa de erro tanto de C4.5 quanto de *naive Bayes*. Nenhum dos algoritmos de forma isolada pode capturar o conceito subjacente ao problema. Nesse

Tabela 8.4 Dois exemplos do conjunto de dados de Nível₁

Cabeça	Corpo	Sorriso	Objeto	Cor	Gravata	P(amigo)	P(inimigo)	Classe
redonda	redondo	sim	espada	vermelho	sim	0,135	0,864	inimigo
redonda	redondo	não	balão	azul	não	0,303	0,696	amigo

caso, a generalização em cascata alcançou um desempenho notável. A Figura 8.7 apresenta uma das árvores geradas pelo C4.5 ∇ *naive Bayes*.

**Figura 8.7** A árvore gerada pelo C4.5 ∇ *naive Bayes*.

A árvore contém uma mistura de alguns dos atributos originais (*Sorriso*, *Gravata*) com alguns dos novos atributos construídos pelo *naive Bayes* (*P(amigo)*, *P(inimigo)*). Na raiz da árvore aparece o atributo *P(amigo)*. Esse atributo representa a probabilidade de um exemplo pertencer à classe *amigo*, calculada pelo *naive Bayes*. A árvore de decisão gerada pelo C4.5 usa os atributos construídos pelo *naive Bayes*, mas redefinindo diferentes limiares. Sendo um problema de duas classes, os modelos bayesianos usam *P(amigo)* com limiar 0,5, enquanto a árvore de decisão ajusta o valor do limiar para 0,27. Os nós de decisão com testes nos atributos construídos são um tipo de função dada pela estratégia bayesiana. Por exemplo, o atributo *P(amigo)* pode ser visto como uma função que calcula $p(Classe = amigo | \mathbf{x})$ usando o teorema de Bayes. Em alguns ramos, a árvore de decisão realiza mais de um teste de probabilidade de classes. De certa forma, essa árvore de decisão combina duas linguagens de representação: a representação do *naive Bayes* com a linguagem da árvore de decisão. O passo construtivo executado pela generalização em cascata insere novos atributos que incorporam novo conhecimento fornecido pelo *naive Bayes*. Esse fato permite o aumento significativo de desempenho verificado com a árvore de decisão.

Discussão. Generalização em cascata pertence à família de algoritmos de generalização em pilha. Wolpert (1992) define generalização em pilha como um contexto geral para combinar classificadores. Ele envolve obter as previsões de vários classificadores e usar essas previsões como a base para o próximo estágio de classificação.

Generalização em cascata pode ser considerada um caso especial de generalização em pilha principalmente devido à estrutura de aprendizado em camadas. Alguns aspectos que fazem a generalização em cascata particular são:

- Os novos atributos são contínuos. Ele obtém a forma da distribuição de probabilidade da classe. Combinar classificadores através de classes categóricas perde a força do classificador na sua predição. O uso da distribuição das classes de probabilidades nos permite explorar essa informação.
- Todos os classificadores têm acesso aos atributos originais. Qualquer novo atributo construído por um classificador mais baixo é considerado exatamente do mesmo modo que qualquer um dos atributos originais.
- Generalização em cascata não usa validação cruzada interna. Esse aspecto afeta a eficiência computacional da generalização em cascata.

Muitas dessas ideias foram discutidas na literatura. Ting e Witten (1997) usaram a distribuição de probabilidade da classe como atributos *Nível₁*, mas não usaram os atributos originais. Chan e Stolfo (1995a) usaram os atributos originais e as predições das classes em um esquema denotado como *combinador-atributo-classe*. Explorar todos esses aspectos é o que faz a generalização em cascata bem-sucedida. Entretanto, essa combinação particular implica algumas diferenças *conceituais*.

- Enquanto a generalização em pilha tem uma natureza paralela, a generalização em cascata é sequencial. O efeito é que classificadores intermediários têm acesso aos atributos originais mais as predições dos classificadores de baixo nível.
- O objetivo final da generalização em pilha é combinar predições. O objetivo da generalização em cascata é obter um modelo que pode ser usar termos na linguagem de representação dos classificadores de mais baixo nível. Na generalização em cascata, os classificadores de mais baixo nível adiam a decisão final para os classificadores de alto nível.
- A utilização da generalização em cascata usa sequências de poucos classificadores. O perfil para os classificadores no início da sequência é baixa variância. Os classificadores no fim da sequência deverão ter baixo enviesamento.

8.3.3 Meta-Aprendizado

Chan e Stolfo (1995a) apresentam dois esquemas para combinação de classificadores: *árbitro* e *combinador*. Ambos os esquemas são baseados em meta-aprendizado, em que um metaclassificador é gerado a partir de metadados, com base nas predições dos classificadores de base. Um árbitro é também um classificador, e é usado para arbitrar entre predições geradas por diferentes classificadores de base. O conjunto de treinamento para o árbitro é obtido a partir de todos os dados disponíveis usando regras de seleção. Um exemplo de uma regra de seleção é “*Selecione os exemplos cuja classificação não pode ser predita consistentemente usando classificadores de base.*” Esse árbitro, junto com uma regra de arbitragem, determina a classificação final com base nas predições de base. Um exemplo de uma regra de arbitragem é: “*Use a predição do árbitro quando o classificador de base não obtém a maioria*”. Mais tarde (Chan e Stolfo, 1995b, 1997), esse *framework* foi estendido usando árbitro/combinadores em um modo hierárquico, gerando árvores binárias de árbitros/combinadores. Uma árvore de árbitros/combinadores é uma estrutura

hierárquica composta de árbitros/combinadores que são calculados em uma árvore binária de um modo de baixo-para-cima (*bottom-up*). Um árbitro/combinador é inicialmente aprendido a partir da saída de um par de classificadores base, e, recursivamente, um árbitro/combinador é aprendido a partir da saída de dois árbitros. Para nc classificadores, há $\log_2(nc)$ níveis gerados.

8.3.4 Sistemas Híbridos

Resultados de comparações empíricas de algoritmos de aprendizado existentes (Michie et al., 1994) ilustram que cada algoritmo tem certa *superioridade seletiva*. Ele é melhor para algumas tarefas, mas não para todas. Essa é a motivação por trás do desenvolvimento de algoritmos que podem se adequar aos dados com representações *heterogêneas*. Isto é, diferentes regiões do espaço de entrada são aproximadas usando diferentes tipos de modelos.

Domingos (1998) propõe CMM, (do inglês *Combined Multiple Models*), um metaclassificador que procura reter os maiores ganhos de precisão de propostas de modelos múltiplos enquanto ainda produz um único modelo comprehensível. CMM gera um novo conjunto de treinamento, composto de um grande número de exemplos gerados e classificados de acordo com um conjunto, mais os exemplos originais. CMM foi usado com regras C4.5 como um classificador de base, e com *bagging* como a metodologia de modelos múltiplos. Em 26 conjuntos de dados de referência, CMM retém em média 60% dos ganhos de precisão obtidos pelo *bagging* relativo a uma única execução das regras C4.5, enquanto produz um conjunto de regras cuja complexidade é tipicamente um pequeno múltiplo da complexidade das regras C4.5.

Brodley (1995, 1993) apresenta uma *Seleção do Modelo de Classe - sistema MCS*, um algoritmo híbrido que combina, em uma única árvore, nós que são testes invariantes, testes multivariantes gerados por *máquinas lineares* e classificadores baseados em exemplo. Cada nó MCS usa um conjunto de regras *Se-Então* para executar uma pesquisa heurística *best-first* para a melhor hipótese para uma partição dada do conjunto de dados. O conjunto de regras incorpora conhecimento do especialista. Uma dessas regras é:

se o número de exemplos é inferior à capacidade do hiperplano
então encontra o melhor teste univariado
caso contrário gera uma combinação linear LT_n usando todos os atributos.

MCS usa uma estratégia de controle da pesquisa dinâmica para executar uma seleção automática do modelo. *MCS* constrói árvores que podem ser aplicadas a diferentes modelos em diferentes regiões do espaço de objetos.

Kohavi (1996) apresenta um sistema híbrido que gera uma árvore de decisão invariante usando classificadores *naive Bayes* nas folhas da árvore. Essa proposta tenta utilizar as vantagens tanto de árvores de decisão (isto é, segmentação) quanto de *naive Bayes* (acumulação de evidência de múltiplos atributos). O autor alega que o modelo retém a interpretabilidade do *naive Bayes* e árvores de decisão, enquanto resulta em classificadores que frequentemente superam ambos os constituintes, especialmente em grandes bancos de dados.

Domingos (1997b, 1996) apresenta um sistema *RISE* (*Rule Induction from a Set of Exemplars*) que unifica aprendizado baseado em regras e aprendizado baseado em exem-

plos. Exemplos são tratados como regras de especificidade máxima. As regras são aprendidas gradualmente pela generalização de exemplos até que não é obtida nenhuma melhoria na precisão. Os exemplos de teste são classificados usando a regra, que pode ser um exemplo, mais próximo da base de conhecimentos.

8.4 Considerações Finais

Um conjunto de preditores é em si um algoritmo de aprendizado supervisionado. O modelo múltiplo representa uma hipótese; no entanto, essa hipótese não está necessariamente contida no espaço de hipóteses dos modelos base a partir dos quais ela é construída. Assim, os conjuntos podem ter maior flexibilidade nas funções que representam. Muitos métodos procuram promover a diversidade entre os modelos que combinam, por exemplo, perturbando a distribuição dos exemplos no conjunto de treinamento. *Bagging* e *boosting* Bauer e Kohavi (1999) são das formas mais eficientes para melhorar a precisão de classificadores, tais como árvores de decisão e redes neurais.

Quinlan (1996) e Bauer e Kohavi (1999) concordam que *boosting* apresenta maior benefício, mas produz degradações severas em alguns conjuntos de dados. Ali e Pazzani (1996) mostram que o número de exemplos treinados necessários para o *boosting* aumenta em função da precisão do modelo aprendido. Bauer e Kohavi (1999) referem que o principal problema com *boosting* é a falta de robustez em dados com ruído. Isso é esperado, porque exemplos ruidosos tendem a ser classificados incorretamente, e seu peso será consequentemente aumentado. A maioria dos estudos em *boosting* e *bagging* requer um número considerável de classificadores. Por exemplo, Bauer e Kohavi (1999) usam 25 classificadores, Freund e Schapire (1996) e Breiman (1998) usam 100 classificadores.

Wolpert (1992) refere que uma implementação bem-sucedida da generalização em pilha para tarefas de classificação é uma “*black art*”, e as condições sobre as quais o *stacking* trabalha são ainda desconhecidas. Posteriormente, Ting e Witten (1997) mostraram que o sucesso da generalização em pilha requer o uso da distribuição das classes de saída. Nas experiências reportadas, somente o algoritmo MLR (com um discriminante linear) foi apropriado para o generalizador *Nível*₁.

Capítulo 9

Avaliação de Modelos Preditivos

Na aplicação de algoritmos de AM a problemas reais, em geral, o conhecimento que se tem do domínio sendo investigado é provido unicamente pelo conjunto de exemplos, a partir do qual a indução de um modelo preditivo/descriptivo é então realizada. Os capítulos anteriores apresentaram várias técnicas de AM que podem ser utilizadas na indução de modelos de classificação e/ou de regressão a partir de um conjunto de exemplos rotulados. De maneira geral, pode-se afirmar que não existe técnica universal, ou seja, não é possível estabelecer *a priori* que uma técnica de AM em particular se sairá melhor na resolução de qualquer tipo de problema.

Em certos casos, as próprias características das técnicas existentes e do problema que está sendo solucionado podem ser consideradas para auxiliar na escolha da técnica a ser utilizada sobre um novo conjunto de dados. Por exemplo, em domínios em que os exemplos possuem alta dimensionalidade, as SVMs são boas candidatas, enquanto o algoritmo k -NN usando a distância euclidiana pode, a princípio, não parecer uma escolha adequada. Caso seja necessário que o modelo obtido seja interpretável, técnicas simbólicas, como as árvores de decisão, podem ser preferíveis a modelos “caixa-preta” como os gerados pelas RNAs e pelas SVMs.

Mesmo com o uso dessas *heurísticas*, diversos algoritmos podem ser considerados candidatos à solução de um dado problema. Ainda que um único algoritmo seja escolhido, pode ser necessário realizar ajustes em seus parâmetros livres, o que leva à obtenção de múltiplos modelos para os mesmos dados.

Os parágrafos anteriores evidenciam uma característica particular do domínio de AM: a necessidade de experimentação. De fato, a validação de qualquer nova técnica de AM proposta geralmente envolve a realização de experimentos controlados, em que se demonstre a sua efetividade na solução de diferentes problemas, representados por seus conjuntos de dados associados. Dessa forma, é recomendável seguir procedimentos que garantam a corretude, a validade e a reproduzibilidade dos experimentos realizados e, mais importante, das conclusões obtidas a partir de seus resultados.

Essa avaliação experimental de um algoritmo de AM pode ser realizada segundo diferentes aspectos, tais como acurácia do modelo gerado, comprehensibilidade do conhecimento extraído, tempo de aprendizado, requisitos de armazenamento do modelo, entre outros. Considerando modelos preditivos, iremos concentrar a nossa discussão em medidas relacionadas ao desempenho obtido nas previsões realizadas. Muitos dos conceitos e procedimentos discutidos são facilmente generalizáveis a outros tipos de medidas de desempenho. Já a avaliação de resultados no contexto de modelos descriptivos é abordada no Capítulo 14.

Inicialmente, na Seção 9.1, é realizada uma discussão acerca das principais medidas de

erro utilizadas na avaliação de preditores em AM. Na Seção 9.2 são apresentadas técnicas para a amostragem de dados em experimentos. Na Seção 9.3 são apresentadas outras medidas de desempenho no contexto de classificação binária e as curvas ROC. Na Seção 9.4 são apresentadas discussões a respeito da comparação estatística do desempenho de diferentes modelos. Este capítulo é concluído com a apresentação da decomposição viés-variância do erro de um preditor (Seção 9.5), seguida por considerações finais (Seção 9.6).

9.1 Métricas de Erro

A avaliação de um algoritmo de AM supervisionado é normalmente realizada por meio da análise do desempenho do preditor gerado por ele na rotulação de novos objetos, não apresentados previamente em seu treinamento (Monard e Baranauskas, 2003).

9.1.1 Métricas para Classificação

Uma medida de desempenho usualmente empregada na avaliação de um classificador \hat{f} é a sua taxa de erro ou de classificações incorretas, ilustrada na Equação 9.1, em que $I(a) = 1$ se a é verdadeiro e é 0 em caso contrário. Dado um conjunto de dados contendo n objetos, sobre o qual a avaliação será realizada, essa taxa equivale à proporção de exemplos desse conjunto classificados incorretamente por \hat{f} e é obtida pela comparação da classe conhecida de \mathbf{x}_i , y_i , com a classe predita, $\hat{f}(\mathbf{x}_i)$. Esse tipo de medida equivale ao uso da função de custo 0-1 relacionando os rótulos dos objetos às previsões obtidas (Equação 7.8).

$$err(\hat{f}) = \frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{f}(\mathbf{x}_i)) \quad (9.1)$$

A taxa de erro varia entre 0 e 1, e valores próximos ao extremo 0 são melhores. O complemento dessa taxa corresponde à taxa de acerto ou acurácia do classificador:

$$ac(\hat{f}) = 1 - err(\hat{f}) \quad (9.2)$$

Nesse caso, valores próximos de 1 são considerados melhores. Outra alternativa para visualizar o desempenho de um classificador é com o uso de uma matriz de confusão. Essa matriz ilustra o número de previsões corretas e incorretas em cada classe. Para um determinado conjunto de dados, as linhas dessa matriz representam as classes verdadeiras, e as colunas, as classes preditas pelo classificador. Logo, cada elemento m_{ij} de uma matriz de confusão \mathbf{M}_c apresenta o número de exemplos da classe i classificados como pertencentes à classe j . Para k classes, \mathbf{M}_c tem então dimensão $k \times k$. A diagonal apresenta os acertos do classificador, enquanto os outros elementos correspondem aos erros cometidos nas suas previsões.

Por meio do exame dessa matriz, têm-se medidas quantitativas de quais classes o algoritmo de aprendizado tem maior dificuldade. Na Figura 9.1 é apresentado um exemplo de matriz de confusão para um problema com três classes. Segundo essa matriz, onze dos quinze exemplos da classe 1 foram corretamente classificados, um foi incorretamente classificado como pertencente à classe 2 e três foram preditos como sendo da classe 3.

		Classe predita		
		1	2	3
Classe verdadeira	1	11	1	3
	2	1	4	0
	3	2	1	6

Figura 9.1 Exemplo de matriz de confusão para um problema com três classes.

9.1.2 Métricas para Regressão

As considerações anteriores diziam respeito a problemas de classificação. No caso de problemas de regressão, o erro da hipótese \hat{f} pode ser calculado pela distância entre o valor y_i conhecido e aquele predito pelo modelo, ou seja, $\hat{f}(\mathbf{x}_i)$ (Monard e Baranauskas, 2003). As medidas de erro mais conhecidas e usadas nesse caso são o erro quadrático médio (MSE - *mean squared error*) e a distância absoluta média (MAD - *mean absolute distance*):

$$\text{MSE} (\hat{f}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(\mathbf{x}_i))^2 \quad (9.3)$$

$$\text{MAD} (\hat{f}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{f}(\mathbf{x}_i)| \quad (9.4)$$

O MSE e MAD são sempre não negativos. Para ambas as medidas, valores mais baixos correspondem a melhores modelos, ou seja, melhores aproximações dos rótulos verdadeiros dos objetos.

9.2 Amostragem

Em diversos casos, tem-se apenas um conjunto com n objetos, o qual deve ser empregado na indução do preditor e também em sua avaliação. Calcular o desempenho preditivo – em termos de taxa de acerto ou de erro, por exemplo – do modelo nos mesmos objetos empregados em seu treinamento produz estimativas otimistas, uma vez que todos os algoritmos de AM tentam melhorar de alguma forma o seu desempenho preditivo nesses objetos durante a fase indutiva. O uso do mesmo conjunto de exemplos no treinamento e avaliação do preditor é conhecido como ressubstituição (Toussaint, 1974). Em geral, o erro/acerto obtido nesse tipo de avaliação é denominado aparente.

Devem-se então utilizar métodos de amostragem alternativos para obter estimativas de desempenho preditivo mais confiáveis, definindo subconjuntos de treinamento e de teste. Os dados de treinamento são empregados na indução e no ajuste do modelo, enquanto os exemplos de teste simulam a apresentação de objetos novos ao preditor, os quais não foram vistos em sua indução. Esses subconjuntos são disjuntos para assegurar que as medidas de desempenho sejam obtidas a partir de um conjunto de exemplos diferente daquele usado no aprendizado. Alguns dos principais métodos de amostragem existentes

são ilustrados na Figura 9.2: *holdout* (9.2(a)), amostragem aleatória (9.2(b)), validação cruzada (9.2(c)) e *bootstrap* (9.2(d)).

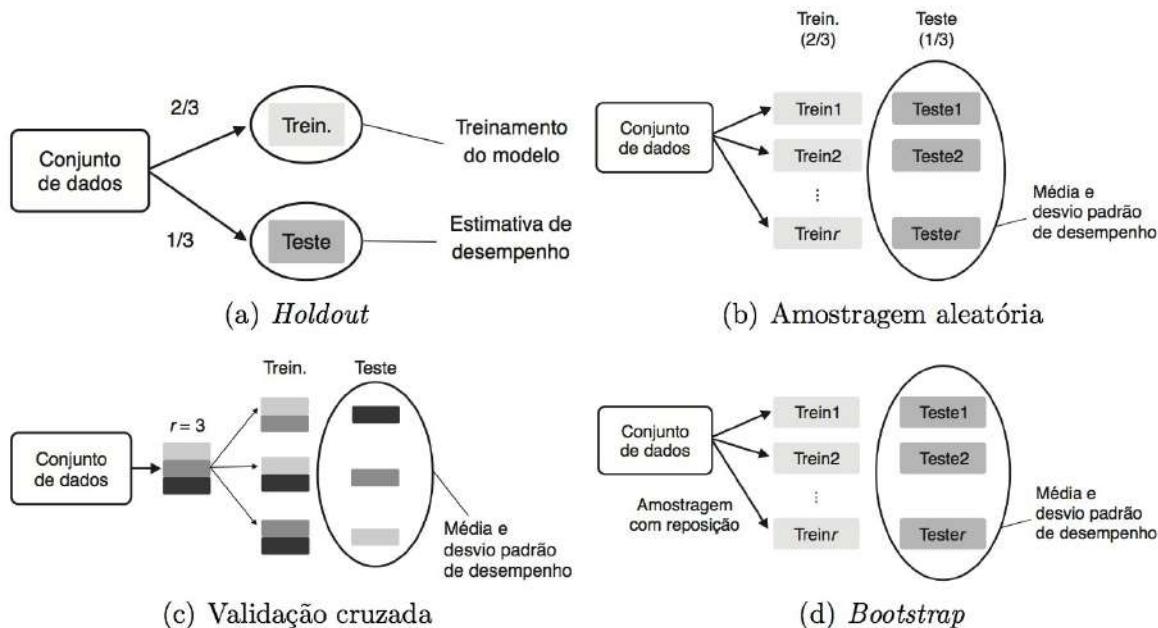


Figura 9.2 Métodos de amostragem.

No caso dos procedimentos que envolvem médias de desempenho, deve-se reportar também os valores de desvio padrão associados. Um alto desvio padrão indica uma alta variabilidade nos resultados, ou seja, uma instabilidade do modelo perante mudanças nos objetos que são todos provenientes de uma mesma distribuição. Isso pode ser um indicativo de sensibilidade aos objetos usados no treinamento. Na validação cruzada com 10 partições, por exemplo, tem-se pequenas mudanças nos objetos em cada partição, conforme será discutido na Seção 9.2.2. Dessa forma, um algoritmo com alto desvio padrão em validação cruzada apresenta sensibilidade a poucas alterações nos objetos usados em seu treinamento.

Para reportar estimativas mais precisas do desempenho esperado do algoritmo nesses casos, deve-se então apresentar algum intervalo de confiança para a média calculada, que irá envolver também o desvio padrão obtido. De maneira geral, isso é realizado pela comunidade reportando ao menos os valores de média e desvio padrão dos experimentos realizados. Contudo, pode-se também optar por calcular intervalos com maior nível de confiança, empregando para tal métodos estatísticos. Detalhes a respeito do cálculo de intervalos de confiança podem ser consultados em diversos livros da área de Estatística, entre os quais Devore (2006).

De forma geral, o desvio padrão pode até mesmo auxiliar na escolha entre dois algoritmos com desempenho médio semelhante: o de menor desvio padrão apresenta um desempenho mais estável e pode ser preferido. Contudo, em cenários como esse, o recomendável é fazer uma análise mais rigorosa, envolvendo testes de hipóteses para comparação dos desempenhos de modelos. Esse tópico é abordado na Seção 9.4.

9.2.1 Holdout e Amostragem Aleatória

No caso do *holdout*, divide-se o conjunto de dados em uma proporção de p para treinamento e $(1-p)$ para teste, como ilustrado na Figura 9.2(a). Normalmente, emprega-se $p = \frac{2}{3}$. Esse tipo de separação pode subestimar a taxa de acerto, uma vez que um preditor produzido sobre todos os objetos em geral apresentará uma taxa de acerto maior que a gerada a partir de uma parte deles (Baranauskas e Monard, 2000). Porém, segundo Michie et al. (1994), para conjuntos de dados grandes isso não representa um problema – embora a definição de quando um conjunto de dados é grande o suficiente não seja clara e dependa de vários fatores, entre eles a própria complexidade da estrutura presente nos dados, que não é conhecida de antemão.

Outra crítica usual ao *holdout* é que este não permite avaliar o quanto o desempenho de uma técnica varia quando diferentes combinações de objetos são apresentadas em seu treinamento. De fato, é possível que, em uma divisão realizada, objetos considerados “mais fáceis” tenham sido agrupados no subconjunto de teste.

Para tornar os resultados menos dependentes da partição feita, é possível fazer diversas partições aleatórias e obter uma média de desempenho em *holdout*, um método às vezes referenciado como *random subsampling* (amostragem aleatória). Na Figura 9.2(b), é apresentado um exemplo em que são gerados r diferentes subconjuntos aleatórios.

9.2.2 Validação Cruzada

No método de validação cruzada *r-fold cross-validation*, o conjunto de exemplos é dividido em r subconjuntos de tamanho aproximadamente igual. Os objetos de $r - 1$ partições são utilizados no treinamento de um preditor, o qual é então testado na partição restante. Esse processo é repetido r vezes, utilizando em cada ciclo uma partição diferente para teste. O desempenho final do preditor é dado pela média dos desempenhos observados sobre cada subconjunto de teste. Esse processo é ilustrado na Figura 9.2(c), empregando $r = 3$.

Uma variação desse método para problemas de classificação é o *r-fold cross-validation* estratificado, que mantém em cada partição a proporção de exemplos de cada classe semelhante à proporção contida no conjunto de dados total. Se, por exemplo, o conjunto de dados original tem 20% dos objetos na classe c_1 e 80% na classe c_2 , cada partição também procura manter essa proporção, apresentando 20% de seus exemplos na classe c_1 e 80% na classe c_2 .

No caso extremo em que $r = n$, em que n representa o número de casos disponíveis, tem-se o método *leave-one-out*. No *leave-one-out*, a cada ciclo exatamente um exemplo é separado para teste, enquanto os $n - 1$ exemplos restantes são utilizados no treinamento do preditor. O desempenho é dado pela soma dos desempenhos verificados para cada exemplo de teste individual. Esse método produz uma estimativa mais fiel do desempenho preditivo do modelo. Porém, ele é computacionalmente caro, e geralmente aplicado somente em amostras de dados pequenas.

A principal crítica aos procedimentos de validação cruzada é que uma parte dos dados é compartilhada entre os subconjuntos de treinamento. Para $r \geq 2$ partições, uma proporção de $(1 - \frac{2}{r})$ dos objetos é compartilhada (Monard e Baranauskas, 2003). Por exemplo, utilizando 10 partições (*folds*), 80% dos objetos contidos nos subconjuntos de

treinamento são compartilhados. Logo, embora as partições de teste sejam distintas entre si, com $r > 2$ não se tem completa independência entre os subconjuntos de treinamento.

9.2.3 *Bootstrap*

No método *bootstrap*, r subconjuntos de treinamento são gerados a partir do conjunto de exemplos original (Figura 9.2(d)). Os exemplos são amostrados aleatoriamente desse conjunto, com reposição. Logo, um exemplo pode estar presente em um determinado subconjunto de treinamento mais de uma vez. Os exemplos não selecionados compõem os subconjuntos de teste. O resultado final é dado pela média do desempenho observado em cada subconjunto de teste.

Normalmente adota-se $r \geq 100$. A ideia básica é repetir o experimento um número alto de vezes e estimar o desempenho nesses experimentos replicados. Por esse motivo, o *bootstrap* é um procedimento custoso e é aplicado geralmente em amostras de dados pequenas.

Há vários estimadores *bootstrap*, e o mais comum é o e_0 (Jain et al., 1987). Neste, cada conjunto de treinamento tem n exemplos, amostrados com reposição do conjunto original, sendo n o número total de exemplos nesse conjunto. Cada exemplo tem probabilidade $1 - (1 - 1/n)^n$ de ser selecionado ao menos uma vez. Para n grande, essa probabilidade tende a $1 - 1/e = 0,632$, ou seja, a fração média de exemplos não repetidos nos conjuntos de treinamento é de 63,2%. Exemplos remanescentes formam o subconjunto de teste. O desempenho é dado pela média das iterações. A estimativa de desempenho obtida é estatisticamente equivalente à do *leave-one-out*, com uma menor variância.

9.3 Problemas de Duas Classes e o Espaço ROC

Por simplicidade, seja um problema com duas classes. Usualmente, uma classe é denotada positiva (+) e a outra é denominada negativa (-). Temos então a matriz de confusão ilustrada na Tabela 9.1, em que:

- VP corresponde ao número de verdadeiros positivos, ou seja, o número de exemplos da classe positiva classificados corretamente.
- VN corresponde ao número de verdadeiros negativos, ou seja, o número de exemplos da classe negativa classificados corretamente.
- FP corresponde ao número de falsos positivos, ou seja, o número de exemplos cuja classe verdadeira é negativa mas que foram classificados incorretamente como pertencendo à classe positiva.
- FN corresponde ao número de falsos negativos, ou seja, o número de exemplos pertencentes originalmente à classe positiva que foram incorretamente preditos como da classe negativa.

Ademais, $n = VP + VN + FP + FN$.

Tabela 9.1 Matriz de confusão para um problema com duas classes

		Classe predita	
		+	-
Classe verdadeira	+	VP	FN
	-	FP	VN

9.3.1 Medidas de Desempenho

A partir da matriz de confusão, uma série de outras medidas de desempenho pode ser derivada. Entre elas, temos (Monard e Baranauskas, 2003):

- *Taxa de erro na classe positiva*: proporção de exemplos da classe positiva incorretamente classificados pelo preditor \hat{f} , também conhecida como taxa de falsos negativos (TFN).

$$err_+(\hat{f}) = \frac{FN}{VP + FN} \quad (9.5)$$

- *Taxa de erro na classe negativa*: proporção de exemplos da classe negativa incorretamente classificados por \hat{f} , também conhecida como taxa de falsos positivos (TFP).

$$err_-(\hat{f}) = \frac{FP}{FP + VN} \quad (9.6)$$

- *Taxa de erro total*: dada pela soma dos valores da diagonal secundária da matriz, dividida pela soma dos valores de todos os elementos da matriz.

$$err(\hat{f}) = \frac{FP + FN}{n} \quad (9.7)$$

- *Taxa de acerto ou acurácia total*: calculada pela soma dos valores da diagonal principal da matriz, dividida pela soma dos valores de todos os elementos da matriz.

$$ac(\hat{f}) = \frac{VP + VN}{n} \quad (9.8)$$

- *Precisão*: proporção de exemplos positivos classificados corretamente entre todos aqueles preditos como positivos por \hat{f} .

$$prec(\hat{f}) = \frac{VP}{VP + FP} \quad (9.9)$$

- *Sensibilidade ou revocação*: corresponde à taxa de acerto na classe positiva. Também é chamada de taxas de verdadeiros positivos (TVP).

$$sens(\hat{f}) = rev(\hat{f}) = TVP(\hat{f}) = \frac{VP}{VP + FN} \quad (9.10)$$

- *Especificidade*: corresponde à taxa de acerto na classe negativa. Seu complemento corresponde à taxa TFP.

$$esp(\hat{f}) = \frac{VN}{VN + FP} = 1 - TFP(\hat{f}) \quad (9.11)$$

As medidas anteriores podem ser facilmente generalizadas para problemas com mais de duas classes pela consideração de cada classe como positiva em relação ao conjunto das demais classes (exceto a taxa de erro/acerto total, em que todas as classes são consideradas globalmente). Obtém-se um valor de desempenho para cada classe. Por exemplo, se a taxa que está sendo observada é a precisão, é obtido um valor de precisão para cada classe, no qual a classe em consideração é vista como positiva, enquanto as demais são consideradas negativas.

A precisão pode ser vista como uma medida de exatidão do modelo, e a revocação, como uma medida de sua completude. Uma precisão de 1,0 para uma classe C significa que cada item rotulado como pertencente à classe C realmente pertence a essa classe, mas não fornece informação a respeito do número de exemplos da classe C que não foram classificados corretamente. Por outro lado, uma revocação de 1,0 significa que cada exemplo da classe C foi rotulado como pertencendo à classe C , mas não diz nada a respeito de quantos outros exemplos foram classificados incorretamente como pertencendo à classe C . Desse modo, geralmente a precisão e a revocação não são discutidas isoladamente, mas são combinadas em uma única medida, como a medida-F, que é a média harmônica ponderada da precisão e a revocação:

$$F_m(\hat{f}) = \frac{(w + 1) \times rev(\hat{f}) \times prec(\hat{f})}{rev(\hat{f}) + w \times prec(\hat{f})} \quad (9.12)$$

Usando um peso igual a 1, que equivale a dar o mesmo grau de importância à revocação e à precisão, temos a medida F_1 :

$$F_1(\hat{f}) = \frac{2 \times prec(\hat{f}) \times rev(\hat{f})}{prec(\hat{f}) + rev(\hat{f})} \quad (9.13)$$

9.3.2 Análise ROC

Uma forma alternativa de avaliar classificadores em problemas binários, ou seja, que possuem apenas duas classes, é com o uso das curvas ROC (*Receiving Operating Characteristics*) (Fawcett, 2005). Seu uso inicial em AM foi reportado na avaliação e comparação de algoritmos (Spackman, 1989). Desde então sua utilização tem se estendido até mesmo à proposição de novos algoritmos e técnicas de AM com base na análise ROC (Prati e Flach, 2005).

O gráfico ROC é um gráfico bidimensional plotado em um espaço denominado espaço ROC, com eixos X e Y representando as medidas de falsos positivos (TFP) e taxa de verdadeiros positivos (TVP), respectivamente. O desempenho de um dado classificador pode ser plotado nessa curva, equivalendo a um ponto no espaço bidimensional.

Os principais aspectos de um espaço ROC são ilustrados na Figura 9.3. A linha diagonal representa classificadores que realizam previsões aleatórias. Qualquer classificador abaixo dessa linha pode então ser considerado pior que o aleatório. A figura inclui,

como exemplo, TFP e TVP de modelos gerados por três algoritmos. O ponto (0,1) representa classificações perfeitas, em que todos os exemplos positivos e negativos são classificados corretamente, sendo por isso denominado *céu ROC*. O ponto (1,0), por outro lado, representa o *inferno ROC*. O ponto (1,1) representa classificações sempre positivas, e o ponto (0,0), classificações sempre negativas. Assim, classificadores na região próxima do ponto (1,1) quase sempre rotulam os exemplos como positivos, e classificadores na região próxima do ponto (0,0) rotulam a maioria dos exemplos como negativa. Um classificador é considerado melhor que um outro se seu ponto no espaço ROC encontra-se acima e à esquerda do ponto correspondente ao segundo classificador (Prati, 2006).

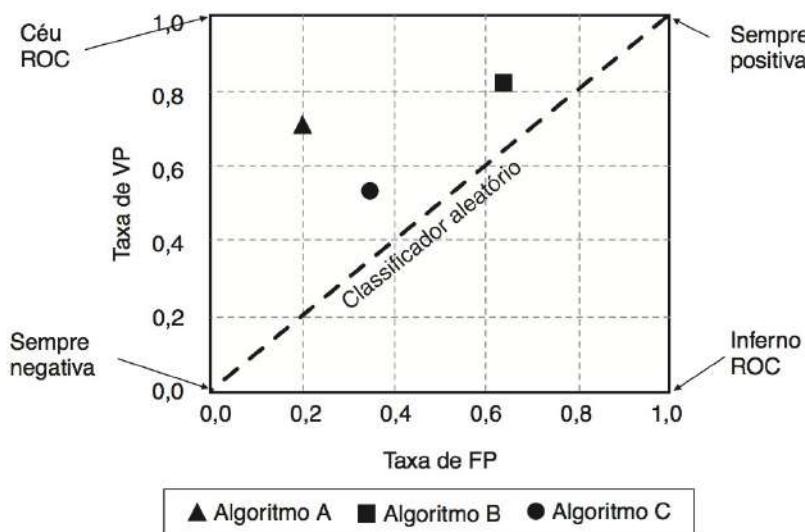


Figura 9.3 Espaço ROC com três classificadores.

Embora existam mecanismos para comparar diferentes classificadores com base em seus pontos no espaço ROC, o procedimento mais usual é gerar uma curva ROC. Nesse caso, é necessário empregar algum ranqueamento na classificação. Muitos classificadores produzem como saída um valor contínuo em sua classificação (ou podem ser adaptados para tal), que pode então ser usado nesse ranqueamento. Como exemplos temos: o NB, que produz a probabilidade de o exemplo pertencer a uma dada classe; as RNAs MLP, em que o neurônio da camada de saída produz um número contínuo; as SVMs, que empregam uma função sinal sobre um resultado que representa a margem de confiança na predição feita, entre outros.

Normalmente, para realizar a classificação final, os valores contínuos obtidos são discretizados de alguma maneira. Geralmente emprega-se algum limiar que permite dizer se a predição corresponde à classe positiva ou negativa. No caso do NB, por exemplo, o limiar adotado por *default* é normalmente um valor de probabilidade de 0,5. Usando limiares diferentes, novos valores de TFP e TVP podem ser obtidos. Plotando os valores de TFP e TVP para diferentes limiares e unindo esses pontos, tem-se como resultado uma curva ROC para cada técnica de classificação. Pode-se dessa maneira deixar a análise independente do limiar empregado. Inclusive, é possível escolher valores de limiar mais adequados para cada problema em particular (Matsubara, 2008).

Na Figura 9.4 são ilustradas duas curvas ROC, que podem hipoteticamente corresponder a curvas geradas por dois algoritmos de classificação distintos utilizando cinco valores para cada. Nesse caso, não há interseção entre as curvas, porém isso pode não ocorrer. Ao se comparar duas ou mais curvas, não havendo interseção entre elas, aquela que mais

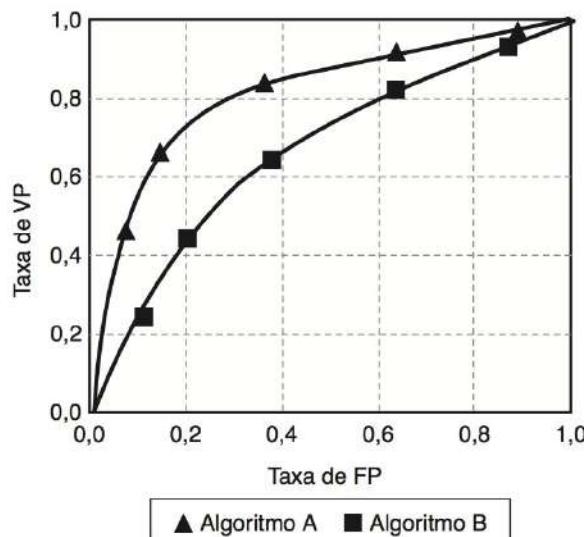


Figura 9.4 Exemplos de curva ROC.

se aproxima do ponto $(0,1)$ é a de melhor desempenho. No caso de haver interseções, cada algoritmo tem uma região em que é melhor que a do outro. É comum, porém, comparar o desempenho dos algoritmos em termos de uma medida única extraída da sua curva ROC: a área abaixo da curva ROC (AUC, do Inglês *Area Under ROC Curve*).

A medida AUC produz valores entre 0 e 1. Valores mais próximos de 1 são considerados melhores. Portanto, ao se comparar dois ou mais algoritmos segundo essa medida, aquele que possuir AUC mais próximo de 1 é considerado superior. Contudo, é recomendável comparar as medidas obtidas estatisticamente. Além disso, aconselha-se calcular o AUC em um procedimento de validação cruzada, assim como é sugerido para as outras medidas de desempenho, obtendo a média e desvio padrão de seus valores para diferentes partições dos dados.

Entre as principais vantagens da análise ROC estão a possibilidade de realizar medidas de desempenho independentes de condições como o limiar de classificação e também de custos associados às classificações incorretas e à distribuição das classes (Prati, 2006). De fato, o uso de diferentes limiares de classificação representa uma maior ou menor ênfase à classe positiva, permitindo lidar com questões de desbalanceamento das classes e de diferentes custos de classificação. A taxa de acerto/erro, por outro lado, é bastante influenciada por desbalanceamentos. Seja, por exemplo, um conjunto de dados com 90 exemplos positivos e 10 negativos. A obtenção de uma estimativa de taxa de acerto de 0,90 com esses objetos não é necessariamente indicativa de bom desempenho preditivo, uma vez que o modelo pode estar simplesmente classificando todos os exemplos na classe positiva.

Uma desvantagem da análise por curvas ROC é que ela é originalmente limitada a problemas de classificação binários. Existem trabalhos relacionados à generalização dessas curvas a problemas multiclasse, tais como a geração de múltiplas curvas, uma para cada classe, quando essa classe é então considerada positiva e as demais são tomadas como negativas (Provost e Domingos, 2001). Contudo, o uso mais difundido da análise ROC tem sido em problemas de classificação binários.

9.4 Testes de Hipóteses

Diversos estudos na área de AM requerem a comparação de dois ou mais algoritmos na solução de um ou mais problemas práticos. Nesse processo, é comum dividir o(s) conjunto(s) de dados com uma estratégia de amostragem e usar para todos os algoritmos comparados exatamente as mesmas partições dos dados. Sem perda de generalidade, iremos assumir nesta seção que o método usado é o *r-fold cross-validation* estratificado. Ou seja, a cada iteração da validação cruzada, todos os algoritmos usam a mesma partição de treinamento e de teste para obter seus resultados, e, dessa forma, a média de desempenho obtida por todos é calculada sobre os mesmos objetos. Isso significa impor que os algoritmos sejam comparados em igualdade de condições.

Após o processo descrito anteriormente, considerando um conjunto de dados em particular, para cada algoritmo tem-se a média de alguma medida de seu desempenho, como o erro ou o AUC médio e o seu desvio padrão nas partições. Determinar se um algoritmo é melhor do que o outro pelo simples exame de superioridade/inferioridade de médias não é aconselhável. Muitas vezes as diferenças verificadas não são significativas, e pode-se considerar os desempenhos obtidos equivalentes. É necessário conduzir um teste de hipóteses para a comparação dos desempenhos dos modelos em investigação.

Uma hipótese estatística é uma alegação sobre o valor de um ou mais parâmetros ou sobre a forma de uma distribuição de probabilidade (Devore, 2006). Se μ_1 e μ_2 são os erros médios de dois modelos em validação cruzada, por exemplo, uma hipótese possível é a expressão $\mu_1 - \mu_2 = 0$, ou seja, essas médias podem ser consideradas equivalentes. Outra é que $\mu_1 - \mu_2 > 0$, ou seja, que a média 1 é superior à média 2.

Nos testes de hipóteses, normalmente há duas suposições contraditórias em consideração, tal como $H_0 : \mu_1 - \mu_2 = 0$ versus $H_1 : \mu_1 - \mu_2 \neq 0$. Deve-se então decidir qual das duas hipóteses é a correta. A hipótese nula, representada por H_0 , é inicialmente assumida como verdadeira. H_1 é denominada hipótese alternativa. A hipótese nula é rejeitada em favor da hipótese alternativa se alguma evidência na amostra representada pelos desempenhos nos experimentos sugerir que H_0 é falsa. O teste de hipóteses é então realizado com o objetivo de rejeitar ou não (nesse caso, aceitar) a hipótese H_0 . A regra para decidir se H_0 é rejeitada é denominada procedimento de teste.

Em um procedimento de teste, tem-se (Devore, 2006):

- Uma estatística de teste, em função dos dados da amostra em que a decisão se baseia;
- Uma região de rejeição, que representa o conjunto de valores da estatística de teste para os quais H_0 é rejeitada. Ou seja, a hipótese nula é rejeitada se e somente se o valor da estatística calculada cair na região de rejeição.

Os procedimentos de teste podem cometer erros, devido à própria variabilidade das amostras sobre as quais são calculados. Um erro do tipo I ocorre quando a hipótese nula é rejeitada apesar de ser verdadeira. Já um erro do tipo II configura-se quando H_0 é falsa e não é rejeitada. Deve-se procurar procedimentos que apresentem um compromisso em relação à ocorrência de ambos os tipos de erro. Normalmente cometer erros do tipo I é considerado mais sério. A maioria dos testes realizados envolve então controlar a probabilidade de ocorrência de erro do tipo I, que é denotada usualmente pelo termo α . A região de rejeição é calculada de maneira a manter a probabilidade de ocorrência de erro

do tipo I sob controle. O valor α também é denominado nível de significância do teste. É comum empregar-se um valor de 0,05 para α . Isso equivale a dizer que o resultado do teste possui um nível de confiança de 95% de não ter rejeitado a hipótese nula quando ela é verdadeira.

Nas seções seguintes serão discutidos os procedimentos de teste comumente utilizados pela comunidade de AM. É importante destacar que, embora a realização desses testes seja recomendável para a comparação de dois ou mais modelos, não há ainda um consenso quanto à adequabilidade de vários procedimentos de teste existentes ao cenário típico dos experimentos de AM, em que as amostras utilizadas para avaliação de desempenho apresentam dependências. Descreveremos dois dos testes mais utilizados atualmente, porém recomendamos ao leitor atenção à literatura relacionada ao tema, que também discute outros procedimentos de teste, tais como Dietterich (1998), Salzberg (1997), Nadeau e Bengio (2003), Demsár (2006), e García e Herrera (2008).

Os testes que serão apresentados, *Wilcoxon signed-rank* e Friedman, são pareados e não paramétricos, não possuindo assim a restrição de as amostras sobre os quais são aplicados terem que seguir alguma distribuição (como a normal), o que não é garantido de ocorrer na prática quando modelos de AM são comparados. Ambos os testes são baseados em ranqueamentos, sendo também interessantes por permitirem comparar outras medidas de desempenho além das tradicionais, estabelecidas em algum tipo de erro/acerto preditivo. Pode-se, por exemplo, utilizá-los para comparar os tempos de treinamento de diferentes algoritmos.

Podemos distinguir dois casos na aplicação dos testes. O primeiro caso ocorre quando se deseja comparar o desempenho dos modelos em um único conjunto de dados. Esse cenário ocorre, por exemplo, quando se quer determinar que modelo possui destaque em um domínio em particular, representado por um único conjunto de dados. Dietterich (1998) e Salzberg (1997) discutem diferentes testes para esse caso.

No segundo caso, os modelos são comparados considerando vários conjuntos de dados. A avaliação em múltiplos conjuntos de dados é recomendada, uma vez que os resultados obtidos em um único conjunto de dados podem ser muito específicos para essa amostra em particular. Na proposta de um novo algoritmo de AM, por exemplo, usualmente deseja-se que este seja aplicável a uma gama de problemas relacionados. O mais indicado é então testar o seu desempenho em vários conjuntos de dados. O trabalho de Demsár (2006) sugere, nesse tipo de trabalho, comparar as médias de desempenho (em validação cruzada, por exemplo) dos modelos nos vários conjuntos de dados. Esses testes serão discutidos a seguir.

9.4.1 Comparando Dois Modelos

Consideraremos inicialmente a comparação de dois modelos. Normalmente tem-se nesse caso um modelo gerado por um algoritmo padrão, enquanto o outro modelo é obtido pelo uso de um novo algoritmo. Desejamos determinar se este algoritmo se destaca em relação ao já conhecido da literatura. A hipótese nula H_0 afirma que os desempenhos dos modelos comparados são equivalentes.

Um teste recomendado para a comparação de dois modelos é o *Wilcoxon signed-ranks* (Wilcoxon, 1943). Nesse teste, calculam-se inicialmente as diferenças nas medidas de desempenho dos algoritmos. Em seguida, os valores absolutos dessas diferenças são ranqueados (menores diferenças assumem primeiras posições e assim sucessivamente). Pelo

teste, comparam-se as posições das diferenças positivas e negativas entre os algoritmos. Dados dois algoritmos A e B, caso as diferenças sejam calculadas sempre considerando o desempenho de B menos o de A e usando uma medida de desempenho em que maiores valores são melhores (como taxa de acerto, AUC, precisão, entre outras), diferenças positivas indicam um melhor desempenho de B, enquanto as negativas refletem um melhor desempenho de A. No caso de medidas de desempenho como o erro, em que valores menores são melhores, basta aplicar o raciocínio inverso.

Seja d_i a diferença entre os desempenhos de A e B em um conjunto de dados i . Calculadas todas as diferenças, elas são ranqueadas de acordo com seus valores absolutos. No caso de empates, atribuem-se valores médios das posições na ordenação. Como exemplo, consideremos a Tabela 9.2, em que se comparam duas versões do algoritmo C4.5, a qual foi adaptada de uma tabela apresentada como exemplo em Demsár (2006).

Tabela 9.2 Exemplo de tabela de diferenças de resultados em teste de Wilcoxon

Conj. dados	C4.5	C4.5+m	Diferença	Dif.absoluta	Posição
Pulmão	0,583	0,583	0,000	0,000	1,5
Fungo	0,583	0,583	0,000	0,000	1,5
Atmosfera	0,882	0,888	+0,006	0,006	3,0
Mama	0,599	0,591	-0,008	0,008	4,0

Seja $R+$ a soma das posições (*ranks*) de conjuntos de dados em que o algoritmo B é melhor que o algoritmo A e $R-$ a soma de posições oposta. As posições das diferenças nulas são repartidas igualmente entre as duas somas. Se há um número ímpar de diferenças nulas, uma é ignorada. Temos então:

$$R+ = \sum_{d_i > 0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i) \quad (9.14)$$

$$R- = \sum_{d_i < 0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i) \quad (9.15)$$

Seja S a menor dessas somas. Alguns livros de Estatística apresentam tabelas com os valores críticos exatos para S , com N variando até 25. Para mais conjuntos de dados, a estatística do teste é:

$$z = \frac{S - \frac{1}{4}N(N-1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \quad (9.16)$$

Com $\alpha = 0,05$, a hipótese nula de que os algoritmos se comportam de maneira similar pode ser rejeitada se z é menor que $-1,96$.

9.4.2 Comparando Mais Modelos

Quando vários modelos são comparados, o número de comparações é maior e o teste deve ser alterado para levar isso em consideração. Se múltiplos testes são realizados, aumenta a probabilidade de ao menos um deles detectar uma diferença estatística quando esta não existe. Para J testes, a probabilidade de cometer ao menos um erro é de $1 - (1 -$

$\alpha)^J$ (Feelders e Verkooijen, 1996). Para $J = 20$ e $\alpha = 0,05$, por exemplo, a probabilidade de detectar-se uma ou mais diferenças estatísticas quando elas não existem é de 64%. Esse problema é conhecido como efeito da multiplicidade (Salzberg, 1997).

Nessa situação, Demsár (2006) recomenda a utilização do teste de Friedman (Friedman, 1937). Esse teste também é baseado na comparação de ranqueamentos de desempenhos. Contudo, nesse caso, o valor absoluto da medida de desempenho de cada algoritmo individualmente em cada conjunto de dados é considerado para realizar o ranqueamento. Logo, para cada conjunto de dados, realiza-se o ranqueamento dos algoritmos de acordo com seu desempenho (dos melhores para os piores). Em caso de empates, valores médios de posição (*rank*) são atribuídos.

Seja r_j^i a posição do desempenho do algoritmo j (entre A algoritmos) no conjunto de dados i (entre N conjuntos de dados). O teste de Friedman irá comparar os ranqueamentos médios R_j dos diferentes algoritmos. A hipótese nula H_0 afirma que todos os algoritmos são equivalentes e que suas posições no ranqueamento são então iguais. A estatística calculada é:

$$F_F = \frac{(N - 1) \chi_F^2}{N(A - 1) - \chi_F^2} \quad (9.17)$$

em que:

$$\chi_F^2 = \frac{12N}{A(A + 1)} \left[\sum_j R_j^2 - \frac{A(A + 1)^2}{4} \right] \quad (9.18)$$

A hipótese nula é rejeitada caso a estatística calculada seja maior que $F_{A-1,(A-1)(N-1)}$, em que $F_{A-1,(A-1)(N-1)}$ denota a distribuição de probabilidade F com $A-1$ e $(A-1)(N-1)$ graus de liberdade, que pode ser consultada em livros de Estatística. Caso a hipótese nula seja rejeitada, existe diferença de desempenhos, porém ela não aponta diretamente quais algoritmos possuem diferença. Para conseguir essa informação, deve-se prosseguir com um pós-teste.

No pós-teste, o desempenho de dois algoritmos em particular é estatisticamente diferente caso a diferença entre os seus valores médios de posição no ranqueamento seja maior ou igual ao valor de diferença crítica CD (do inglês *Critical Difference*):

$$CD = q_\alpha \sqrt{\frac{A(A + 1)}{6N}} \quad (9.19)$$

Se todos os algoritmos estão sendo comparados entre si em pares, os valores de q_α podem ser fornecidos pela estatística de Nemenyi (Nemenyi, 1963). O trabalho de García e Herrera (2008) menciona pós-testes alternativos para comparações entre todos os pares de algoritmos.

Quando a comparação é de vários algoritmos em relação a um único algoritmo (por exemplo, o desempenho de várias modificações de um algoritmo é comparado ao do algoritmo base), q_α pode ser menos restritivo, pois menos comparações são realizadas. Nesse caso, a estatística de Bonferroni-Dunn (Dunn, 1961) pode ser empregada. Na Tabela 9.3, adaptada a partir de Demsár (2006), são apresentados os valores de $q_{0,05}$ para diferentes números de algoritmos A sendo comparados, segundo os pós-testes de Nemenyi e Bonferroni-Dunn.

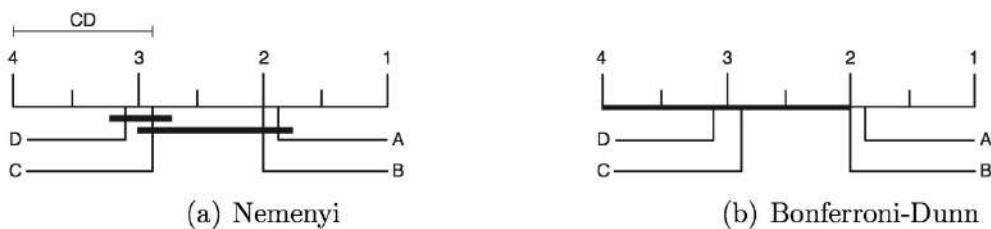
Em Demsár (2006) é apresentada ainda uma interessante maneira gráfica de dispor

Tabela 9.3 Valores de $q_{0,05}$ para diferentes pós-testes

A	2	3	4	5	6	7	8	9	10
Nemenyi	1,960	2,343	2,569	2,728	2,850	2,949	3,031	3,102	3,164
Bonferroni-Dunn	1,960	2,241	2,394	2,498	2,576	2,648	2,690	2,724	2,773

os resultados dos pós-testes. A Figura 9.5(a) apresenta o caso em que quatro diferentes algoritmos, denotados por A, B, C e D, são comparados entre si usando o pós-teste de Nemenyi. Inicialmente, uma escala contendo os ranqueamentos de 1 a 4 (quatro algoritmos sendo comparados) é desenhada de maneira a que a posição 1 é colocada mais à direita. Em seguida, os modelos são posicionados nessa escala de acordo com os valores médios de suas posições no ranqueamento calculado no teste de Friedman. Por exemplo, o modelo B apresentou-se em média na segunda posição com relação ao seu desempenho nos conjuntos de dados. Linhas horizontais conectam modelos cujos resultados não tiveram diferença estatística de acordo com o pós-teste. Além disso, o valor crítico usado para verificar se dois algoritmos são diferentes é posto acima da linha de ranqueamentos desenhada, a partir da esquerda. Pelo exame da Figura 9.5(a), é possível identificar que D possui resultado significativamente inferior aos de A e B. Nada pode ser dito a respeito de C, somente que nenhuma conclusão pode ser tirada com base nos experimentos realizados, pois um indivíduo não pode pertencer a duas populações diferentes em termos estatísticos.

Na Figura 9.5(b) é apresentado o caso em que os algoritmos A, B e C são todos comparados com o algoritmo base D, segundo o pós-teste de Bonferroni-Dunn. Nessa situação, o CD pode ser desenhado à esquerda e à direita da posição média do algoritmo de controle D. Qualquer algoritmo que possua posição fora dessa área tem desempenho significativamente diferente do controle.

**Figura 9.5** Representação gráfica de pós-testes.

9.5 Decomposição Viés-Variância da Taxa de Erro

Uma análise muito valiosa das taxas de erro dos algoritmos de aprendizado é fornecida pela chamada decomposição *Viés-Variância* do erro. A ideia básica por trás desse enquadramento teórico é que um algoritmo de aprendizado comete dois tipos de erros: *erros sistemáticos*, devido à linguagem de representação usada pelo algoritmo de aprendizado, e erros devidos à dependência do modelo gerado e ao conjunto de treinamento. Nesta seção, apresentaremos esse *framework* geral.

As origens da análise viés-variância podem estar relacionadas à *regressão quadrática* (Geman et al., 1992). Essa é uma ferramenta poderosa da teoria da amostragem da estatística para analisar cenários de aprendizado supervisionado que têm uma função

de custo quadrática. Dados uma variável objetivo e o tamanho do conjunto de treinamento, a formulação convencional da decomposição separa o erro esperado na soma de três quantidades não negativas:

- *Ruído do objetivo intrínseco*

Essa quantidade é o limite inferior do erro esperado de qualquer algoritmo de aprendizado. É o erro esperado do classificador de Bayes ótimo.

- *Quadrado do enviesamento*

Essa quantidade mede quão bem a predição média do algoritmo de aprendizado (sobre todos os possíveis conjuntos de treinamento cujo tamanho é igual ao tamanho do conjunto de treinamento dado) corresponde à variável objetivo.

- *Variância*

Essa qualidade mede o quanto a predição do algoritmo de aprendizado “aproxima” para diferentes conjuntos de objetos de um dado tamanho.

A decomposição viés-variância fornece diversas informações úteis. A mais relevante é conhecida como *equilíbrio viés-variância*. Quando algum aspecto de um algoritmo de aprendizado particular é modificado, pode haver efeito oposto no viés e na variância. Usualmente, quando se aumenta o número de graus de liberdade de um algoritmo, o viés decremente, mas a variância incrementa. O número ótimo de graus de liberdade otimiza esse equilíbrio entre viés e variância.

Para problemas de classificação, a função de custo quadrática é inapropriada porque os rótulos da classe não são numéricos. Diversas outras propostas para decompor o erro de classificação em viés e variância foram sugeridos, incluindo Breiman (1996b, 1998), Kong e Dietterich (1995), e Kohavi e Wolpert (1996). Não há consenso com relação à decomposição. Aqui seguiremos a decomposição sugerida em Kohavi e Wolpert (1996).

9.5.1 Definição de Viés e Variância

No aprendizado preditivo, é assumida a existência de uma função desconhecida f que mapeia o espaço de entrada X no espaço de saída Y . f representa uma distribuição de probabilidade condicional $P(y_f|\mathbf{x})$. Uma *hipótese* h gerada por um algoritmo de aprendizado representa uma distribuição similar $P(y_h|\mathbf{x})$. Vamos assumir que $P(y_f|\mathbf{x}) = 1$ para um valor de y_f e 0 para todos os outros. De igual forma, $P(y_h|\mathbf{x}) = 1$ para um valor de y_h e 0 para todos os demais.

Na *função de custo 0-1*, $\ell(y_f, y_h) = 0$ se e somente se $y_f = y_h$. O custo esperado para um único exemplo é:

$$E(c) = 1 - \sum_{y \in Y} P(Y_h = Y_f = y) \quad (9.20)$$

O custo esperado para um conjunto de exemplos \mathbf{X} de tamanho n é estimado como a média do custo calculada sobre os n exemplos. Kohavi e Wolpert (1996) mostram que:

$$E(c) = \sum_{\mathbf{x} \in \mathbf{X}} P(\mathbf{x})(\sigma_{\mathbf{x}}^2 + \text{viés}_{\mathbf{x}}^2 + \text{variância}_{\mathbf{x}}) \quad (9.21)$$

em que

$$\text{viés}^2_{\mathbf{x}} = \frac{1}{2} \sum_{y \in Y} (P(y_f = y) - P(y_h = y))^2 \quad (9.22)$$

$$\text{variância}_{\mathbf{x}} = \frac{1}{2} (1 - \sum_{y \in Y} P(y_h = y)^2) \quad (9.23)$$

$$\sigma_{\mathbf{x}}^2 = \frac{1}{2} (1 - \sum_{y \in Y} P(y_f = y)^2) \quad (9.24)$$

Essas definições de viés², variância e ruído (σ) têm as seguintes propriedades:

1. O componente viés² mede o quadrado da diferença entre a saída média do objetivo e a saída média do algoritmo. É uma quantidade real não negativa e igual a zero somente se $P(y_f|\mathbf{x}) = P(y_h|\mathbf{x})$ para todo \mathbf{x} e y . Essa propriedade é compartilhada pelo viés² para o erro quadrático.
2. O termo variância mede a *variabilidade* de $P(y_h|\mathbf{x})$. É uma quantidade real não negativa e igual a zero para um algoritmo que sempre faz a mesma previsão independentemente do conjunto de treinamento (por exemplo, o classificador bayesiano ótimo). Assim que o algoritmo se torna mais sensível a mudanças no conjunto de treinamento, a variância aumenta. Dada uma distribuição sobre o conjunto de treinamento, a variância mede a sensibilidade do algoritmo de aprendizado a mudanças no conjunto de treinamento e é independente de y_f .
3. O ruído é independente do algoritmo de aprendizado.

9.5.2 Medindo os Componentes Viés-Variância

Para um dado algoritmo e um dado conjunto de dados, estimamos o viés² e a *variância* como se segue.

1. Dividimos aleatoriamente o conjunto de dados em duas partes, T e E . T é usado como se ele fosse o “universo” a partir da qual amostramos o conjunto de treinamento, enquanto E é usado para avaliar os termos na decomposição.
2. Geramos N conjuntos de treinamento a partir de T , cada geração usando amostragem aleatória uniforme sem substituição.
3. Executamos o algoritmo de aprendizado para cada conjunto de treinamento e estimamos os termos *variância* da Equação 9.23 e *viés* da Equação 9.22 usando o classificador gerado para cada exemplo \mathbf{x} no conjunto de avaliação E . Todos os termos são estimados usando contadores de frequência.
4. O componente ruído σ^2 , que é o erro do classificador de Bayes ótimo, é muito difícil de estimar na prática. Usando um estimador de contagem de frequência, a estimativa de σ^2 poderá ser zero se todas as instâncias forem únicas (independentemente do valor real de σ^2). É usualmente agregado ao termo viés.

Informalmente, essa análise tenta capturar as seguintes quantidades: o termo viés mede o *erro persistente* do algoritmo de aprendizado. Esse erro não pode ser eliminado pelo aumento do número de classificadores treinados independentemente. O termo variância mede a *flutuação* do erro que é devido à utilização de uma amostra particular de exemplos de treinamento. Para um número crescente de exemplos, essa componente se reduz, e tende para zero quando o número de exemplos tende para infinito.

Deve-se notar que a soma do componente viés mais o componente variância pode diferir ligeiramente da taxa de erro medida usando validação cruzada. Isso ocorre porque são usados conjuntos de treinamento de diferentes tamanhos para estimar essas quantidades.

9.6 Considerações Finais

Neste capítulo foram discutidas questões relativas ao desenho de experiências e à avaliação de modelos preditivos. Esses aspectos são de grande relevância, uma vez que em geral os trabalhos envolvendo o uso de técnicas de AM devem incluir a realização de experimentos controlados.

A medida de desempenho mais empregada para avaliar classificadores é a taxa de erro (ou de acerto), enquanto em regressão é comum o exame do MSE do modelo. Em trabalhos na área médica, também é usual observar a sensibilidade do modelo, uma vez que a taxa de acerto para a doença de interesse, em geral, é considerada mais importante. No domínio de mineração de textos e recuperação de informação, por outro lado, é frequente a análise da precisão do modelo, ou seja, o quanto suas previsões positivas são confiáveis. Todas as medidas de desempenho apresentadas neste capítulo consideram o poder preditivo dos modelos obtidos, embora outros aspectos possam ser levados em consideração, tais como sua interpretabilidade, seu tamanho, entre outros. No caso de problemas de classificação binários, outra medida empregada na avaliação preditiva dos modelos é o AUC, proveniente da análise de curvas ROC.

Foram discutidos ainda métodos de amostragem de dados, com o objetivo de obter subconjuntos para uma estimativa apropriada do desempenho do preditor. Na prática, o *r-fold cross-validation* (estratificado para problemas de classificação) é o método de amostragem mais comumente utilizado, com um número de partições (*folds*) normalmente igual a 10. Em conjuntos de exemplos pequenos, o uso do *10-fold cross-validation* pode levar à obtenção de subconjuntos de teste com poucos dados. Nesses casos, é recomendável usar o *leave-one-out*, que produz estimativas mais fiéis do erro esperado, ou o *bootstrap*.

O *holdout* é muitas vezes usado para definir subconjuntos para o ajuste de parâmetros das técnicas a partir das partições de treinamento, também denominados subconjuntos de validação, tais como os usados pelas RNAs na estratégia de treinamento *early-stop* (Seção 7.1.4). Separa-se assim uma parte dos objetos de treinamento para avaliar o desempenho de diferentes combinações de parâmetros na predição do rótulo de novos objetos, diferentes daqueles empregados em sua indução. Encontrada a combinação de melhor desempenho, esta é utilizada na indução de um novo preditor sobre o subconjunto de treinamento completo (ou seja, incluindo o subconjunto de validação), o qual pode então ser finalmente avaliado sobre os objetos de teste. Também é comum realizar-se um procedimento de validação cruzada sobre cada partição de treinamento na definição dos parâmetros do modelo. Contudo, há claramente um maior custo computacional associado.

É importante observar que os dados de teste nunca devem ser considerados em ne-

nhuma etapa indutiva ou de ajuste do modelo, somente em sua avaliação final. Dessa forma garante-se uma fiel simulação da chegada de novos dados nunca apresentados ao modelo quando este estiver sendo utilizado na prática na predição dos rótulos de novos dados.

Normalmente, nos estudos experimentais realizados em AM é feita a avaliação de dois ou mais algoritmos, cujos resultados devem ser comparados. Nesse caso, é recomendável proceder a um teste de hipóteses, que poderá indicar a um nível de confiança, normalmente de 95%, se o desempenho dos algoritmos difere de fato ou não. A literatura a respeito da forma apropriada de comparar preditores com os testes estatísticos ainda é muito incipiente, e não há um consenso sobre os testes mais corretos a serem usados na prática. Optou-se por apresentar neste capítulo alguns dos testes mais empregados recentemente, embora outros trabalhos possam fornecer outras metodologias.

Por fim, apresentou-se a teoria de decomposição viés-variância do erro de preditores. Nesta, os erros cometidos pelos algoritmos de AM são decompostos em: (a) sistemáticos (viés), devido à linguagem de representação que adotam; (b) de flutuação (variância), devido à dependência em relação ao conjunto de treinamento utilizado. Esse tipo de análise fornece subsídios para as modificações dos algoritmos e as propostas de novas técnicas.

Parte III

Modelos Descritivos

Introdução aos Modelos Descritivos

As tarefas do aprendizado descritivo, ou não supervisionado, se referem à identificação de informações relevantes nos dados sem a presença de um elemento externo para guiar o aprendizado. Essencialmente, o aprendizado reside na identificação de propriedades intrínsecas aos dados de entrada, de maneira a construir representações desses dados que possam servir a diversos propósitos como auxílio a tomada de decisões ou descoberta de conhecimento. Essas técnicas são utilizadas principalmente quando o objetivo do aprendizado é encontrar padrões ou tendências que auxiliem no entendimento dos dados (Souto et al., 2003). Mais precisamente, no aprendizado não supervisionado não existem atributos meta. A partir do conjunto de dados \mathbf{X} , um algoritmo de AM não supervisionado aprende a representar as entradas submetidas segundo algum critério de qualidade.

Como já definido, as tarefas descritivas podem ser divididas em: sumarização, associação e agrupamento. A sumarização tem o objetivo de encontrar uma descrição simples e compacta dos dados. Para isso, podem ser utilizadas desde medidas estatísticas simples como mínimo, média, desvio padrão, até técnicas sofisticadas de visualização e de determinação de relações funcionais entre atributos (Han e Kamber, 2000; Mirkin, 2011). Algumas medidas estatísticas e técnicas de visualização mais simples foram descritas no Capítulo 2. Já a associação se refere à busca de padrões frequentes de associações entre os atributos de um conjunto de dados. O agrupamento, por sua vez, lida com a identificação de grupos nos dados de acordo com a similaridade entre os objetos. Nesta parte do livro será brevemente apresentada a associação (Capítulo 10), sendo dada mais ênfase às tarefas de análise de agrupamento (Capítulos 11 a 14). Nos capítulos a seguir, serão apresentados os principais aspectos do aprendizado não supervisionado, com uma atenção maior ao tema de agrupamento de dados.

A mineração de um *conjunto de itens* frequentes é um dos temas de grande importância na descoberta de conhecimento em bases de dados. Os primeiros trabalhos nessa área visavam identificar grupos de produtos que frequentemente eram comprados em conjunto para auxiliar em campanhas de marketing, por exemplo. No Capítulo 10 será detalhado o tema de mineração de padrões frequentes, sendo apresentados alguns dos principais algoritmos voltados para o tema.

Já as técnicas de agrupamento são instrumentos valiosos na análise exploratória de dados e encontram aplicações em várias áreas, tais como: Biologia, Medicina, Engenharia, Marketing, Visão Computacional, Sensoriamento Remoto e Bioinformática. Análise de agrupamento é apropriada para explorar e verificar estruturas presentes em um conjunto de dados.

Como já mencionado, a análise de agrupamento pertence ao paradigma de aprendizado não supervisionado, em que o aprendizado é dirigido aos dados, não requerendo

conhecimento prévio sobre as suas classes ou categorias (Mitchell, 1997). Considerando a descrição dos dados feita na Seção 2.1, o agrupamento é voltado para dados que não possuem um atributo de saída ou atributo alvo. Essa característica, aliada à falta de uma definição precisa e única do conceito de *cluster* (ou grupo), gera uma série de dificuldades tanto na escolha dos algoritmos mais apropriados como na avaliação dos resultados obtidos.

Assim, em muitos aspectos, a análise de agrupamento tem um caráter subjetivo, sendo seus resultados altamente influenciados pelo perfil do profissional que realiza a análise. Por exemplo, é bastante comum pesquisadores das áreas biológicas se aterem à utilização de algoritmos hierárquicos, devido à sua familiaridade com as estruturas em hierarquia obtidas pelos algoritmos, similares às taxonomias bastante comuns na área. E essa escolha, nem sempre, leva à utilização do algoritmo mais apropriado.

Em resumo, os pontos mais críticos ao se aplicar a análise de agrupamento estão relacionados à grande diversidade de critérios de agrupamento, heterogeneidade de critérios que podem definir uma estrutura dos dados, possibilidade de haver várias estruturas que descrevam um mesmo conjunto de dados e falta de conhecimento das estruturas verdadeiras presentes nos dados para guiar os resultados obtidos e também as comparações entre algoritmos. Todos esses problemas, bem como as principais abordagens para solucioná-los, serão detalhados ao longo dos Capítulos 11 a 14. Mais especificamente, os conceitos básicos de agrupamento, bem como as etapas essenciais à análise de agrupamento, serão apresentados no Capítulo 11. Alguns dos principais algoritmos de agrupamento serão apresentados no Capítulo 12. No Capítulo 13, serão apresentadas formas de combinar múltiplos modelos preditivos. Finalmente, no Capítulo 14 serão discutidas formas de planejar experimentos e de analisar os resultados obtidos na análise de agrupamento.

Capítulo 10

Mineração de Padrões Frequentes

A mineração de um *conjunto de itens* frequentes¹ é um dos tópicos de pesquisa mais ativos em descoberta de conhecimento em bases de dados. O trabalho pioneiro nessa área foi a análise de cestas de compras, especificamente a mineração de dados transacionais descrevendo o comportamento de compra de clientes. O objetivo é descobrir grupos de produtos que frequentemente são comprados em conjunto e, com base nesses grupos, inferir os produtos que serão comprados dado que foram comprados outros produtos. Desde então, um grande número de algoritmos eficientes tem sido desenvolvido. Neste capítulo, serão revisados alguns dos principais algoritmos para mineração de conjunto de itens frequentes, regras de associação, assim como suas extensões para sequências de itens, quando é importante considerar a ordem dos itens no conjunto.

10.1 Mineração de Conjuntos de Itens Frequentes

Seja $A = \{a_1, \dots, a_m\}$ o universo de m itens. Os itens podem ser produtos, componentes de equipamentos, opções de serviço etc. Qualquer subconjunto $I \subseteq A$ é chamado um conjunto de itens (*itemset*). Um conjunto de itens diz respeito a qualquer conjunto de produtos que podem ser comprados juntos.

Seja $T = (t_1, \dots, t_n)$ um conjunto de n transações denotado por banco de dados de transações. Cada transação é um par $\langle tid_i, k - items_i \rangle$, em que tid_i é a identificação da transação e $k - item_i \subseteq A$ é um conjunto de k itens. Um banco de dados de transações pode listar, por exemplo, o conjunto de produtos adquiridos por um cliente de um supermercado em uma compra, o conjunto de páginas visitadas por um usuário de um site em uma sessão etc. Cada transação é um conjunto de itens, mas alguns conjuntos de itens podem não aparecer no conjunto T . Uma transação $t \in T$ suporta o conjunto de itens I ou o conjunto de itens I está contido no $k - item_i$ da transação t , se e somente se, $I \subseteq t$. Ou seja, a transação t contém todos os elementos do conjunto de itens I . Por exemplo, uma transação em que foram comprados os itens queijo, pão e manteiga suporta ou dá suporte ao conjunto de itens formado pelos itens pão e queijo. Intuitivamente, dar suporte significa fortalecer ou testemunhar a favor. O conjunto $K(I)$ de transações que suporta o *itemset* I é designado por suporte do *itemset*. A partir de conjuntos frequentes, é possível derivar regras de associação. As regras de associação têm a forma de regras **se antecedente então consequente**, em que antecedente e consequente são *itemsets*. Por exemplo, se o cliente compra pão então também compra manteiga. Essas regras são calculadas a partir

¹Também denominado *itemset*.

dos dados e são de natureza probabilística. O grau de incerteza de uma regra é dado pela *confiança* da regra. É a relação entre o número de transações que incluem todos os itens no conjunto $\{consequente \cup antecedente\}$ para o número de transações que incluem todos os itens da antecedente.

O suporte pode ser ainda **absoluto** ou **relativo**. O suporte absoluto de um *itemset* é o número total de elementos do conjunto $K_T(I)$. O suporte relativo de um *itemset* é a fração de transações que o contém, que é calculado pela divisão do suporte absoluto pelo número de transações. A Figura 10.1 apresenta um banco de dados de transações, com 10 transações, e a enumeração de todos os conjuntos de itens frequentes usando o suporte mínimo de $s_{min} = 3$ ou $\sigma_{min} = 0,3 = 30\%$.

TID	Itens	0 itens	1 item	2 itens	3 itens
1	{a,d,e}	$\emptyset: 10$	{a}: 7	{a,c}: 4	{a,c,d}: 3
2	{b,c,d}		{b}: 3	{a,d}: 5	{a,c,e}: 3
3	{a,c,e}		{c}: 7	{a,e}: 6	{a,d,e}: 4
4	{a,c,d,e}		{d}: 6	{b,c}: 3	
5	{a,e}		{e}: 7	{c,d}: 4	
6	{a,c,d}			{c,e}: 4	
7	{b,c}				{d,e}: 4
8	{a,c,d,e}				
9	{b,c,e}				
10	{a,d,e}				

Figura 10.1 Um banco de dados de transações, com 10 transações, e a enumeração de todos os conjuntos de itens frequentes usando o suporte mínimo de $s_{min} = 3$.

Formalmente, o suporte absoluto de I com relação a T , $s_T(I)$, é definido pela expressão $s_T(I) = |K_T(I)|$. A expressão $\sigma_T(I) = \frac{1}{n} |K_T(I)|$ calcula o suporte relativo de I com respeito a T . Algumas vezes, $\sigma_T(I)$ é também chamado de frequência (*relativa*) de I em T .

O problema de mineração de conjuntos de itens frequentes pode ser formalmente definido como:

- **Dados:**

- um conjunto $A = \{a_1, \dots, a_m\}$ de itens,
- uma tabela $T = (t_1, \dots, t_n)$ de transações de sobre A ,
- um número σ_{min} tal que $0 < \sigma_{min} \leq 1$, o **suporte mínimo**.

- **Objetivos:**

- encontrar o conjunto de **itens frequentes**, tais que o suporte relativo de cada conjunto de itens é maior ou igual ao σ_{min} definido pelo usuário.
- encontrar o conjunto de regras de associação com confiança maior que um mínimo definido pelo utilizador.

Desde sua introdução em Agrawal et al. (1993), os problemas de aprendizado de *itemsets* frequentes e regras de associação receberam uma grande atenção. Na década de 1990, centenas de artigos de pesquisa foram publicados apresentando novos algoritmos ou melhorias nos algoritmos existentes para resolver esses problemas de aprendizado de uma forma mais eficiente. Um dos objetivos dessa área de AM é o de conhecer quais são os

artigos comprados em conjunto. Algumas das conclusões que se podem tirar são do tipo: *20% das pessoas que compram café também compram bolachas*. Essa informação, quando conhecida, poderá resultar num conjunto de ações que vão desde a promoção conjunta de artigos até alterações da sua localização no supermercado.

10.1.1 O Espaço de Busca

O espaço de busca de todos os possíveis conjuntos de itens para um conjunto de itens A contém exatamente $2^{|A|}$ *itemsets* diferentes. Isso pode ser representado por um subconjunto-reticulado, com o *itemset* vazio no topo e o conjunto de todos os itens na base. A Figura 10.2 ilustra o reticulado para 5 itens. Se $|A|$ é grande o suficiente, então uma proposta simples de gerar e contar os suportes de todos os *itemsets* sobre o banco de dados não pode ser alcançada dentro de um período de tempo razoável. A principal propriedade explorada pela maioria dos algoritmos de mineração de conjuntos de itens frequentes é que o suporte é monotonicamente decrescente com relação ao número de itens de um *itemset*. Ou seja, o conjunto de suporte de um conjunto de itens diminui sempre que se acrescenta um novo item. Formalmente, considere X e Y dois conjuntos de itens em um banco de dados de transações T sobre I , assim $X, Y \subseteq I$. É fácil mostrar que $X \subseteq Y \Rightarrow \text{suporte}(Y) \leq \text{suporte}(X)$. Essa regra é uma consequência imediata de que o conjunto de suporte de X está incluído no conjunto de suporte de Y . Por isso, se um *itemset* é pouco frequente, todos os seus superconjuntos devem ser pouco frequentes. Adicionalmente, a propriedade *todos os subconjuntos de um conjunto de itens frequente são frequentes* é válida. Essa é a *propriedade da monotonicidade* do suporte.²

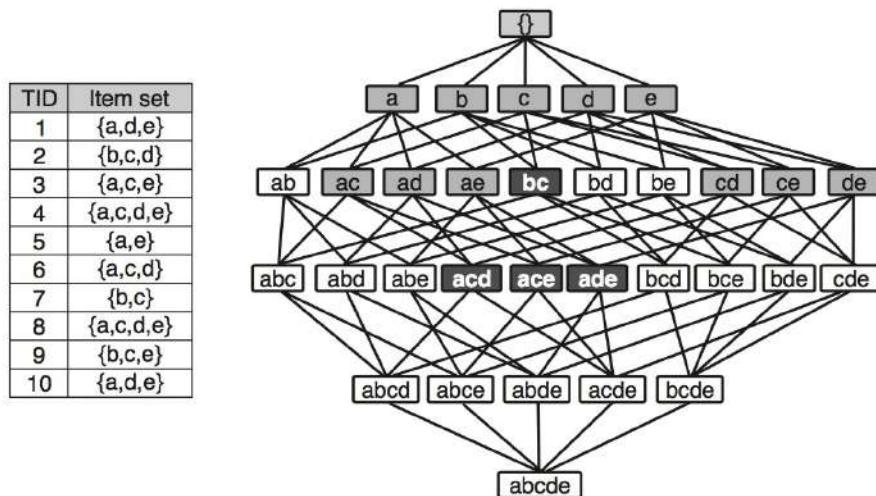


Figura 10.2 Um banco de dados de transações, com 10 transações, e o espaço de busca para encontrar todos os possíveis *itemsets* frequentes usando o suporte mínimo de $s_{\min} = 3$.

10.2 O Algoritmo Apriori

Apriori, introduzido em Agrawal et al. (1993); Agrawal e Srikant (1994), foi o primeiro algoritmo para mineração de *itemsets* e regras de associação. Ele se baseia no

²Uma função diz-se monotônica em x , se $x_1 < x_2$ implica que $f(x_1) < f(x_2)$.

princípio de que qualquer subconjunto de *itemsets* frequentes deve ser um *itemset* frequente. O algoritmo Apriori (Algoritmo 10.1) utiliza uma estratégia de busca em largura, *breadth-first*, com um algoritmo de geração e teste. Em cada nível são gerados os *itemsets* possíveis, tendo em conta os *itemsets* frequentes gerados no nível anterior. Após serem gerados, a frequência desses *itemsets* é testada, percorrendo novamente a base de dados de transações.

O algoritmo Apriori começa com a geração do conjunto F_1 de *itemsets* de tamanho 1, de modo que cada item é um membro do conjunto de *itemsets* candidatos. Os *itemsets* de tamanho $k + 1$ são obtidos a partir dos *itemsets* de tamanho k em dois passos. No primeiro passo, é realizada uma autocombinação sobre o conjunto F_k , quando um conjunto de candidatos com $k + 1$ *itemsets* é gerado pela combinação de F_k com ele mesmo. A união $X \cup Y$ dos *itemsets* $X, Y \in F_k$ é gerada se eles têm o mesmo $k - 1$ -prefixo. Esse passo pode ser realizado eficientemente se os *itemsets* estiverem em ordem lexicográfica. No segundo passo, passo de poda, $X \cup Y$ é inserido em F_{k+1} somente se todos os seus k -subconjuntos ocorrem em F_k . Depois disso, é necessário contar os suportes de todos os $k + 1$ *itemsets* candidatos. Para isso, o banco de dados é varrido, uma transação por vez, e os suportes de todos os *itemsets* candidatos que estão incluídos naquela transação são incrementados. Todos os *itemsets* que se tornam frequentes são inseridos em F_{k+1} . Com isso, o algoritmo executa uma busca em largura no espaço de pesquisa.

A Figura 10.1 apresenta um banco de dados de transações e a enumeração de todos os possíveis *itemsets* frequentes usando o suporte mínimo de $s_{\min} = 3$ ou $\sigma_{\min} = 0,3 = 30\%$. Há $2^5 = 32$ conjuntos de itens possíveis sobre $A = \{a, b, c, d, e\}$. Nesse banco de dados de transações, há 15 conjuntos de itens frequentes (para suporte mínimo de 0,3). Há 5 conjuntos frequentes com 1 elemento, 7 conjuntos frequentes com 2 elementos e 3 conjuntos frequentes com 3 elementos (Figura 10.1). Para esse nível de suporte, não há nenhum conjunto frequente com 4 elementos.

10.2.1 Regras de Associação

Além de encontrar quais são os termos que são adquiridos conjuntamente, também é útil saber quais são as combinações de termos que poderão ser descobertas e qual o seu nível de interesse nessas combinações. As combinações de termos são dadas pela construção de regras sob a forma $A \rightarrow B$, em que $A \cup B$ é um *itemset* frequente. O seu grau de interesse é representado pela confiança das regras, dada pela Equação 10.1, e que consiste na probabilidade de ocorrer um conjunto de termos dado que ocorreu um outro conjunto.

$$\text{confiança}(A \rightarrow B) = \frac{P(A \cup B)}{P(A)} = \frac{\text{suporte}(A \cup B)}{\text{suporte}(A)} \quad (10.1)$$

A segunda fase do algoritmo Apriori gera regras a partir dos conjuntos de termos frequentes encontrados e que obejam ao critério de confiança mínima. O algoritmo básico, descrito no Algoritmo 10.2, gera para cada *itemset* frequente todos os subconjuntos não vazios. Sendo s um subconjunto do *itemset* i , é calculada a confiança da regra $s \rightarrow \{I \setminus s\}$. O algoritmo seleciona as regras cuja confiança é superior à confiança mínima definida pelo usuário.

Retomando o exemplo da Figura 10.1 e para o conjunto de conjuntos frequentes obtidos com suporte mínimo de 0,3, serão extraídas todas as regras, considerando o valor de

Algoritmo 10.1 O algoritmo Apriori para gerar *itemsets* frequentes

Entrada: Um banco de dados de transações DB ;

Um limiar do suporte mínimo σ ;

Saída: Todos os *itemsets* com suporte maior que σ

- 1 Varrer o banco de dados DB uma vez;
- 2 Coletar C_1 , o conjunto de itens frequentes e o suporte de cada item;
- 3 $k \leftarrow 1$;
- 4 **enquanto** $C_k \neq \{\}$ **faça**
- 5 **para cada** transação $I \in DB$ **faça**
- 6 **para cada** itemset candidato $X \in C_k$ **faça**
- 7 se $X \subset I$ **então**
- 8 Incrementa o suporte de X ;
- 9 **fim**
- 10 **fim**
- 11 **fim**
- 12 /* Extrai todos os *itemsets* frequentes */ ;
- 13 $C_k \leftarrow \{X | X : \text{Suporte} \geq \sigma\}$;
- 14 /* Gera os novos *itemsets* candidatos */;
- 15 **para cada** $X, Y \in C_k$ **faça**
- 16 se $X[i] = Y[i]$ para $1 \geq i \geq k - 1 \wedge X[k] < Y[k]$ **então**
- 17 $I = X \cup \{Y[k]\}$;
- 18 se $\forall J \in I, |J| = k : J \in C_k$ **então**
- 19 $C_{k+1} \leftarrow C_{k+1} \cup I$
- 20 **fim**
- 21 **fim**
- 22 **fim**
- 23 $k \leftarrow k + 1$;
- 24 **fim**

80% para o parâmetro de confiança, conforme ilustrado na Tabela 10.1. O processo consiste em testar todas as combinações de termos que possam formar regras. Deverão existir, pelo menos, um item como antecedente da regra e outro como consequente. A Figura 10.3 ilustra o processo de procura de regras de associação a partir do *itemset* frequente $\{a, d, e\}$. A geração de regras de associação não requer mais nenhuma passagem pela base de transações. O Algoritmo 10.2 pode ser otimizado tendo em conta que, quando um item é movido do antecedente para o consequente, a confiança não pode aumentar.

10.2.2 Discussão

A proposta de nível de confiança do algoritmo Apriori implica diversas varreduras sobre o banco de dados para calcular o suporte dos *itemsets* frequentes candidatos. Como alternativa, diversos algoritmos significantemente reduziram essas varreduras por meio da geração de coleções de *itemsets* candidatos em uma estratégia de busca em profundidade. O primeiro algoritmo proposto foi o algoritmo Eclat (do inglês *Equivalence Class Trans-*

Algoritmo 10.2 O algoritmo Apriori para gerar regras de associação

Entrada: Itemset frequente I ;
Limiar de confiança $conf_{min}$;
Saída: R : Todas as regras de associação em I com confiança maior que $conf_{min}$

- 1 Gera todos os subconjuntos não vazios de I ;
- 2 $R = \emptyset$;
- 3 **para cada** subconjunto não vazio s de I **faça**
- 4 Avalia a confiança da regra $s \rightarrow \{I \setminus s\}$;
- 5 **se** $\frac{\text{suporte}(I)}{\text{suporte}(s)} \geq conf_{min}$ **então**
- 6 $R \leftarrow R \cup \{s \rightarrow \{I \setminus s\}\}$;
- 7 **fim**
- 8 **fim**

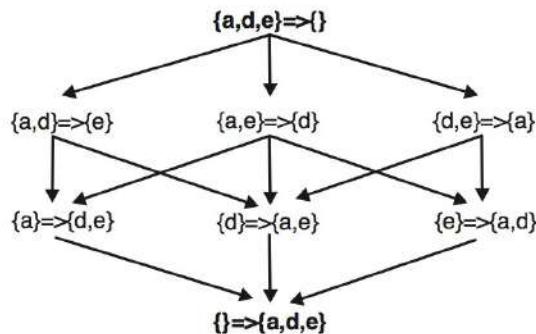


Figura 10.3 Espaço de busca de regras de associação para um itemset frequente.

formation), desenvolvido por Zaki (2000), e o algoritmo FP-growth (do inglês *Frequent Pattern Growth*) criado por Han et al. (2004). Esse último algoritmo, descrito na Seção 10.3, utiliza uma árvore de prefixos (*trie*) para armazenar *itemsets* (Figura 10.4). Isso evita as autocombinações requeridas no Apriori para geração dos candidatos. Para gerar todas as possíveis extensões de um *itemset* por meio de um único item, simplesmente adiciona-se o item à árvore de sufixo. Esse esquema de busca gera cada conjunto de itens candidatos no máximo uma vez. O algoritmo FP-growth foi utilizado como um bloco de construção em *itemsets* frequentes (Chi et al., 2004) e mineração de sequências em fluxo contínuo de dados (Pei et al., 2001).

10.3 O Algoritmo FP-growth

A estratégia de busca por profundidade e árvores de sufixo utilizadas pelo algoritmo FP-growth são empregadas na maioria dos algoritmos de mineração de padrões frequentes aplicados a dados de fluxo contínuo (Chi et al., 2004). Para a construção de regras de associação, o algoritmo FP-growth procede em duas fases. Na primeira fase constrói uma estrutura de dados a FP-tree percorrendo a base de dados duas vezes. A FP-tree é então usada para encontrar as regras de associação. O pseudocódigo³ do algoritmo para

³Seguimos o pseudocódigo apresentado em Han et al. (2004).

Tabela 10.1 Regras extraídas

Regra	Confiança	Suporte do <i>itemset</i>
$\{b\} \rightarrow \{c\}$	100,0%	30%
$\{d, e\} \rightarrow \{a\}$	100,0%	40%
$\{e\} \rightarrow \{a\}$	85,7%	60%
$\{a\} \rightarrow \{e\}$	85,7%	60%
$\{d\} \rightarrow \{a\}$	83,3%	50%
$\{a, d\} \rightarrow \{e\}$	80,0%	40%

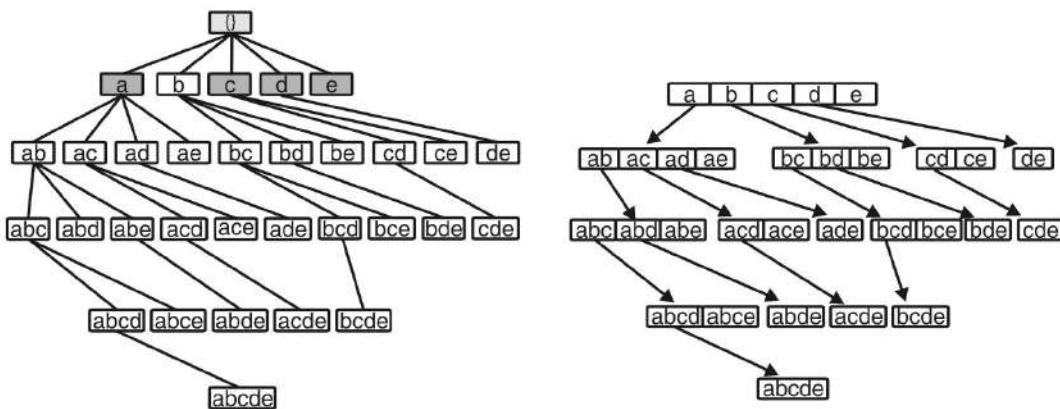


Figura 10.4 O espaço de pesquisa utilizando a profundidade e a árvore de prefixos correspondente para 5 itens.

construir a FP-tree é apresentado no Algoritmo 10.3. A ideia base consiste em percorrer o espaço de procura em profundidade como é mostrado na Figura 10.4.

O processo de construção de um FP-tree está apresentado na Figura 10.5. O algoritmo executa duas varreduras sobre o banco de dados. Na primeira varredura, ele define o conjunto de itens frequentes (*1-itemsets*) e suas contagens de suporte. O conjunto de itens frequentes é ordenado na ordem decrescente do seu suporte e armazenado em uma matriz **L**. No caso da base de transações da Figura 10.5, o item mais frequente é o *a*, seguido de *b*, *c*, *d* e *e*. O algoritmo faz uma segunda varredura sobre a base de transações, construindo a árvore de padrões frequentes, FP-tree, como se segue. Primeiro, cria-se o nó raiz da árvore, rotulado com *null*. Para cada transação presente no banco de dados, os itens são processados em ordem decrescente de suporte. Por exemplo, a primeira transação $\{a, b\}$, gera o percurso: *null* $\rightarrow a \rightarrow b$. Cada nó tem associado um contador de frequência com o valor 1. A segunda transação gera um novo conjunto de nós, correspondente ao percurso: *null* $\rightarrow b \rightarrow c \rightarrow d$. Esse percurso é disjunto do primeiro porque as transações não compartilham um prefixo comum. Como *b* aparece nos dois percursos, é estabelecida uma ligação que possibilita calcular a frequência de *b*. A terceira transação partilha um prefixo comum (o item *a*, com a primeira transação). Por esse motivo, o percurso *null* $\rightarrow a \rightarrow c \rightarrow d \rightarrow e$ se sobrepõe ao primeiro percurso, e a frequência de *a* é incrementada. Esse processo continua até todas as transações terem sido processadas.

A razão para a primeira varredura no banco de dados e processamento de transações em ordem decrescente de suporte é que quanto mais frequentemente os itens são dispostos

Algoritmo 10.3 O algoritmo FP-tree

Entrada: Um Banco de Dados de Transações DB ;
 Um limiar do suporte mínimo σ ;

Saída: Todos os *itemsets* com suporte maior que σ

- 1 Varrer o banco de dados DB uma vez ;
- 2 Coletar F , o conjunto de itens frequentes e o suporte de cada item ;
- 3 Ordenar F em ordem decrescente de suporte como $Flist$;
- 4 Criar a raiz de FP-tree, T , e rotulá-la como *null* ;
- 5 **para cada transação** $t \in DB$ **faça**
- 6 Selecionar os itens frequentes em t ;
- 7 Ordená-los de acordo com $Flist$;
- 8 Seja $[i|It]$ os itens frequentes ordenados em t ;
- 9 **Chamar** *insere_arvore*($[i|Its], T$);
- 10 **fim**
- 11 **Função** *insere_arvore*($[i|Its], T$) ;
- 12 **se** T tem um filho N rotulado i **então**
- 13 Incremente os contadores de N com 1 ;
- 14 **fim**
- 15 **senão**
- 16 Crie um novo nó, N , com seu contador inicializado com 1, seu pai ligado a T , e seu nó ligado aos nós com o mesmo rótulo i ;
- 17 **se** Its é não vazio **então**
- 18 Chamar *insere_arvore*(Its, N);
- 19 **fim**
- 20 **fim**

mais próximo da raiz da FP-tree, mais provavelmente serão compartilhados. Dessa forma, a representação da FP-tree do banco de dados é mantida tão pequena quanto possível.

10.4 Sumarização de *Itemsets*

A abordagem utilizada por um algoritmo de descoberta de regras de associação pode fazer com que o número de regras geradas pelo algoritmo seja muito grande, e que nem todas as regras encontradas sejam interessantes. Por exemplo, o conjunto de todos os *itemsets* frequentes, pode ser completamente representado por um conjunto com menos elementos eliminando todos os *itemsets* que são subconjuntos de outros *itemsets* frequentes. A *propriedade da monotonicidade* do suporte sugere uma representação sumarizada do conjunto de *itemsets* frequentes:

- *Itemsets frequentes máximas*

Um conjunto de itens é maximal se é frequente mas nenhum dos seus superconjuntos próprios é frequente.⁴

⁴ A é um superconjunto próprio de B se e somente se a cardinalidade de A é maior que a de B e todos os elementos de B pertencem a A .

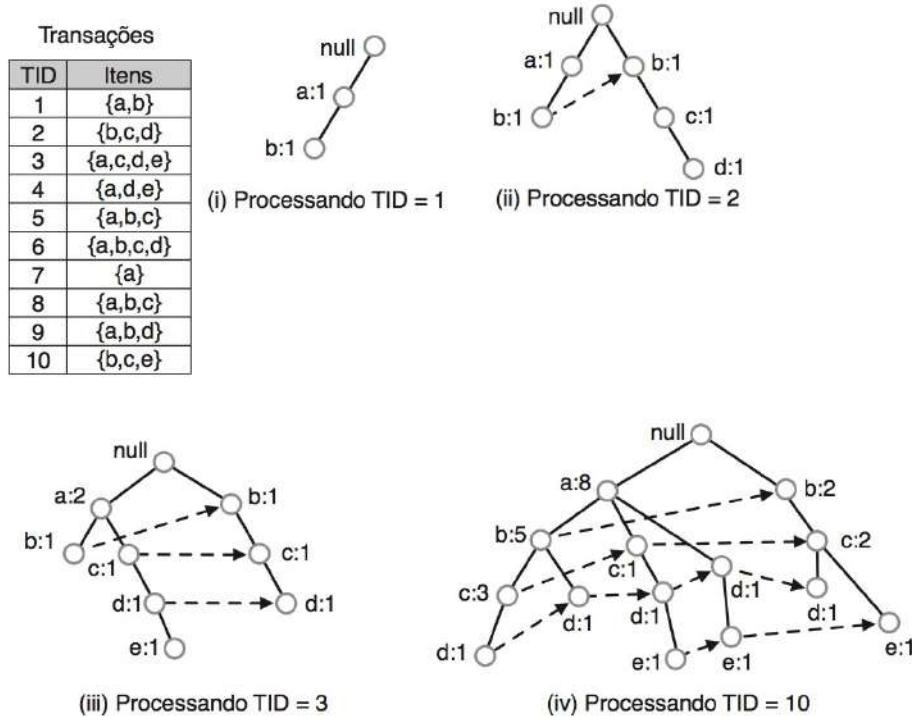


Figura 10.5 Construção de uma FP-tree.

- *Itemsets frequentes fechados*

Um conjunto frequente é chamado *fechado* se e somente se ele não tem superconjuntos frequentes com a *mesma frequência*.

No exemplo da Figura 10.1, há 13 conjuntos frequentes fechados (suporte mínimo 0,3): $\{b, c\}$, $\{d, c, a\}$, $\{e, c, a\}$, $\{c, a\}$, $\{d, c\}$, $\{e, c\}$, $\{d, e, a\}$, $\{d, a\}$, $\{d\}$, $\{e, a\}$, $\{a\}$, $\{c\}$, $\{e\}$, e 4 conjuntos de itens maximais, que são: $\{b, c\}$, $\{a, c, d\}$, $\{a, c, e\}$, $\{a, d, e\}$. Todos os *itemsets* frequentes podem ser vistos como um subconjunto de pelo menos um dos conjuntos maximais.

O seguinte relacionamento é válido entre esses conjuntos: $Maximal \subseteq Fechado \subseteq Frequente$. Os *itemsets* maximais são um subconjunto dos *itemsets* fechados. A partir dos *itemsets* maximais, é possível derivar todos os *itemsets* frequentes (mas não o seu suporte) calculando todas as interseções não vazias, justificado pelo fato de que, se um *itemset* é frequente, todos os seus subconjuntos são também frequentes.

O conjunto de todos os conjuntos de itens fechados preserva o conhecimento sobre o valor do suporte de todos os *itemsets* frequentes, devido ao fato de que todos os subconjuntos de um *itemset* fechado têm o mesmo suporte.

10.4.1 Heurísticas para Seleção de Regras de Associação

Os algoritmos de regras de associação tendem a gerar um número excessivo de regras. Nos últimos anos, têm sido propostas várias medidas para extraír padrões *interessantes* a partir de grandes bases de dados. A ideia consiste em selecionar um subconjunto de padrões ou regras que de alguma forma sejam mais relevantes. Nesta seção, serão apresentadas as características gerais de uma medida de interesse e definidas algumas medidas concretas para extração de padrões interessantes. Uma medida objetiva deverá ser base-

ada nos dados e ser independente do domínio de dados considerado. Foram definidos por Piatetsky-Shapiro (1991) três princípios a que qualquer medida (M) deve obedecer:

- Se A e B são duas variáveis estatisticamente independentes, $P(A, B) = P(A).P(B)$ e $M = 0$. Isso significa que quando duas variáveis são estatisticamente independentes, a medida de interesse terá valor zero;
- A medida M cresce se $P(A, B)$ crescer e todos os restantes parâmetros permaneçam constantes;
- A medida M decresce se $P(A)$ ou $P(B)$ decrescerem, mantendo-se constantes todos os demais parâmetros.

O primeiro princípio estabelece que os padrões que ocorrerem aleatoriamente não são de interesse. O segundo princípio estabelece que o interesse de uma medida deverá ser tanto maior quanto maior for a sua significância estatística. E, finalmente, o terceiro é utilizado para comparar os valores de interesse de padrões com mesma significância estatística.

Assuma que estamos interessados em estudar a relação entre as pessoas que bebem chá e café. Após obter informação sobre as preferências de 1000 pessoas, sumarizamos essa informação na Tabela 10.2. Essa informação pode ser usada para avaliar a associação da regra $\{\text{Chá}\} \rightarrow \{\text{Café}\}$. Assim o suporte da regra é $150/1000 = 0,15$, e a confiança $0,15/0,2 = 0,75$. A confiança da regra é elevada, no entanto a probabilidade de uma pessoa beber café, independentemente de beber chá, é de 80%. Sabendo que uma pessoa bebe chá, diminui a probabilidade de beber também café! A regra $\{\text{Chá}\} \rightarrow \{\text{Café}\}$ de fato é enganadora.

Tabela 10.2 Preferências sobre o consumo de Chá e Café de 1000 consumidores

	Café	Não Café	Total
Chá	150	50	200
Não Chá	650	150	800
Total	800	200	1000

Coeficiente de Interesse ou *Lift*

O coeficiente de interesse, ou *Lift*, reflete a noção estatística de independência entre duas variáveis aleatórias. Foi abordado por muitos autores como uma medida para avaliar os níveis de associação. Essa métrica é definida pelo quociente entre a probabilidade conjunta de duas variáveis em relação à sua probabilidade, pressupondo a hipótese de independência. Ela é calculada pela Equação 10.2. Nessa equação, o valor 1 corresponde à independência estatística entre as variáveis A e B .

$$I(A, B) = \frac{P(A, B)}{P(A)P(B)} \quad (10.2)$$

O coeficiente de interesse aplicado a regras de associação costuma ser designado por *lift*. O *lift* não é senão o quociente entre a confiança e o valor esperado para a confiança.

A confiança esperada é o número de transações que incluem o consequente dividido pelo número total de transações, conforme ilustrado pela Equação 10.3.

$$lift(A \rightarrow B) = \text{confiança}(A \rightarrow B) / \text{suporte}(B) = \frac{\text{suporte}(A \cup B)}{\text{suporte}(A) \times \text{suporte}(B)} \quad (10.3)$$

Um valor de *lift* igual a 1 indica que *A* e *B* são independentes. Valores de *lift* inferiores a 1 indicam que *A* e *B* são negativamente correlacionados, enquanto valores superiores a 1 indicam uma correlação positiva.

O *lift* da regra $\{\text{Chá}\} \rightarrow \{\text{Café}\}$, apresentada na Tabela 10.2, é $0,15/(0,2 \times 0,8) = 0,9375$. Esse resultado evidencia a correlação negativa entre Chá e Café.

O *lift* é uma medida para o desvio da regra em relação à independência estatística entre o antecedente e consequente de uma regra de associação. Toma valores entre 0 e infinito:

- Um valor de *lift* superior a 1 indica que *A* e *B* aparecem mais frequentemente juntos do que o esperado, isso significa que a ocorrência de *A* tem um efeito positivo sobre a ocorrência de *B*.
- Um *lift* menor do que 1 indica que *A* e *B* aparecem com menos frequência do que o esperado em conjunto, indicando que a ocorrência de *A* tem um efeito negativo sobre a ocorrência de *B*.
- Um valor próximo de 1 indica que *A* e *B* aparecem quase sempre juntos. Isso significa que a ocorrência de *B* não tem efeito sobre a ocorrência de *A*.

Convicção

A última medida apresentada, a convicção em uma regra, mede o quanto convincente é a regra. A convicção pode ser interpretada como o quociente da frequência esperada de *A* ocorrer sem *B* (ou seja, a frequência de erro da regra) como se *A* e *B* fossem independentes dividido pela frequência de previsões incorretas. Ela é definida pela Equação 10.4.

$$\text{convicção}(A \rightarrow B) = \frac{1 - \text{suporte}(B)}{1 - \text{confiança}(A \rightarrow B)} \quad (10.4)$$

A convicção da regra $\{\text{Chá}\} \rightarrow \{\text{Café}\}$, apresentada no exemplo anterior, é: $(1 - 0,8)/(1 - 0,75) = 0,8$. Tal como a medida *lift*, na medida de convicção valores inferiores a 1 indicam que a associação entre *A* e *B* é aleatória.

10.5 Considerações Finais

Em mineração de dados, aprendizado de regras de associação é um tema muito pesquisado e um método popular para a descoberta de relações interessantes entre as variáveis em grandes bases de dados. As regras de associação podem ser usadas como base para decisões sobre atividades de marketing, como, por exemplo, promoção de preços ou colocação de produto. Recentemente, a análise de regras de associação tem sido usada em muitas áreas de aplicação, incluindo sistemas de recomendação, mineração na Web, detecção de intrusos e bioinformática.

Capítulo 11

Análise de Agrupamentos

O objetivo de uma técnica de agrupamento é encontrar uma estrutura de *clusters* (grupos) nos dados em que os objetos pertencentes a cada *cluster* compartilham alguma característica ou propriedade relevante para o domínio do problema em estudo, ou seja, são de alguma maneira similares (Jain e Dubes, 1988). Por exemplo, a Figura 11.1 ilustra um conjunto de objetos (11.1(a)) agrupados de três maneiras diferentes. Visualmente identificamos que uma das divisões dos objetos em dois grupos agrupa os objetos pela forma 11.1(b) e a outra divide os objetos pelo preenchimento 11.1(c). A divisão em quatro grupos considera uma combinação dessas características 11.1(d). Cada uma dessas maneiras de agrupar os objetos é uma estrutura ou um modelo que descreve os dados e poderia ter sido obtida por meio de um algoritmo de agrupamento.

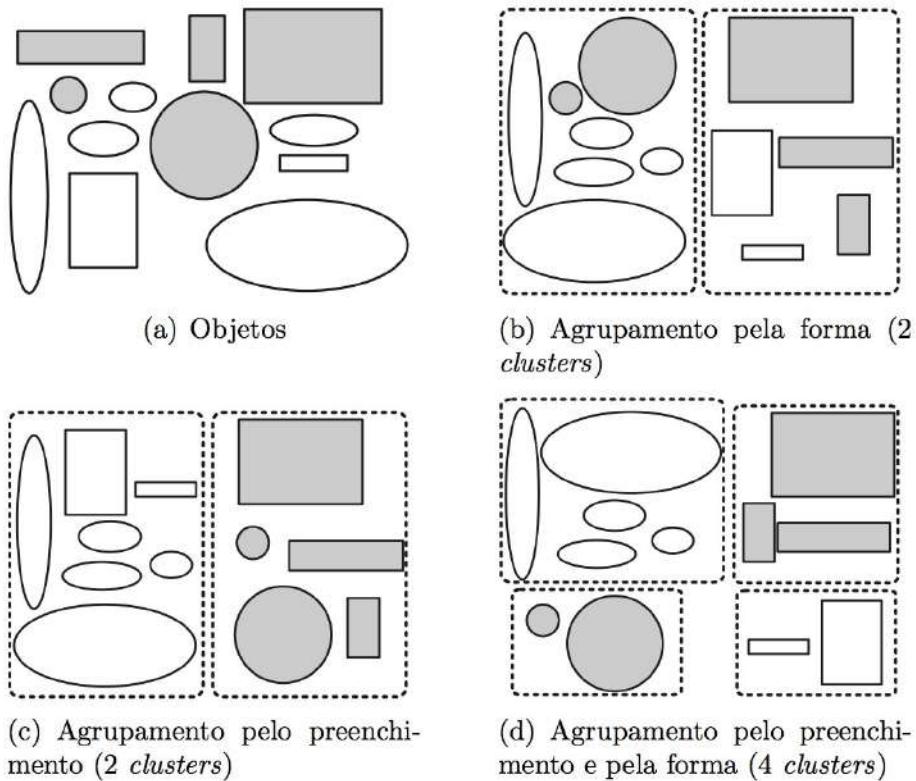


Figura 11.1 Objetos agrupados de diferentes maneiras.

Neste capítulo serão descritos os principais aspectos relacionados à análise de agrupamento. Na Seção 11.1 serão apresentados as definições básicas sobre *clusters* e os critérios de agrupamento e relação desses elementos com propriedades dos dados que devem ser

consideradas ao se fazer análise de agrupamento. Em seguida, na Seção 11.2, são descritas as etapas envolvidas no processo de análise de agrupamento: preparação dos dados, escolha da medida de proximidade, aplicação de algoritmos de agrupamento, validação e interpretação dos resultados.

11.1 Definições Básicas

Considerando os objetos pontos em um espaço de dimensão d , um *cluster* pode ser visto como uma coleção de objetos próximos ou que satisfazem alguma relação espacial. Embora a ideia do que constitui um *cluster* seja intuitiva (grupos de objetos similares), não existe uma definição formal única e precisa para esse conceito. Ao contrário, existe uma grande variedade de definições na literatura. Isso é resultado da grande diversidade de visões/objetivos dos pesquisadores de diferentes áreas que utilizam/desenvolvem algoritmos de agrupamento. Algumas definições comuns para *cluster* são (Barbara, 2000):

- *Cluster bem separado*: um *cluster* é um conjunto de pontos tal que qualquer ponto em um determinado *cluster* está mais próximo (ou é mais similar) a cada outro ponto nesse *cluster* do que a qualquer ponto não pertencente a ele.
- *Cluster baseado em centro*: um *cluster* é um conjunto de pontos tal que qualquer ponto em um dado *cluster* está mais próximo (ou é mais similar) ao centro desse *cluster* do que ao centro de qualquer outro *cluster*. O centro de um *cluster* pode ser um centroide, como a média aritmética dos pontos do *cluster*, ou um medoide (isto é, o ponto mais representativo do *cluster*).
- *Cluster contínuo ou encadeado* (vizinho mais próximo ou agrupamento transitivo): um *cluster* é um conjunto de pontos tal que qualquer ponto em um dado *cluster* está mais próximo (ou é mais similar) a um ou mais pontos nesse *cluster* do que a qualquer ponto que não pertence a ele.
- *Cluster baseado em densidade*: um *cluster* é uma região densa de pontos, separada de outras regiões de alta densidade por regiões de baixa densidade.
- *Cluster baseado em similaridade*: um *cluster* é um conjunto de pontos que são similares, enquanto pontos em *clusters* diferentes não são similares.

Cada possível definição de *cluster* resulta em um critério de agrupamento que essencialmente é uma forma de selecionar uma estrutura de *clusters* (ou modelo) que melhor se ajuste a um determinado conjunto de dados (Estivill-Castro, 2002). Cada algoritmo de agrupamento é baseado em um critério de agrupamento e usa uma medida de proximidade e um método de busca para encontrar uma estrutura ótima ou subótima que descreva os dados, de acordo com o critério de agrupamento adotado (Jiang et al., 2004). Os critérios de agrupamento podem ser agrupados em três grandes categorias, de acordo com o tipo de característica que eles apresentam (Handl et al., 2005):

- *Compactação*: a compactação ou homogeneidade de um *cluster* é geralmente associada a uma variação intracluster pequena. Algoritmos que otimizam esse tipo de critério tendem a ser muito efetivos na descoberta de *clusters* esféricos e/ou bem separados, mas podem falhar para estruturas mais complexas.

- *Encadeamento ou ligação*: encadeamento é um conceito mais local, baseado na ideia de que objetos vizinhos devem compartilhar o mesmo *cluster*. Esse tipo de critério é bastante apropriado para a detecção de *clusters* de formas arbitrárias, mas não é robusto para os casos em que há pouca separação espacial entre os *clusters*.
- *Separação espacial*: a separação considera as distâncias entre os *clusters* e, por si só, fornece pouca orientação durante o processo de agrupamento, podendo facilmente levar a soluções triviais. Esse conceito é comumente empregado em associação a outros.

O critério de agrupamento, além de representar o principal aspecto de um algoritmo de agrupamento, também está ligado à maioria das alternativas de avaliação dos resultados de um algoritmo de agrupamento. Existe um grande número de algoritmos de agrupamento, cada um buscando *clusters* de acordo com um critério diferente (Law et al., 2004). Algoritmos como o *k*-médias procuram *clusters* compactos, identificando assim mais facilmente *clusters* de forma esférica. Por outro lado, os algoritmos que optimizam um critério baseado no conceito de encadeamento, como o hierárquico de ligação média, captam a densidade local e podem detectar *clusters* de forma arbitrária, mas não são robustos para encontrar *clusters* com uma certa sobreposição. Esses dois algoritmos serão descritos em detalhe no Capítulo 12 e são usados neste capítulo para ilustrar alguns dos aspectos importantes que influenciam na escolha dos algoritmos a serem utilizados.

Mas, independentemente do critério de agrupamento, a análise de agrupamento compreende diversas etapas que vão além da aplicação de um algoritmo de agrupamento em um conjunto de dados. Essas etapas são discutidas na Seção 11.2.

A Figura 11.2 apresenta dois conjuntos de dados. O conjunto de dados **globular** possui dois *clusters* esféricos que não estão bem separados. Já o conjunto **anel** apresenta dois *clusters* na forma de anel, claramente distintos. Algoritmos cujos critérios se baseiam na compactação, como o *k*-médias (Seção 12.2), conseguem identificar claramente a estrutura do conjunto de dados **globular**, mas falham para o conjunto **anel**. Por outro lado, algoritmos baseados no encadeamento, como o algoritmo hierárquico com ligação simples (Seção 12.1), identificam facilmente a estrutura do conjunto **anel**, mas têm problemas com a estrutura de **globular**.

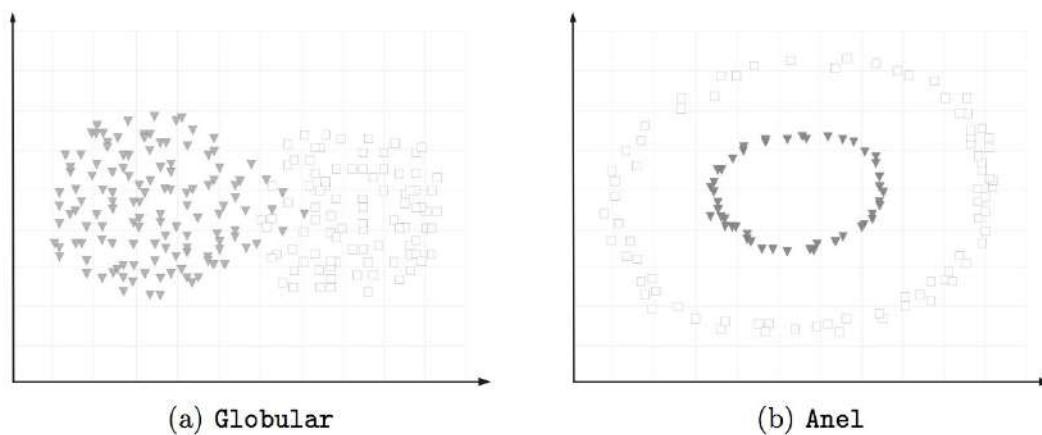


Figura 11.2 Dados com *clusters* em conformidade com diferentes critérios.

Um outro aspecto importante sobre algoritmos de agrupamento é que eles podem encontrar estruturas em diferentes níveis de refinamento (níveis de *clusters* diferentes

ou *clusters* de densidades diferentes), dependendo de suas configurações de parâmetros (Jain e Dubes, 1988). A Figura 11.3 ilustra um conjunto de dados que apresenta duas estruturas claramente distintas em dois níveis de refinamento, ambas em conformidade com o conceito de compactação. Na Figura 11.3(a) observa-se uma estrutura com dois *clusters*, enquanto na Figura 11.3(b) observa-se uma estrutura com seis *clusters*.

Essa questão ressalta a importância do ajuste de parâmetros. Nesse exemplo em particular, o ajuste seria em relação ao número de *clusters*. Mas como decidir qual deles é o mais adequado? A etapa de validação auxilia tanto na escolha do algoritmo mais apropriado como no ajuste de parâmetros (Handl et al., 2005). No entanto, é importante ter em mente que a maioria das técnicas usadas para validação também é tendenciosa para um critério de agrupamento, assim como os algoritmos. Assim, várias medidas de validação diferentes devem ser aplicadas para selecionar os resultados mais consistentes entre os obtidos com uma variedade de algoritmos de agrupamento usando configurações diferentes de parâmetros (Handl et al., 2005). É importante observar que esse processo exige um conhecimento relativamente profundo de análise de agrupamento, o que os especialistas no domínio de dados, aplicando análise de agrupamento, normalmente não têm.

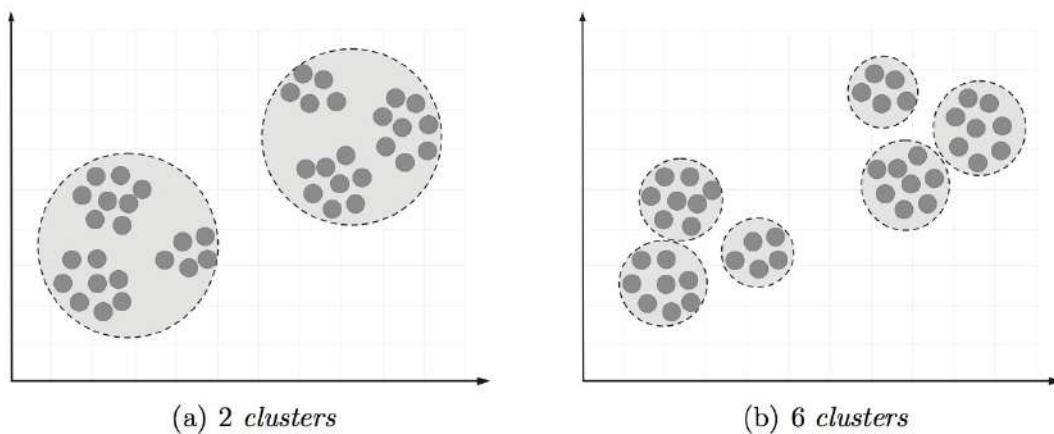


Figura 11.3 Dados com *clusters* em diferentes níveis de refinamento.

Um outro aspecto importante a ser considerado é que cada algoritmo procura por uma estrutura homogênea, ou seja, em que todos os *clusters* estejam em conformidade com o mesmo critério de agrupamento. Entretanto, os dados podem apresentar uma estrutura heterogênea, em que cada *cluster* esteja em conformidade com um critério de agrupamento diferente (Law et al., 2004). Por exemplo, o conjunto de dados ilustrado na Figura 11.4 contém três *clusters*, um dos quais em forma de anel, que é facilmente identificado com um critério baseado no encadeamento, como previamente discutido, e dois com formato globular, facilmente identificados com critérios relacionados à compactação. Nesse caso, um algoritmo como o *k*-médias identificaria os *clusters* globulares, mas não o que possui forma de anel. Por outro lado, o algoritmo hierárquico com ligação simples provavelmente encontraria apenas o *cluster* em anel, misturando os objetos dos dois *clusters* globulares em um único *cluster*. Isso é evidenciado na Figura 11.5, que ilustra o resultado da aplicação dos algoritmos *k*-médias e hierárquico com ligação simples ao conjunto de dados da Figura 11.4. Na verdade, não existe um único algoritmo de agrupamento capaz de encontrar todos os tipos de agrupamentos que podem estar presentes em um conjunto de dados (Estivill-Castro, 2002; Kleinberg, 2002).

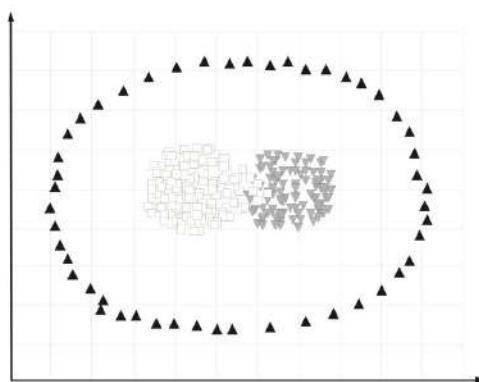


Figura 11.4 Conjunto de dados com uma estrutura de *clusters* heterogênea.

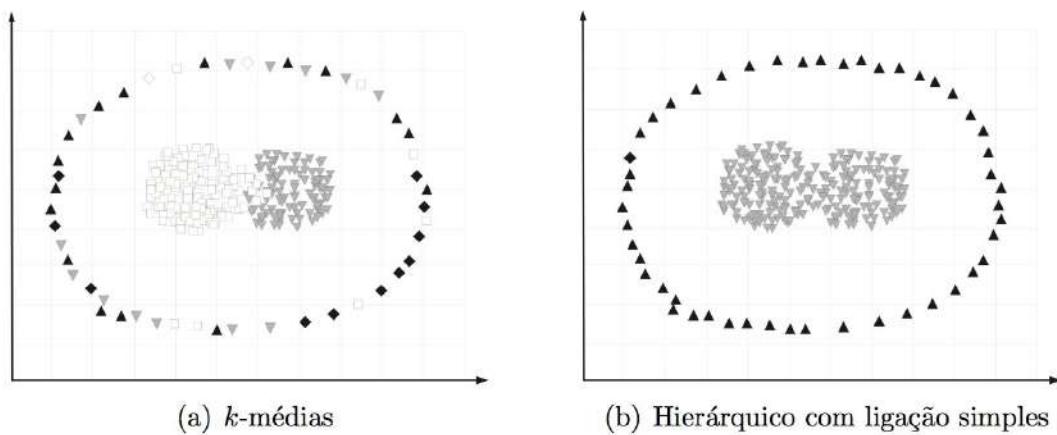


Figura 11.5 Resultados dos algoritmos em dados com estrutura heterogênea.

Além de todos esses aspectos poderem estar simultaneamente presentes em um conjunto de dados, existe ainda a possibilidade de que mais de uma estrutura relevante esteja presente. Em outras palavras, um mesmo conjunto de dados pode ter mais de uma estrutura, cada uma representando uma diferente interpretação dos dados (Handl e Knowles, 2007). Cada uma dessas estruturas pode ser compatível com um critério de agrupamento diferente, estar em um nível de refinamento diferente, ou ainda ser heterogênea. Por exemplo, considere o conjunto de dados mostrado na Figura 11.6. Esse conjunto de dados contém três estruturas distintas: E1 (Figura 11.6(a)), E2 (Figura 11.6(b)) e E3 (Figura 11.6(c)). A estrutura mais simples e evidente é E1. Essa estrutura tem dois *clusters* bem separados e de formato esférico. Em princípio, qualquer algoritmo de agrupamento seria capaz de identificá-la. A estrutura E2 é um refinamento de E1, contendo cinco *clusters*. E3, por sua vez, é um refinamento de E2, com 13 *clusters*. E2 e E3 são altamente heterogêneas, em relação à forma de *clusters*. A aplicação de algoritmos tradicionais para explorar um conjunto de dados como esse concentra-se na descoberta de apenas uma única estrutura que melhor se ajusta aos dados. Nesse caso, a estrutura que seria facilmente encontrada por qualquer algoritmo seria a estrutura E1, que pode ser considerada a mais evidente. As demais estruturas dificilmente seriam encontradas com a aplicação de um único algoritmo. Como a quantidade de conhecimento que pode ser obtido com a aplicação usual da análise de agrupamento é limitada, esse aspecto deve ser considerado. Abordagens mais recentes, como as descritas no Capítulo 13, lidam melhor com várias das questões aqui apresentadas.

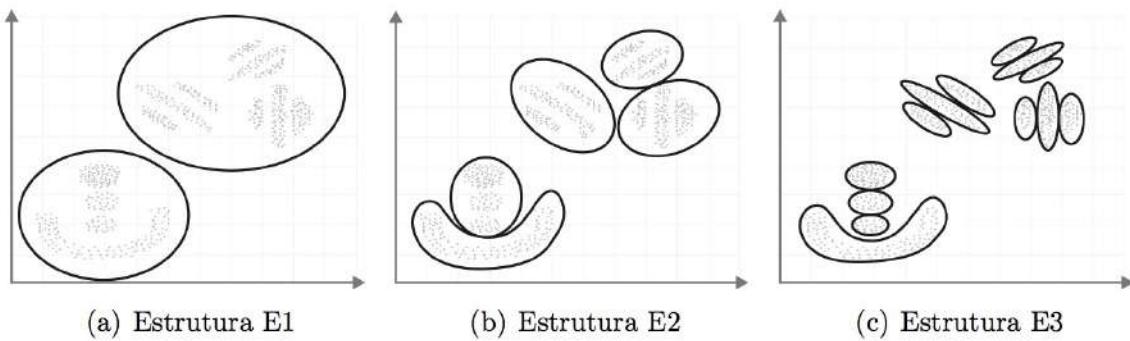


Figura 11.6 Conjunto de dados com várias estruturas heterogêneas.

11.2 Etapas da Análise de Agrupamento

Podemos dividir o processo de agrupamento nas etapas ilustradas na Figura 11.7 que vão desde a preparação dos dados, até a interpretação dos *clusters* obtidos (Jain et al., 1999; Barbara, 2000). Nessa figura, também são ressaltadas as informações utilizadas e geradas em cada etapa. Cada uma dessas etapas será descrita mais detalhadamente nas próximas seções.

Essas etapas são importantes para que se possa garantir que os resultados sejam realmente significativos e úteis, uma vez que qualquer algoritmo de agrupamento obtém um resultado, quer realmente haja uma estrutura subjacente nos dados, em conformidade com o critério do algoritmo, quer não. Assim, para que se possa aplicar com sucesso cada uma das etapas e obter um resultado significativo da análise de agrupamento, é importante ter em mente as possíveis características dos dados, algumas das quais previamente descritas na Seção 11.1, pois tais características impõem certos desafios na aplicação da análise de agrupamento, que, se não considerados, podem levar a conclusões errôneas sobre a estrutura encontrada nos dados.

Considerando, por exemplo, a preparação dos dados, se eles possuem valores dos atributos em escalas diferentes, dependendo da medida usada, um pode dominar o outro no cálculo da proximidade. Quando isso acontece, pode-se tratar a questão com duas alternativas: normalizar os dados (Seção 3.6) ou escolher uma proximidade que não dependa da magnitude dos atributos. Isso ressalta a interdependência entre as etapas de preparação dos dados e escolha da medida de proximidade a ser aplicada, também evidenciada no fluxo dos dados entre essas duas etapas mostrado na Figura 11.7.

Ainda como pode ser observado na Figura 11.7, apesar de ser uma tarefa não supervisionada, o conhecimento do especialista no domínio dos dados pode ser considerado nas várias etapas. Entretanto, é importante ressaltar que ele é usado de maneira diferente do aprendizado supervisionado. Em agrupamento, o conhecimento do especialista ajuda a guiar as escolhas das técnicas aplicadas em cada etapa, e principalmente na interpretação dos *clusters* obtidos na última etapa do processo.

11.2.1 Preparação dos Dados

A preparação dos dados para o agrupamento engloba vários aspectos relacionados ao seu pré-processamento e à forma de representação apropriada para sua utilização por um algoritmo de agrupamento.

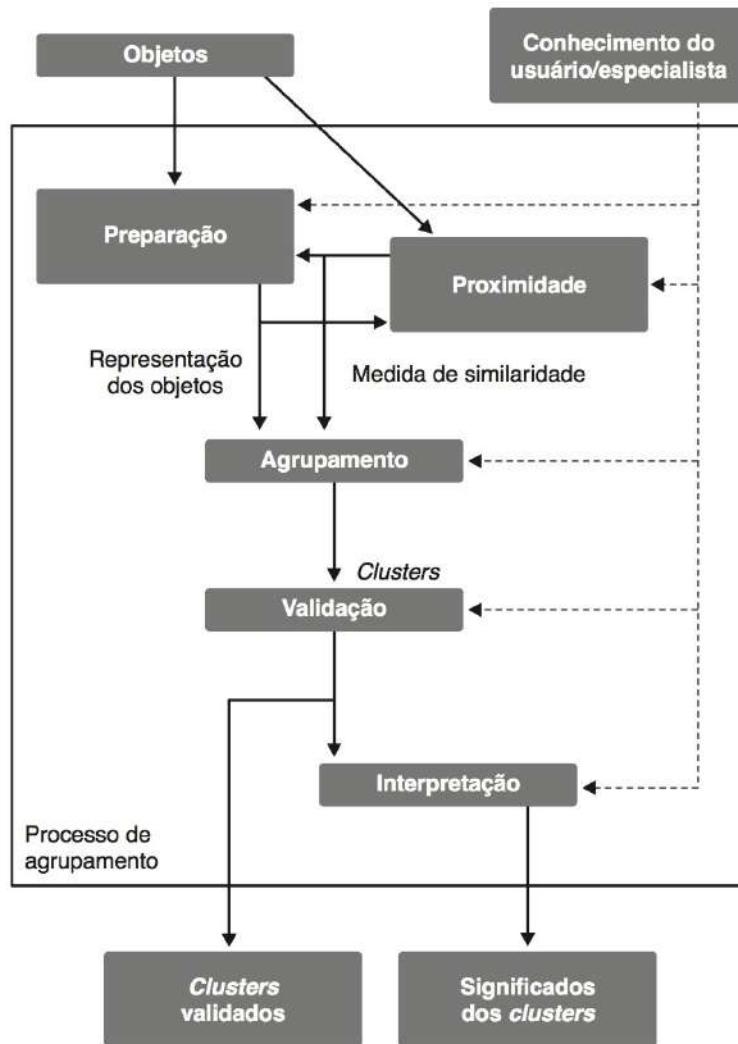


Figura 11.7 Etapas do processo de agrupamento.

O pré-processamento em análise de agrupamento pode incluir normalizações, conversão de tipos e redução do número de atributos por meio de seleção ou extração de características (Jain et al., 1999). Entretanto, é importante ressaltar que na análise de agrupamento não se tem informações a respeito de possíveis classificações dos dados. Assim, várias das técnicas de seleção ou extração de características descritas no Capítulo 3 não se aplicam, ou precisam ser adaptadas para serem utilizadas em agrupamento.

Quanto à representação, na maioria dos casos os objetos a serem agrupados são representados pela matriz de objetos \mathbf{X} , descrita no Capítulo 1. Entretanto, alguns algoritmos de agrupamento exigem uma forma de representação específica, ou algumas vezes apenas a relação de proximidade entre os objetos é conhecida. Assim, além da matriz de objetos, outras duas formas de representação bastante comuns são as matrizes e os grafos de similaridade/dissimilaridade.

A matriz de similaridade/dissimilaridade representa a similaridade ou a dissimilaridade entre cada par de objetos, isto é, cada elemento da matriz $S_{n \times n}$, s_{ij} , é dado pela distância, $d(\mathbf{x}_i, \mathbf{x}_j)$, ou pela similaridade, $s(\mathbf{x}_i, \mathbf{x}_j)$, entre os objetos \mathbf{x}_i e \mathbf{x}_j (Jain e Dubes, 1988). Já para os grafos, existem diversas alternativas, como o diagrama de Delaunay e árvores geradoras mínimas, todas elas representando os objetos de acordo com algum aspecto de proximidade e/ou a topologia dos dados. Considerando a representação em grafos,

realizar um agrupamento é equivalente a quebrar o grafo em componentes conectados, cada um representando um *cluster*. Muitos dos algoritmos de agrupamento são naturalmente descritos usando uma representação de grafo, conforme será visto no Capítulo 12.

11.2.2 Proximidade

Esta etapa consiste na definição de medidas de proximidade apropriadas ao domínio da aplicação e ao tipo de informação que se deseja extrair dos dados. Existem diferentes níveis de proximidade que podem ser considerados em agrupamento: a proximidade entre objetos, a proximidade entre um objeto e um grupo de objetos e a proximidade entre dois grupos de objetos (He, 1999). Todos os algoritmos de agrupamento consideram a similaridade/dissimilaridade entre objetos, e um mesmo algoritmo pode ser implementado considerando medidas diferentes. Por outro lado, as similaridades entre objetos e grupos e entre dois grupos fazem parte da caracterização de cada algoritmo específico, sendo usadas, geralmente, para decidir a atribuição de um objeto a um *cluster* ou para unir/dividir *clusters*, e dependem do tipo de *cluster* que o algoritmo visa identificar. Esse é o caso das métricas de integração usadas nos algoritmos de agrupamento hierárquicos, que serão descritos na Seção 12.1.

A medida de proximidade entre pares de objetos pode ser uma medida de similaridade ou de dissimilaridade entre dois objetos. A escolha da medida de proximidade deve considerar os tipos e escalas dos atributos que definem os objetos, além das propriedades dos dados que se deseja focar. Por exemplo, deve-se ter em mente se a magnitude relativa dos atributos descrevendo dois objetos é suficiente ou se seu valor absoluto deve ser considerado (Gordon, 1999). As medidas de similaridade/dissimilaridade, em geral, consideram que todos os atributos são igualmente importantes, ou seja, todos contribuem da mesma maneira para o cálculo da medida. Considerando a questão da escala dos valores dos atributos, qual seria o efeito na função distância da representação de um atributo em cm e outro em km, por exemplo? As medidas de distância são diretamente afetadas pela escala dos atributos. Para minimizar esse efeito, os atributos são usualmente normalizados, conforme mencionado anteriormente. Esse problema não ocorre, por exemplo, quando todos os atributos dos objetos assumem apenas valores binários.

Uma das medidas de dissimilaridade mais comum para objetos cujos atributos são todos contínuos é a distância euclidiana, enquanto uma das medidas de similaridade mais usadas é a correlação.

Normalmente, as medidas de similaridade/dissimilaridade satisfazem algumas propriedades. Todas as medidas de distância (que quantificam dissimilaridade entre objetos) satisfazem as propriedades 1, 2 e 3 listadas a seguir. Algumas dessas medidas, chamadas métricas, satisfazem também as propriedades 4 e 5. Para as medidas de distância, quanto menor o valor da medida, mais similares são os objetos.

1. $d(\mathbf{x}_i, \mathbf{x}_i) = 0$ para todo \mathbf{x}_i (Os objetos não são diferentes de si próprios)
2. $d(\mathbf{x}_i, \mathbf{x}_j) = d(\mathbf{x}_j, \mathbf{x}_i)$ (Simetria)
3. $d(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ para todo \mathbf{x}_i e \mathbf{x}_j (Positividade)
4. $d(\mathbf{x}_i, \mathbf{x}_j) = 0$ somente se $\mathbf{x}_i = \mathbf{x}_j$
5. $d(\mathbf{x}_i, \mathbf{x}_l) \leq d(\mathbf{x}_i, \mathbf{x}_j) + d(\mathbf{x}_j, \mathbf{x}_l)$ para todo \mathbf{x}_i , \mathbf{x}_j e \mathbf{x}_l (Desigualdade triangular)

Por outro lado, as medidas de similaridade têm uma definição menos rigorosa em relação às propriedades que devem satisfazer.

Gordon (1999) apresenta diversas medidas que são mais apropriadas para objetos cujos atributos são todos de um mesmo tipo. Ele classifica as medidas de acordo com o tipo dos atributos para o qual a medida é apropriada.

Medidas para Atributos Quantitativos

Mesmo quando todos os atributos dos objetos são quantitativos, algumas medidas são mais utilizadas quando os valores são contínuos e racionais, e outras, quando os valores são binários.

As medidas mais utilizadas para atributos contínuos e racionais são as medidas de distância baseadas na métrica de Minkowski, como a distância euclidiana, a distância de Manhattan e a distância *supremum*. Quando todos os atributos são binários, é comum a utilização da distância de Manhattan, que neste contexto é chamada de distância de Hamming.

Para dados binários e nominais existem também diversos coeficientes de casamento (*matching*), como o coeficiente de casamento simples e o coeficiente de Jaccard. A seguir são apresentadas as principais medidas de distância e similaridade para valores quantitativos.

A métrica de Minkowski é definida pela Equação 11.1.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt[p]{\sum_{l=1}^d |x_i^l - x_j^l|^p} \quad (11.1)$$

A escolha de diferentes valores para p , com $1 \leq p < \infty$, define variações da métrica. Por isso, essa métrica também é chamada de distância L_p . Os menores valores de p correspondem a estimativas mais robustas (menos sensíveis a *outliers*). As métricas de Minkowski são sensíveis a variações de escala dos atributos, isto é, atributos representados em uma escala maior tendem a dominar os outros. Isso pode ser solucionado pela normalização dos atributos para um intervalo ou variância comum, ou pela aplicação de outros esquemas de ponderação, como os descritos no Capítulo 3 (Jain et al., 1999).

As principais variações da métrica de Minkowski para diferentes valores de p são dadas pelas Equações 11.2, 11.3 e 11.4. A Figura 11.8 ilustra graficamente o significado dessas métricas quando os objetos possuem duas dimensões.

- $p = 1$: **Distância de Manhattan** (ou distância bloco-cidade), dada pela Equação 11.2.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^d |x_i^l - x_j^l| \quad (11.2)$$

- $p = 2$: **Distância euclidiana**, dada pela Equação 11.3. Essa métrica é a medida de distância mais popular, e uma das mais utilizadas em análise de agrupamentos.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\left(\sum_{l=1}^d (x_i^l - x_j^l)^2 \right)} \quad (11.3)$$

- $p = \infty$: **Distância de Chebyschev ou supremum**, dada pela Equação 11.4, calcula o máximo da diferença absoluta em coordenadas. Em outras palavras, é a diferença máxima entre quaisquer atributos dos objetos.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \max_{1 \leq l \leq d} |x_i^l - x_j^l| \quad (11.4)$$

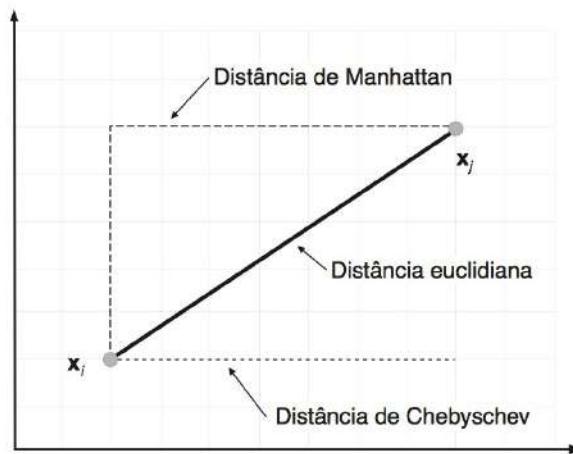


Figura 11.8 Interpretação das métricas de Minkowski para $d = 2$.

Duas formas comumente usadas em agrupamento para avaliar a similaridade entre pares de objetos são dadas pelo valor absoluto das medidas de separação angular e correlação de Pearson, que quantificam a correlação entre os objetos \mathbf{x}_i e \mathbf{x}_j .

A separação angular, ou simplesmente cosseno, é dada pela Equação 11.5.

$$\text{cosseno}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{l=1}^d x_i^l x_j^l}{\sqrt{(\sum_{l=1}^d x_i^{l^2}) (\sum_{l=1}^d x_j^{l^2})}} \quad (11.5)$$

Já a correlação de Pearson é dada pela Equação 11.6, em que $\bar{x}_i = \sum_{l=1}^d x_i^l / d$. Essa medida é frequentemente descrita como uma medida da forma, no sentido de que é insensível a diferenças na magnitude dos atributos, sendo muito usada para determinar a similaridade entre objetos em áreas como Bioinformática, em que apenas o padrão de variação dos atributos dos objetos é importante.

$$\begin{aligned} \text{pearson}(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\text{covariânci}(\mathbf{x}_i, \mathbf{x}_j)}{\text{variânci}(\mathbf{x}_i)\text{variânci}(\mathbf{x}_j)} \\ &= \frac{\sum_{l=1}^d (x_i^l - \bar{x}_i)(x_j^l - \bar{x}_j)}{\sqrt{(\sum_{k=1}^d (x_i^k - \bar{x}_i)^2) (\sum_{l=1}^d (x_j^l - \bar{x}_j)^2)}} \end{aligned} \quad (11.6)$$

Considerando \mathbf{x}_i e \mathbf{x}_j como dois vetores no espaço d -dimensional, a separação angular e a correlação de Pearson podem ser interpretadas geometricamente como o cosseno dos ângulos entre os vetores originais e transformados, respectivamente. A Figura 11.9 ilustra a interpretação quando os objetos são bidimensionais. A separação angular se refere ao cosseno do ângulo α entre os vetores originais, \mathbf{x}_i e \mathbf{x}_j . Já a correlação de Pearson corresponde ao cosseno do ângulo β entre os vetores que correspondem aos objetos transformados de maneira a ter média zero e variância 1, ou seja, $\mathbf{x}'_i = (\mathbf{x}_i - \bar{\mathbf{x}}_i) / \text{variância}(\mathbf{x}_i)$ e $\mathbf{x}'_j = (\mathbf{x}_j - \bar{\mathbf{x}}_j) / \text{variância}(\mathbf{x}_j)$.

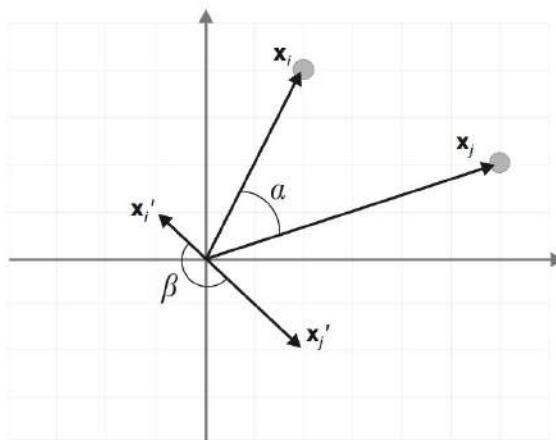


Figura 11.9 Interpretação geométrica da separação angular e da correlação de Pearson para $d = 2$.

A correlação de Pearson é sensível a *outliers* e é menos intuitiva do que métricas como a distância euclidiana. Os valores dessas duas medidas variam no intervalo $[-1, 1]$, e sua magnitude indica a força da correlação, enquanto o sinal indica a direção. Assim, valores tanto próximos de 1 quanto próximos de -1 indicam similaridade entre os objetos (ou seja, os objetos são correlacionados). Enquanto o valor 1 indica que os objetos são diretamente correlacionados, -1 indica que eles são inversamente correlacionados.

Considerando a perspectiva vetorial, uma correlação igual a 1 indica que os vetores representando os objetos são paralelos e apontam no mesmo sentido (ângulo de separação de 0°), uma correlação igual -1 indica vetores paralelos, mas de sentido oposto (ângulo de 180°), e uma correlação igual a 0 indica vetores ortogonais (ângulo de 90°).

Assim, a similaridade entre dois objetos \mathbf{x}_i e \mathbf{x}_j , considerando a separação angular, é dada por $s(\mathbf{x}_i, \mathbf{x}_j) = |\cos(\mathbf{x}_i, \mathbf{x}_j)|$, e, considerando a correlação de Pearson, é dada por $s(\mathbf{x}_i, \mathbf{x}_j) = |\text{pearson}(\mathbf{x}_i, \mathbf{x}_j)|$. O valor 1 indica a maior similaridade entre os objetos. O valor absoluto deve ser usado para considerar tanto os objetos diretamente quanto os inversamente correlacionados como similares (Jain e Dubes, 1988). Dependendo dos dados e da aplicação, pode ser mais apropriado considerar objetos inversamente relacionados como diferentes, caso em que não é necessário considerar o valor absoluto (dois objetos são similares se possuem valor próximo de 1).

Medidas para Atributos Qualitativos

As medidas para esses tipos de atributo são obtidas pela soma das contribuições individuais de todos os atributos.

Para atributos nominais, uma medida de distância muito utilizada é a distância de Hamming, que é ilustrada pela Equação 11.7, em que $a(\mathbf{x}_i, \mathbf{x}_j)$ é dada pela Equação 11.8.

A distância de Hamming conta o número de atributos categóricos com valores diferentes nos dois objetos (Nadler e Smith, 1993). Seu intervalo de variação é $[0, d]$, em que 0 indica a maior similaridade entre os objetos.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{q=1}^d a(\mathbf{x}_i, \mathbf{x}_j) \quad (11.7)$$

$$a(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 & \text{se } x_i^q \neq x_j^q \\ 0 & \text{caso contrário} \end{cases} \quad (11.8)$$

Medidas para Atributos Heterogêneos

Muitos dos conjuntos de dados utilizados em AM apresentam atributos de tipos diferentes, tanto quantitativos quanto qualitativos. Algumas medidas foram propostas para medir a similaridade entre objetos descritos por atributos de diferentes tipos, por se adequarem a qualquer um dos tipos individualmente. Um exemplo de medida que pode ser aplicada a atributos heterogêneos é o coeficiente geral de similaridade, ilustrado pela Equação 11.9, em que s_{ijk} é a contribuição do k -ésimo atributo para a similaridade e w_{ijk} é 0 ou 1, dependendo de se a comparação para o atributo k é válida ou não. A equação utilizada para s_{ijk} pode variar para cada tipo de atributo.

$$s(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^d w_{ijk} s_{ijk} / \sum_{k=1}^d w_{ijk} \quad (11.9)$$

Essa medida é adequada para obter a similaridade entre objetos descritos por atributos de diferentes tipos, por se adequar a qualquer um dos tipos individualmente.

Para o caso em que os dados possuem atributos categóricos e contínuos, pode-se utilizar, por exemplo, uma composição das medidas de distância euclidiana e de Hamming. Uma discussão mais abrangente a respeito de medidas de distância heterogêneas pode ser encontrada em Wilson e Martinez (1997).

11.2.3 Agrupamento

A etapa central de todo o processo envolvido na análise de agrupamento é a etapa de agrupamento, em que um ou mais algoritmos de agrupamento são aplicados aos dados para a identificação das possíveis estruturas de *clusters* existentes nos dados. Os diferentes tipos de estruturas que podem ser encontrados por um algoritmo de agrupamento são, por exemplo, partições, hierarquias de partições e partições *fuzzy*. Tradicionalmente, cada algoritmo de agrupamento busca por uma única estrutura de um desses tipos que melhor se ajuste aos dados. Entretanto, várias abordagens recentes, algumas das quais descritas no Capítulo 13, identificam um conjunto de estruturas presentes nos dados. Nesses casos, cada estrutura representa uma visão diferente desses dados. Por exemplo, cada uma dessas estruturas pode estar de acordo com uma definição diferente de *cluster*.

Duas das principais categorizações dos algoritmos de agrupamento estão relacionadas ao tipo de estrutura que pode ser encontrada: agrupamento exclusivo \times não exclusivo e agrupamento hierárquico \times particional (Jain e Dubes, 1988).

Um agrupamento exclusivo resulta em uma partição do conjunto de objetos. O conceito de partição (da teoria dos conjuntos) é uma divisão do conjunto de objetos em subconjuntos menores (em análise de agrupamento, são denominados *clusters*), com algumas propriedades. Formalmente, dado o conjunto de dados $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, uma partição de \mathbf{X} em k *clusters* pode ser definida como: $\pi = \{\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_k\}$ com $k < n$, tal que (Xu e Wunsch, 2005):

1. $\mathbf{C}_j \neq \emptyset, j = 1, \dots, k$ (todos os *clusters* contêm pelo menos um objeto)
2. $\bigcup_{j=1}^k \mathbf{C}_j = \mathbf{X}$ (todos os objetos pertencem a algum *cluster*)
3. $\mathbf{C}_j \cap \mathbf{C}_l = \emptyset, j, l = 1, \dots, k$ e $j \neq l$ (cada objeto pertence exclusivamente a um único *cluster*)

O agrupamento que resulta nesse tipo de estrutura também costuma ser chamado de *hard* (um objeto pertence ou não pertence a um dado *cluster*). Um agrupamento não exclusivo, por sua vez, pode associar um mesmo objeto a vários *clusters* ou pode associar a cada objeto uma medida que quantifica a sua proximidade com cada um dos *clusters*. Nesse último caso se enquadram os algoritmos *fuzzy* e probabilísticos. Assim, os algoritmos de agrupamento *fuzzy*, por exemplo, são uma forma de agrupamento não exclusivo em que cada objeto tem um grau de pertinência a cada um dos *clusters*. A estrutura encontrada por esse tipo de algoritmo é uma partição *fuzzy*. O primeiro caso, em que um objeto pertence a mais de um *cluster*, é apropriado para lidar com dados que possuam mais de uma estrutura. Um exemplo em que objetos podem ser associados a mais de um grupo é o caso de agrupamento de textos por assunto: um mesmo texto pode pertencer a um *cluster* de textos políticos e a um *cluster* de textos esportivos.

Os algoritmos exclusivos podem ainda ser subdivididos em hierárquicos e particionais. O resultado de um algoritmo particional é uma única partição dos dados, enquanto um agrupamento hierárquico resulta em uma sequência aninhada de partições.

Os algoritmos podem ser categorizados ainda de outras maneiras, considerando outros aspectos. Algumas divisões comuns são aglomerativos × divisivos, seriais × simultâneos e baseados em teoria dos grafos × álgebra matricial.

Alguns dos principais algoritmos de agrupamento tradicionais serão apresentados no Capítulo 12, enquanto modelos múltiplos no contexto de análise de agrupamento serão vistos no Capítulo 13.

11.2.4 Validação

Essa etapa avalia o resultado de um agrupamento e deve, de forma objetiva, determinar se os *clusters* são significativos, ou seja, se a solução é representativa para o conjunto de dados analisado. Além de verificar a validade da solução, pode ajudar, por exemplo, na determinação do número apropriado de *clusters* para um conjunto de dados, que em geral não é conhecido previamente.

Como já mencionado, o problema da maioria das abordagens de agrupamento é que elas podem produzir diferentes agrupamentos a partir de um único conjunto de dados (Zeng et al., 2002). Disso surgem algumas questões. Qual resultado é melhor? Quanto se pode confiar nesse resultado? Existe um resultado que seja melhor do que os outros?

Se existe, como obtê-lo? Se não, é possível combinar todos os resultados disponíveis para ter um entendimento melhor dos dados?

A maioria dos problemas de agrupamento é NP (não determinístico polinomial), o que significa que eles são intratáveis ou não computáveis em um tempo razoável (Zeng et al., 2002). Como já foi dito, todas as abordagens disponíveis são heurísticas e podem fornecer apenas uma aproximação do resultado ótimo (Zeng et al., 2002). Além disso, apesar do grande número de algoritmos de agrupamento existentes, não existe um algoritmo de agrupamento universal, capaz de revelar toda a variedade de estruturas que podem estar presentes em um conjunto de dados. Como lembra Hartigan (1985), “*diferentes agrupamentos são corretos para diferentes propósitos, assim, não podemos dizer que um agrupamento é melhor*”.

A definição da medida de proximidade e do critério de agrupamento utilizados pelos algoritmos geralmente depende implicitamente da imposição de certas hipóteses a respeito da forma dos *clusters* ou da configuração dos múltiplos *clusters*. Outro aspecto importante é que os dados dificilmente estão estruturados “idealmente”, ou seja, geralmente não formam configurações hiperesféricas, hiperelipsoidais, lineares etc., de modo que cada algoritmo de agrupamento pode apresentar um comportamento superior ao dos demais para uma dada conformação específica dos dados no espaço de atributos.

A análise e a comparação de algoritmos de agrupamento são tarefas complexas e que dependem muito do conhecimento, tanto do domínio da aplicação como das técnicas de agrupamento empregadas.

Uma característica importante, inerente à análise de agrupamento e que torna difíceis a análise do desempenho e a comparação dos algoritmos, é a ausência de uma estrutura ideal, que seja a resposta esperada para o agrupamento. Ou seja, como o agrupamento é uma tarefa não supervisionada, não há uma classificação conhecida dos objetos. É importante ter em mente que, quando se faz análise de agrupamento para de fato explorar um determinado conjunto de dados e extrair conhecimento desse conjunto, nada se sabe sobre sua(s) estrutura(s) subjacente(s). Outras questões que tornam difíceis a análise, a escolha e a comparação de algoritmos são o grande número de algoritmos disponíveis e, segundo Estivill-Castro (2002), a falta de descrição explícita dos princípios indutivos e modelos descritos na literatura.

A análise do desempenho de algoritmos de agrupamento ainda é uma área em aberto. Atualmente, tal análise tem sido feita com base em conjuntos de dados que já têm uma estrutura conhecida, com o objetivo de avaliar e comparar os algoritmos existentes e/ou que estão sendo propostos. Apesar de esse tipo de análise ser útil nesses casos, ela não faz sentido na análise exploratória dos dados.

De acordo com a literatura na área, alguns fatores têm grande influência no desempenho das técnicas de agrupamento: a estrutura dos *clusters* (forma, tamanho, número de *clusters*), a presença de *outliers*, o grau de sobreposição dos *clusters* e a escolha da medida de similaridade (He, 1999).

Segundo Jain e Dubes (1988): “*Uma comparação teórica dos algoritmos de agrupamento não é factível porque os algoritmos de agrupamento são quase impossíveis de modelar de tal forma que os modelos possam ser comparáveis*”. Mas alguns critérios são úteis quando se deseja comparar diversos algoritmos de agrupamento. Em primeiro lugar, deve-se ter uma ideia clara do critério de agrupamento no qual se baseia o algoritmo. Também é importante entender como o algoritmo representa os *clusters*, ou seja, como é o modelo gerado. Dado um mesmo contexto (critério de agrupamento e modelo), é possível observar

características específicas dos algoritmos relacionadas às suas habilidades, aos resultados que eles podem produzir, aos dados que eles suportam e à necessidade de interação com o usuário. Entretanto, caso o objetivo seja comparar algoritmos baseados em critérios de tipos diferentes, é preciso saber exatamente o que se deseja comparar e pensar se de fato tal comparação faz sentido, já que o objetivo dos algoritmos pode ser diferente.

As características para se comparar algoritmos em um mesmo contexto são (Halkidi et al., 2001; Jiang et al., 2004):

Relacionadas ao algoritmo:

- Complexidade do algoritmo;
- Escalabilidade e eficiência para conjuntos de dados grandes;
- Medidas de similaridade que podem ser empregadas pelo algoritmo;
- Robustez relativa a ruídos e *outliers*;
- Se o algoritmo é capaz de lidar com dados de alta dimensionalidade ou encontra *clusters* em subespaços do espaço original;
- Se para cada execução diferente do algoritmo os dados são alocados aos mesmos *clusters* (estabilidade);
- Se o algoritmo é capaz de manipular incrementalmente a adição de novos objetos ou a remoção de objetos antigos.

Relacionadas ao resultado:

- Forma dos *clusters* que o algoritmo é capaz de encontrar;
- Interpretabilidade dos resultados.

Relacionadas aos dados:

- Tipos de dados que o algoritmo suporta (contínuos, categóricos, binários);
- Dependência da ordem dos dados.

Relacionadas à interação do usuário:

- Se o algoritmo encontra o número de *clusters* ou se o usuário deve fornecê-lo;
- Parâmetros requeridos pelo algoritmo e o conhecimento do domínio requerido do usuário.

Estivill-Castro (2002) discute algumas questões referentes à falta de descrição explícita dos critérios de agrupamento e modelos em muitos dos algoritmos encontrados na literatura, o que pode gerar confusões sobre as propriedades dos algoritmos e tornar difícil a comparação entre eles. Algumas das observações e recomendações de Estivill-Castro são:

- Os algoritmos de agrupamento são categorizados mais de acordo com os modelos do que com os critérios de agrupamento. Então, é preciso atenção na escolha dos algoritmos e no momento da comparação.

- Os pesquisadores devem tentar explicitar matematicamente os modelos e critérios dos algoritmos de agrupamento que estão propondo, facilitando com isso futuras investigações e comparações.
- Os índices de validação de agrupamento são formulações matemáticas diretas de princípios de indução por trás dos critérios de agrupamento. Comparar algoritmos com base nesses índices pode fornecer algumas dicas sobre os contextos nos quais um algoritmo funciona melhor do que outro, mas isso não implica que um algoritmo produza resultados mais válidos que outro. Dois algoritmos aplicados a um conjunto de dados que não possui estrutura irão ambos produzir resultados inválidos.
- Um algoritmo projetado para um universo de modelos não é adequado para conjuntos de dados que têm uma estrutura representável por uma família de modelos radicalmente diferente. Por exemplo, o algoritmo *k*-médias não pode encontrar *clusters* não convexos.

Em Dubes e Jain (1976), um conjunto de critérios de admissibilidade é utilizado para comparar algoritmos de agrupamento. Esses critérios são baseados na maneira como os *clusters* são formados, na estrutura dos dados e na sensibilidade da técnica de agrupamento a mudanças que não afetem a estrutura dos dados. Além desses critérios de admissibilidade, existem algumas questões importantes a serem consideradas, tais como: como os dados deveriam ser normalizados? qual medida de similaridade é apropriada para uma dada situação? como o conhecimento do domínio deve ser utilizado? e como um grande conjunto de dados pode ser agrupado eficientemente?

Assim, é essencial aos usuários dos algoritmos de agrupamento ter um bom entendimento da técnica que estão utilizando, conhecer detalhes do processo de obtenção dos dados, ter algum conhecimento do domínio e ter claramente definido o propósito do agrupamento que deseja obter, para que o agrupamento mais adequado para o problema em questão possa ser obtido.

O conhecimento a respeito dos dados é importante, por exemplo, para determinar as transformações necessárias aos dados antes do agrupamento e para escolher as medidas de similaridade que fazem sentido para esses dados. O conhecimento do domínio e o do propósito do agrupamento permitem determinar as características mais relevantes, os algoritmos de agrupamento mais apropriados e a forma de validação mais adequada.

Toda avaliação dos resultados dos algoritmos de agrupamento, bem como a comparação entre vários algoritmos, deve considerar as questões mencionadas. De maneira prática, a validação de um agrupamento, em geral, é feita com base em índices estatísticos, que julgam, de uma maneira quantitativa e objetiva, o mérito das estruturas encontradas (Jain e Dubes, 1988). Um índice quantifica alguma informação a respeito da qualidade de uma estrutura encontrada por um algoritmo de agrupamento.

Existem três tipos de critérios que empregam os índices estatísticos para investigar a validade de um agrupamento (Jain e Dubes, 1988): critérios relativos, internos e externos. Os índices baseados em critérios relativos comparam diversos agrupamentos para decidir qual deles é o melhor em algum aspecto. Eles podem ser utilizados para comparar algoritmos de agrupamento ou para determinar o valor mais apropriado para um parâmetro de um algoritmo. Índices empregados em tal critério se baseiam apenas nos dados originais. Apesar de sua grande utilidade, é importante mencionar que esses índices são tendenciosos em relação a algum critério de agrupamento.

Os critérios externos e internos são baseados em testes estatísticos e têm um alto custo computacional (Halkidi et al., 2001). Seu objetivo é medir o quanto o resultado obtido confirma uma hipótese pré-especificada. Nesse caso, são utilizados testes de hipóteses para determinar se uma estrutura obtida é apropriada para os dados. Isso é feito testando se o valor do índice utilizado é extraordinariamente grande ou pequeno. Isso requer o estabelecimento de uma população base ou de referência. Um mesmo índice pode ser utilizado em um critério externo e interno, embora as distribuições de referência do índice sejam diferentes (Jain e Dubes, 1988).

O Capítulo 14 contém uma descrição mais aprofundada do processo de validação de agrupamento, bem como de alguns dos índices mais utilizados.

11.2.5 Interpretação

Refere-se ao processo de examinar cada *cluster* com relação a seus objetos para rotulá-los, descrevendo a natureza do *cluster*. A interpretação de *clusters* é mais que apenas uma descrição. Além de ser uma forma de validação dos *clusters* encontrados e da hipótese inicial, de um modo confirmatório, os *clusters* podem permitir avaliações subjetivas que tenham um significado prático. Ou seja, o especialista pode ter interesse em encontrar diferenças semânticas de acordo com os objetos e valores de seus atributos em cada *cluster*.

Nessa etapa, é fundamental o apoio do especialista do domínio, pois é com o conhecimento a respeito dos dados que é possível identificar significados para os *clusters* e possíveis relações entre eles. Além disso, formas de visualizar os *clusters* obtidos são de grande ajuda por fornecer ao especialista do domínio uma maneira fácil e intuitiva de observar os resultados do agrupamento.

11.3 Considerações Finais

Os algoritmos de agrupamento existentes apresentam diferentes formas de explorar e verificar estruturas presentes em um conjunto de dados. Neste capítulo, foram apresentados as principais definições e os principais aspectos relacionados à análise de agrupamento. Foram destacadas as etapas necessárias para a realização do agrupamento em um conjunto de dados e detalhados alguns aspectos importantes dessas etapas.

Nesse detalhamento, foram incluídas a preparação dos dados, a descrição de várias medidas de similaridade que podem ser empregadas em agrupamento, a discussão de aspectos importantes sobre os algoritmos de agrupamento, a validação de agrupamentos e a análise e comparação de algoritmos de agrupamento. No final, é apresentado como pode ser feita a interpretação dos resultados.

Capítulo 12

Algoritmos de Agrupamentos

Existe uma grande variedade de algoritmos de agrupamento. Cada algoritmo emprega um critério de agrupamento, que impõe uma estrutura aos dados. Se os dados estão em conformidade com as exigências do critério empregado, a estrutura verdadeira de *clusters* pode ser encontrada. Porém, apenas um número pequeno de critérios de agrupamento independentes pode ser entendido sob os pontos de vista matemático e intuitivo. Por isso, muitos dos critérios propostos na literatura são relacionados. Muitas vezes, os mesmos critérios aparecem representados sob diferentes denominações (Jain e Dubes, 1988). Alguns critérios citados por Jain e Dubes (1988) são erro quadrático, ajuste de um modelo de densidade misto (*mixture density model*) aos objetos, estimativa de densidade, conectividade de grafos e vizinhos mais próximos.

Os algoritmos de agrupamento podem ser classificados por meio de diferentes aspectos. Uma das classificações mais frequentes é proposta por Jain et al. (1999) e utilizada por Halkidi et al. (2001), em que os algoritmos são classificados de acordo com o método adotado para definir os *clusters*. Nesse caso, os algoritmos são divididos em algoritmos hierárquicos, particionais, baseados em *grid* e baseados em densidade. Muitos algoritmos se enquadram em mais de uma dessas categorias. Neste capítulo, algumas outras categorias serão também consideradas. Tais categorias não têm necessariamente relação com o método adotado para definir os *clusters*.

Vários dos algoritmos descritos ou mencionados se baseiam na ideia de um objeto representativo que resume as informações contidas no *cluster*. Um elemento representativo bastante usado é o centroide. Seja um *cluster* $\mathbf{C}_k = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n_k}\}$, com n_k objetos, $\bar{\mathbf{x}}^{(k)}$, o centroide do *cluster* \mathbf{C}_k , é dado pela Equação 12.1.

$$\bar{\mathbf{x}}^{(k)} = \frac{1}{n_k} \sum_{\mathbf{x}_i \in \mathbf{C}_k} \mathbf{x}_i \quad (12.1)$$

Neste capítulo serão descritos apenas alguns dos algoritmos mais tradicionais, representantes de cada categoria considerada. Existe uma variedade muito maior de algoritmos, e esse número não para de crescer, com a proposta de novas abordagens. Por razões de espaço, apenas dois dos algoritmos mais conhecidos serão apresentados em detalhes, representantes das categorias dos algoritmos hierárquicos e dos algoritmos particionais baseados em erro quadrático.

As categorias em que foram divididos os algoritmos são: hierárquicos (Seção 12.1), particionais baseados em erro quadrático (Seção 12.2), baseados em densidade (Seção 12.3), baseados em grafo (Seção 12.4), baseados em redes neurais (Seção 12.5) e baseados em *grid* (Seção 12.6), lembrando que os algoritmos podem se enquadrar em mais de uma

categoria. A maioria dos algoritmos nas categorias *grid*, densidade, grafo e redes neurais (auto-organizáveis) são algoritmos particionais, embora também existam alguns que são hierárquicos.

Alguns outros algoritmos, que não se enquadram nessas categorias, são SVC (do inglês, *Support Vector Clustering*) (Ben-Hur et al., 2001), MSVC (do inglês, *Multiple sphere Support Vector Clustering*) (Chiang e Hao, 2003), SNNC (do inglês, *Shared Nearest Neighbor Clustering*) (Ertöz et al., 2002), *Biclustering* (Cheng e Church, 2000), *Plaid Model* (Lazzeroni e Owen, 2002) e CTWC (do inglês, *Coupled Two-Way Clustering*) (Getz et al., 2003). Esses três últimos algoritmos realizam agrupamento simultâneo dos objetos e dos atributos.

12.1 Algoritmos Hierárquicos

Um algoritmo de agrupamento hierárquico gera, a partir de uma matriz de proximidade, uma sequência de partições aninhadas. O agrupamento hierárquico pode ser dividido em duas abordagens: a aglomerativa, que começa com n clusters com um único objeto e forma a sequência de partições agrupando os clusters sucessivamente, e a divisiva, que começa com um cluster com todos os objetos e forma a sequência dividindo os clusters sucessivamente. Um algoritmo hierárquico aglomerativo gera uma sequência de partições de n objetos em k clusters em que o nível 1 apresenta n clusters de um objeto e o nível n apresenta um cluster com todos os objetos. Assim, os dados são agrupados de forma que, se dois objetos são agrupados em algum nível, nos níveis mais altos eles continuam fazendo parte do mesmo grupo, construindo uma hierarquia de clusters (Duda et al., 2001).

Os aspectos positivos do agrupamento hierárquico são a sua flexibilidade com respeito ao nível de granularidade, a fácil utilização de qualquer forma de similaridade ou distância e a possibilidade de utilizar qualquer tipo de atributo. Como aspectos negativos, tem-se o critério de terminação vago e o fato de que a maioria dos algoritmos não melhora os clusters, uma vez construídos, ou seja, uma vez que um cluster foi criado no processo do agrupamento, ele permanece até o final, sem que haja mudanças de seus objetos.

As abordagens mais clássicas de agrupamento hierárquico utilizam métricas de integração (*linkage metrics*). Essas métricas são medidas de distância entre clusters. Esse tipo de agrupamento resulta em clusters de formas convexas próprias e, em geral, possuem complexidade $O(n^2)$. Existem ainda várias implementações de algoritmos hierárquicos que privilegiam funcionalidades mais específicas, como, por exemplo, uma melhor manipulação de *outliers*, a obtenção de clusters de diferentes formas e tamanhos e a escalabilidade para números elevados de exemplos ou atributos.

Uma descrição geral de algoritmos hierárquicos é apresentada a seguir. Outros algoritmos hierárquicos, não descritos neste livro são os algoritmos BIRCH (do inglês, *Balanced Iterative Reducing and Clustering using Hierarchies*) (Zhang et al., 1996), CURE (do inglês, *Clustering Using REpresentatives*) (Guha et al., 1998), CHAMELEON (Karypis et al., 1999), OPTICS (do inglês, *Ordering Points To Identify the Clustering Structure*) (Ankerst et al., 1999) e ROCK (do inglês, *RObust Clustering using linKs*) (Guha et al., 2000).

Os algoritmos baseados nas métricas de integração podem ser divisivos ou aglomerativos e funcionam da seguinte maneira: inicializam um agrupamento como um conjunto de clusters de um elemento (aglomerativo) ou um único cluster com todos os elementos

(divisivo) e iterativamente unem ou dividem os *clusters* mais apropriados, até que seja atingido um critério de parada.

A Figura 12.1 ilustra o processo de união e divisão dos *clusters* nos algoritmos aglomerativos e divisivos, respectivamente. Para agrupar/dividir os *clusters*, cada algoritmo considera uma das alternativas de distância/similaridade entre *clusters* dadas pelas métricas de integração. Cada métrica tem influência direta no funcionamento do algoritmo. Esses algoritmos não têm uma função objetivo global. São baseados em decisões locais. Para as técnicas aglomerativas, o critério de agrupamento é tipicamente agrupar os pares de *clusters* mais próximos, de acordo com a métrica de integração utilizada (Barbara, 2000). Para as técnicas divisivas, o critério é, geralmente, dividir os grupos que possam gerar partições mais diferentes.

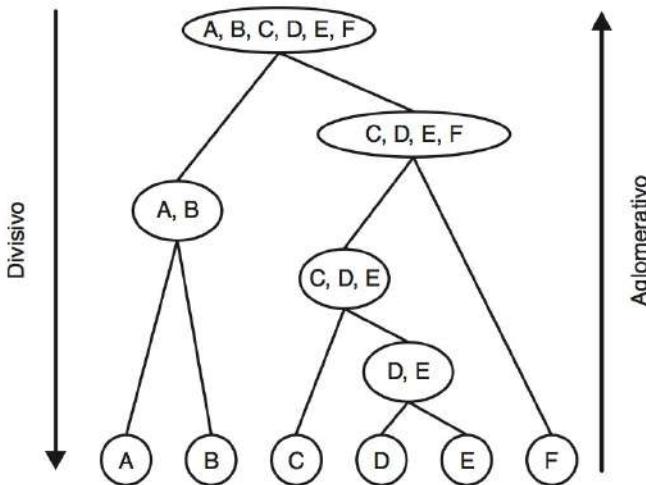


Figura 12.1 Funcionamento dos algoritmos hierárquicos aglomerativos e divisivos.

Dados dois *clusters* $\mathbf{C}_1 = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n_1}\}$ e $\mathbf{C}_2 = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n_2}\}$, com os respectivos centroides $\bar{\mathbf{x}}^{(1)}$ e $\bar{\mathbf{x}}^{(2)}$, para quantificar distâncias entre os *clusters*, podem ser utilizadas distâncias como euclidiana ou de Manhattan entre os centroides dos *clusters*, $d(\bar{\mathbf{x}}^{(1)}, \bar{\mathbf{x}}^{(2)})$, ou ainda quantificar essas distâncias de acordo com as distâncias entre pares de objetos dos dois *clusters*. A ideia, nesse caso, é calcular a distância entre todos os possíveis pares de objetos $(\mathbf{x}_i, \mathbf{x}_j)$, sendo um deles do primeiro *cluster*, \mathbf{C}_1 , e outro do segundo, \mathbf{C}_2 , e, em seguida, aplicar uma operação específica para traduzir essas distâncias na distância entre os *clusters*. Essa operação pode ser, por exemplo, o mínimo, a média ou o máximo dos valores das distâncias entre os objetos. Cada uma das distâncias entre *clusters* estabelecidas dessa maneira é uma métrica de integração.

A distância entre *clusters*, ilustrada pela Equação 12.2, é dada pela distância entre os objetos dos dois *clusters* que estão mais próximos, ou seja, é a distância mínima entre quaisquer dois objetos, um de cada *cluster*. Essa distância é empregada pelo algoritmo hierárquico com ligação mínima (*single-link*). A Equação 12.3 apresenta a distância média entre os objetos dos dois *clusters*. Essa distância é empregada pelo algoritmo hierárquico com ligação média (*average-link*). Por sua vez, a Equação 12.4 descreve a distância entre os objetos mais distantes dos dois *clusters*. Essa distância é empregada pelo algoritmo hierárquico com ligação máxima (*complete-link*).

$$d(\mathbf{C}_1, \mathbf{C}_2) = \min_{\mathbf{x}_i \in \mathbf{C}_1, \mathbf{x}_j \in \mathbf{C}_2} d(\mathbf{x}_i, \mathbf{x}_j) \quad (12.2)$$

$$d(\mathbf{C}_1, \mathbf{C}_2) = \frac{1}{n_1 n_2} \sum_{\substack{\mathbf{x}_i \in \mathbf{C}_1, \\ \mathbf{x}_j \in \mathbf{C}_2}} d(\mathbf{x}_i, \mathbf{x}_j) \quad (12.3)$$

$$d(\mathbf{C}_1, \mathbf{C}_2) = \max_{\substack{\mathbf{x}_i \in \mathbf{C}_1, \\ \mathbf{x}_j \in \mathbf{C}_2}} d(\mathbf{x}_i, \mathbf{x}_j) \quad (12.4)$$

A Figura 12.2 ilustra graficamente o significado das métricas de ligação mínima, máxima e média, bem como a distância euclidiana entre os centroides, entre dois *clusters*.

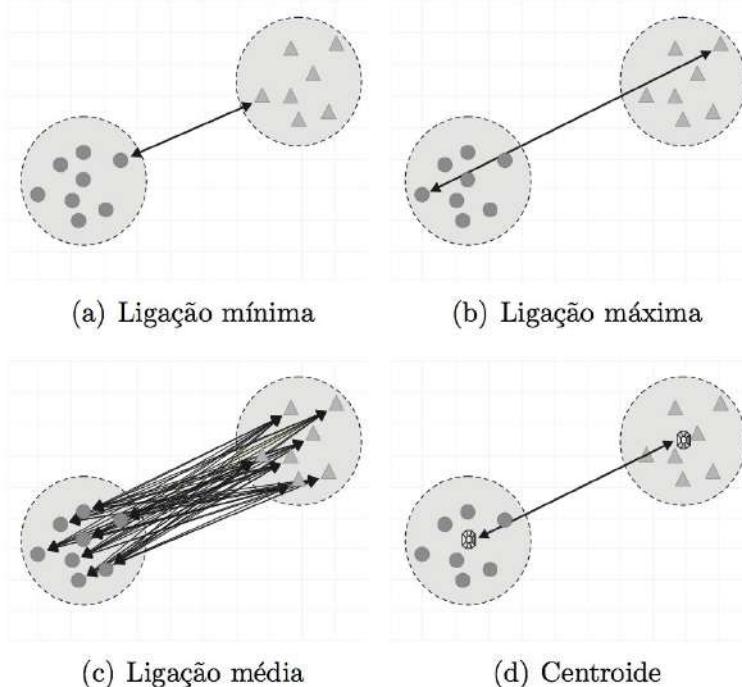


Figura 12.2 Distâncias entre *clusters*.

A métrica utilizada pelo algoritmo com ligação simples é indicada para manipular formas não elípticas, mas é bastante sensível a ruídos e *outliers*. Em geral, essa métrica favorece *clusters* finos e alongados. Já a métrica do algoritmo hierárquico com ligação máxima é menos suscetível a ruídos e *outliers*, mas tende a quebrar *clusters* grandes e tem problemas com formas convexas (Barbara, 2000). Em geral favorece a obtenção de *clusters* esféricos.

Esses algoritmos de agrupamento hierárquico não lidam bem com ruídos e *outliers* e dependem da ordem de entrada dos dados. Por outro lado, não requerem a especificação prévia do número de *clusters*, e seus resultados correspondem a taxonomias, muito comuns nas áreas biológicas.

Quando algoritmos de agrupamento hierárquico são utilizados, as soluções são tipicamente representadas por um dendrograma, que consiste em uma árvore binária que representa uma hierarquia de partições (Barbara, 2000). Essencialmente, um dendrograma é formado por camadas de nós, cada uma representando um *cluster*. Linhas conectam nós representando *clusters* aninhados. O corte de um dendrograma na horizontal representa uma partição.

A Figura 12.3(a) ilustra um dendrograma com a hierarquia de agrupamento para o conjunto de objetos $\{A, B, C, D, E, F\}$. A altura das ramificações, em geral, é propor-

cional à distância dos *clusters* que foram agrupados/divididos. Cortes em cada nível do dendrograma representam diferentes partições dos dados, com diferentes números de *clusters*. A Figura 12.3(b) ilustra cortes no dendrograma representando as partições:

- $\{\{A, B\}, \{C, D, E, F\}\}$, com $k = 2$,
- $\{\{A, B\}, \{C, D, E\}, \{F\}\}$, com $k = 3$ e
- $\{\{A\}, \{B\}, \{C, D, E\}, \{F\}\}$, com $k = 4$.

O Algoritmo 12.1 ilustra o funcionamento dos algoritmos de agrupamento hierárquico aglomerativos baseados em métricas de integração que são mais comumente utilizados. Com algumas modificações, obtém-se o algoritmo divisivo.

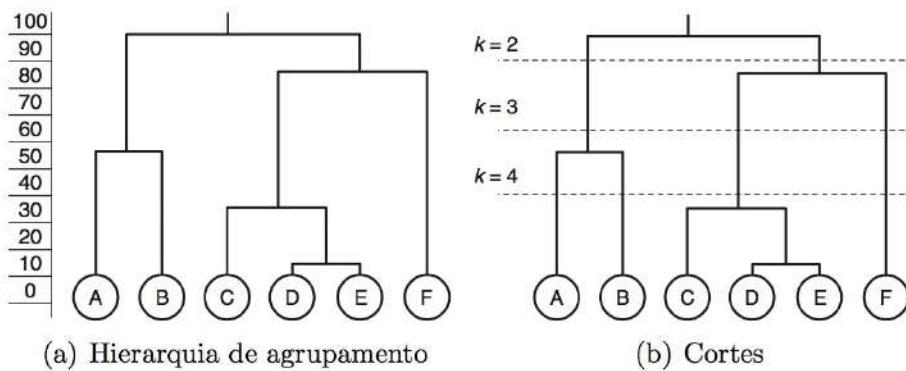


Figura 12.3 Exemplo de dendrograma.

Algoritmo 12.1 Algoritmo hierárquico aglomerativo

Entrada: Uma matriz de dissimilaridade entre pares de objetos $S_{n \times n}$

Saída: Uma hierarquia de partições

- 1 Alocar cada objeto em um *cluster*
 - 2 enquanto há *clusters* para agrupar faça
 - 3 Calcular a matriz de distância entre os pares de *clusters* disponíveis, utilizando uma métrica de integração
 - 4 Combinar o par de *clusters* C_i e C_j mais próximo, gerando um único *cluster* C_{ij}
 - 5 fim
-

12.2 Algoritmos Particionais Baseados em Erro Quadrático

Esses algoritmos otimizam o critério de agrupamento utilizando uma técnica iterativa. O primeiro passo consiste na criação de uma partição inicial. Em seguida, os objetos são movidos de um *cluster* para outro com o objetivo de melhorar o valor do critério de agrupamento. Esses algoritmos são computacionalmente eficientes, porém podem convergir para um ótimo local.

O critério de agrupamento utilizado por esses algoritmos particionais é o erro quadrático, que garante a propriedade de compactação dos *clusters*. O objetivo desses algoritmos

é obter uma partição que minimiza o erro quadrático para um número fixo de *clusters*. Minimizar o erro quadrático, ou a variação dentro de um *cluster*, é equivalente a maximizar a variação entre *clusters* (Jain e Dubes, 1988). O erro quadrático para um agrupamento contendo k *clusters* é a soma da variação dentro dos *clusters*, ilustrada pela Equação 12.5, em que $\bar{\mathbf{x}}^{(j)}$ é o centroide do *cluster* \mathbf{C}_j , como definido na Equação 12.1 e $d(\mathbf{x}_i, \bar{\mathbf{x}}^{(j)})$ é a distância euclidiana entre um objeto \mathbf{x}_i e o centroide $\bar{\mathbf{x}}^{(j)}$. Em alguns casos, a distância de Mahalanobis também pode ser utilizada para definir o erro quadrático. O centroide pode ser a média, como definido na Equação 12.1, ou também a mediana de um grupo de objetos.

$$E = \sum_{j=1}^k \sum_{\mathbf{x}_i \in \mathbf{C}_j} d(\mathbf{x}_i, \bar{\mathbf{x}}^{(j)})^2 \quad (12.5)$$

O objetivo desse tipo de agrupamento é encontrar uma partição contendo k *clusters* que minimiza E , para um valor de k fixo. A partição resultante também é chamada de partição de variância mínima. Minimizar essa função é um problema NP-hard (Jain, 2010). Assim, os algoritmos dessa categoria são gulosos e podem convergir para ótimos locais como já mencionado.

O principal representante dessa categoria é o algoritmo k -médias, que é um dos algoritmos de agrupamento mais simples e conhecidos. Outros algoritmos nessa categoria são: PAM (do inglês, *Partitioning Around Medoids*) (Kaufman e Rousseeuw, 1990), CLARA (do inglês, *Clustering Large Applications*) (Kaufman e Rousseeuw, 1990) e CLARANS (do inglês, *Clustering Large Applications based on the Randomized Search*) (Ng e Han, 1994).

O algoritmo k -médias partitiona o conjunto de dados em k *clusters*, em que o valor de k é fornecido pelo usuário (Duda et al., 2001). Esses *clusters* são formados de acordo com alguma medida de similaridade. O algoritmo k -médias utiliza uma técnica de realocação iterativa, que pode convergir para um ótimo local. Existem várias versões do algoritmo, cada uma solucionando uma deficiência do algoritmo original. A versão tradicional do algoritmo encontra *clusters* compactos e de formato esférico. Mas existem versões, por exemplo, em que a distância de Mahalanobis pode ser utilizada para encontrar *clusters* hiperelipsoidais. Berkhin (2002) e Jain et al. (1999) discutem brevemente algumas dessas versões.

O Algoritmo 12.2 ilustra o funcionamento do algoritmo k -médias básico. Esse algoritmo começa inicializando um conjunto de k centroides para os *clusters*. Essa inicialização pode ser feita de diferentes maneiras. Uma bastante comum é a escolha aleatória de k objetos do conjunto de dados para representar os centroides iniciais. Em seguida, cada ponto do conjunto de dados é associado ao *cluster* com o centroide mais próximo. Depois disso, os centroides são recalculados. O processo é repetido até que os centroides não sejam mais alterados.

Como já mencionado, o critério de agrupamento do k -médias é o erro quadrático definido pela Equação 12.5. Dito de outra maneira, o k -médias minimiza a distância entre cada objeto e o centroide do *cluster* ao qual ele pertence (Halkidi et al., 2001). Essa função objetivo é minimizada por *clusters* de formato globular (hiperesférico) do mesmo tamanho (raio) ou *clusters* bem separados.

A complexidade do algoritmo k -médias é $O(n)$, uma vez que o número de iterações é tipicamente pequeno e $k \ll n$ (Barbara, 2000). Além disso, também é considerado que $d \ll n$.

Algoritmo 12.2 Algoritmo k -médias

Entrada: Um conjunto de dados $\mathbf{X}_{n \times d}$
Número de clusters k
Saída: Uma partição de \mathbf{X} em k clusters

- 1 Escolher aleatoriamente k valores para centroides dos clusters
- 2 repita
 - 3 para cada objeto $\mathbf{x}_i \in \mathbf{X}$ e cluster $\mathbf{C}_j, j = 1, \dots, k$ faça
 - 4 Calcular a distância entre \mathbf{x}_i e o centroide do cluster $\bar{\mathbf{x}}^{(j)}$: $d(\mathbf{x}_i, \bar{\mathbf{x}}^{(j)})$, utilizando uma medida de distância
 - 5 fim
 - 6 para cada objeto \mathbf{x}_i faça
 - 7 Associar \mathbf{x}_i ao cluster com centroide mais próximo
 - 8 fim
 - 9 para cada cluster $\mathbf{C}_j, j = 1, \dots, k$ faça
 - 10 Recalcular o centroide
 - 11 fim
- 12 até não haver mais alteração na associação dos objetos aos clusters;

O algoritmo é sensível à escolha inicial dos centroides e da sua forma de atualização. Dependendo da escolha dos centroides, o algoritmo pode convergir para um ótimo local. Além disso, dependendo da distância empregada, é restrito a objetos em espaços euclidianos. Os clusters encontrados por esse algoritmo são em geral desbalanceados.

12.3 Algoritmos Baseados em Densidade

Esses algoritmos assumem que os clusters são regiões de alta densidade de objetos, separadas por regiões com baixa densidade, no espaço de objetos. Um cluster definido como um componente denso conectado cresce em qualquer direção dada pela densidade (Berkhin, 2002). Portanto, os algoritmos baseados em densidade são capazes de obter clusters de formas arbitrárias.

Além do algoritmo DENCLUE (do inglês, *DENSity-based CLUstering*), descrito a seguir, também podem ser mencionados os algoritmos DBSCAN (do inglês, *Density-Based Spatial Clustering of Applications with Noise*) (Ester et al., 1996) e *Wave-cluster* (também baseado em *grid*) (Sheikholeslami et al., 1998) como baseados em densidade.

O algoritmo DENCLUE modela a densidade global de um conjunto de pontos como a soma de funções “influência” associadas a cada cluster (Hinneburg e Keim, 1998). A função de densidade global resultante tem picos locais que podem ser utilizados para definir clusters. Para cada ponto, encontra-se o pico mais próximo associado a ele. O conjunto de todos os pontos associados a um pico particular (atrator de densidade local) se torna um cluster. Entretanto, se a densidade em um pico local é muito baixa, os pontos associados a esse cluster são considerados ruídos e descartados. Se dois picos locais são conectados por um caminho de pontos e a densidade de cada um desses pontos no caminho está acima de um *threshold* de densidade mínimo ξ , então os clusters associados a esses picos são unidos.

O cálculo da função de densidade global requer a soma das funções influência de todos os pontos. Porém, a maioria dos pontos não contribui para a função de densidade global. Por isso, o algoritmo DENCLUE utiliza uma função de densidade local, que considera apenas os pontos que de fato contribuem para a função de densidade global.

O DENCLUE é baseado em estimativa de densidade por *kernel* (*kernel density estimation*), que tem como objetivo descrever a distribuição dos dados por uma função global. A contribuição de cada ponto para a função de densidade global é expressa por uma função influência ou *kernel*. A função global é a soma das funções associadas a cada ponto.

Para a definição da função influência, é utilizada uma função de distância qualquer, que seja reflexiva e simétrica. Hinneburg e Keim (1998) utilizam a distância euclidiana. Tipicamente, a função influência decresce com o aumento da distância ao ponto. Uma função *kernel* utilizada frequentemente é a função gaussiana (Equação 12.6), em que σ é um parâmetro que governa o quanto rapidamente a influência do ponto diminui.

$$G(x) = \exp^{\frac{-d(\mathbf{x}_i, \mathbf{x}_j)^2}{2\sigma^2}} \quad (12.6)$$

Esse algoritmo possui dois passos: pré-agrupamento e agrupamento. Na etapa de pré-agrupamento, é construído um mapa da porção relevante do espaço de dados para acelerar o cálculo da função de densidade. Na etapa de agrupamento, o algoritmo identifica os atratores de densidade e os pontos atraídos correspondentes.

O mapa é criado dividindo o (hiper-)retângulo de limite mínimo (*minimum bounding hyper-rectangle*) dos dados em hipercubos de dimensão d com aresta de tamanho 2σ . Os hipercubos que contêm pontos de dados são determinados. Em seguida, eles são numerados de acordo com sua posição em relação a uma origem particular. Dessa forma, os hipercubos são mapeados em chaves unidimensionais. As chaves dos cubos povoados são armazenadas em uma árvore de busca para permitir, posteriormente, acesso eficiente.

Para o agrupamento são considerados apenas os cubos mais povoados e os cubos conectados a eles. Para cada ponto \mathbf{x}_i é calculada a função de densidade local considerando apenas os pontos de *clusters* conectados ao *cluster* que contém \mathbf{x}_i e que têm centroides a uma distância de $k\sigma$ de \mathbf{x}_i . Cada ponto \mathbf{x}_j no caminho de \mathbf{x}_i ao seu atrator de densidade é associado ao mesmo *cluster* de \mathbf{x}_i se a distância entre \mathbf{x}_i e \mathbf{x}_j for menor ou igual a $\sigma/2$. Os *clusters* associados a um atrator de densidade cuja densidade seja menor que ξ são descartados. Os atratores de densidade ligados por um caminho de pontos de densidade maior que ξ são unidos.

Esse algoritmo tem uma fundamentação sólida e apresenta uma descrição matemática compacta dos *clusters*. Ele pode também ser classificado como baseado em *grid*. Uma deficiência do DENCLUE é que ele é muito sensível à escolha dos valores dos parâmetros, que são difíceis de determinar. Dependendo da escolha desses valores, ele pode se comportar como os algoritmos DBSCAN, k -médias ou hierárquico.

12.4 Algoritmos Baseados em Grafo

Para a obtenção de um agrupamento por meio de técnicas baseadas na teoria dos grafos, os dados são representados em um grafo de proximidade. No caso mais simples, cada nó representa um objeto e é conectado com os $n - 1$ nós restantes, resultando em um grafo completo. Os pesos das arestas representam a similaridade ou a distância entre os

objetos. Os métodos de agrupamento decompõem os grafos em componentes conectados pela remoção de arestas inconsistentes, ou, ainda, inserem/removem aresta de acordo com algum critério. Cada um desses componentes resultantes do processo de agrupamento vai representar um *cluster*.

Os algoritmos HSC (do inglês, *Highly Connected Subgraph*) (Hartuv e Shamir, 2000) e CLICK (do inglês, *Cluster Identification via Connectivity Kernels*) (Sharan e Shamir, 2000) são exemplos de algoritmos que utilizam uma abordagem baseada na teoria dos grafos para agrupar os dados. Os dados de entrada são representados como um grafo de similaridade.

Os algoritmos HSC e CLICK particionam recursivamente o conjunto atual de elementos em dois subconjuntos. Antes de uma divisão, eles consideram o subgrafo induzido pelo subconjunto atual de elementos. Se o subgrafo satisfaz um critério de parada, então ele é declarado um *kernel*. De outra forma, um corte de peso mínimo é computado naquele subgrafo, e o conjunto é dividido nos dois subconjuntos separados por aquele corte. A saída gerada é uma lista de *kernels* que serve como base para definir os eventuais *clusters*.

A diferença entre os dois algoritmos, HSC e CLICK, está no grafo de similaridade que eles constroem, no critério de parada e no pós-processamento dos *kernels* (Shamir e Sharan, 2002). O algoritmo CLICK é mais recente e, atualmente, mais utilizado.

12.5 Algoritmos Baseados em Redes Neurais

Como apresentado no Capítulo 7, as redes neurais são sistemas paralelos distribuídos compostos de unidades de processamento simples que computam determinadas funções matemáticas, sendo dispostas em uma ou mais camadas e interligadas por um grande número de conexões. Existem diversos modelos de RNs voltados ao aprendizado não supervisionado. Dentre os algoritmos de agrupamento baseados em redes neurais encontram-se os algoritmos SOM (do inglês, *Self Organizing Map*) (Kohonen, 2001), GCS (do inglês, *Growing Cell Structures*) (Fritzke, 1994), SOTA (do inglês, *Self-Organizing Tree Algorithm*) (Herrero et al., 2001), HGSOT (do inglês, *Hierarchically Growing Self-Organizing Tree*) (Luo et al., 2003) e DGSOT (do inglês, *Dynamically Growing Self-Organizing Tree*) (Luo et al., 2004).

O algoritmo SOM (*Self-Organizing Map*) (Kohonen, 2001) é uma rede neural artificial não supervisionada, frequentemente utilizada em tarefas de agrupamento e visualização de dados. É o algoritmo mais tradicional dessa categoria, e será brevemente descrito a seguir.

Nas redes SOM, os neurônios são organizados em um reticulado uni ou bidimensional. Cada neurônio no reticulado está conectado a todas as entradas da rede. Essa rede geralmente utiliza uma única camada computacional. A cada objeto de entrada apresentado à rede, os neurônios computam seus valores de ativação, ativando uma região diferente do reticulado. Para cada objeto de entrada, os neurônios de saída da rede competem entre si para terem seus pesos ajustados. O neurônio com maior valor de ativação é o vencedor da competição. Em seguida, é determinada a localização espacial de uma vizinhança topológica de neurônios ativados, centrada no neurônio vencedor. O próximo passo consiste em uma adaptação dos pesos. Os ajustes dos pesos são tais que o neurônio vencedor e seus vizinhos que tiverem os pesos ajustados aumentam seu valor de ativação para futuros

objetos de entrada que sejam similares ao objeto atual. Assim, durante a execução do algoritmo, os vetores de entrada direcionam o movimento dos vetores de peso, promovendo uma organização topológica dos neurônios da rede. Ainda durante o treinamento, a região de vizinhança dos neurônios vencedores é gradativamente reduzida.

O objetivo da rede SOM é encontrar um conjunto de vetores de referência e associar cada objeto do conjunto de dados ao vetor referência mais próximo. O algoritmo depende da inicialização dos vetores de referência. O resultado consiste em um conjunto de vetores de referência que definem implicitamente os *clusters*.

12.6 Algoritmos Baseados em *Grid*

Esse grupo de algoritmos define um *grid* (reticulado) para o espaço de dados e realiza todas as operações nesse espaço reticulado. Em termos gerais, essa abordagem é muito eficiente para grandes conjuntos de dados, é capaz de encontrar *clusters* de formas arbitrárias e lida bem com *outliers*.

Alguns dos algoritmos que se enquadram nessa categoria são: CLIQUE (do inglês, *Clustering In QUEst*) (Agrawal et al., 1998), descrito a seguir, MAFIA (do inglês, *Merging of Adaptive Finite Intervals*) (Nagesh et al., 2001a,b), OptiGrid (do inglês, *Optimal Grid clustering*) (Hinneburg e Keim, 1999) e STING (do inglês, *Statistical Information Grid-based method*) (Wang et al., 1997).

Alguns desses algoritmos foram projetados com uma funcionalidade em mente. Os algoritmos CLIQUE e MAFIA, por exemplo, são técnicas projetadas especificamente para trabalhar com dados de alta dimensão.

O algoritmo CLIQUE (Agrawal et al., 1998) encontra *clusters* em subespaços dos dados. Esse algoritmo é baseado em *grid* e densidade. Ele identifica *clusters* densos em subespaços de dimensionalidade máxima. Os *clusters* gerados são descritos na forma de expressões FND (Forma Normal Disjuntiva), que são minimizadas para facilitar a compreensão. Os resultados do agrupamento não dependem da ordem de apresentação dos objetos. O algoritmo CLIQUE também não supõe nenhuma forma matemática específica de distribuição dos dados. Além disso, o algoritmo é tolerante a valores ausentes nos dados de entrada.

Como algoritmo baseado em densidade, CLIQUE utiliza a definição de um *cluster* como uma região com densidade maior de pontos do que a região a sua volta. O problema tratado pelo algoritmo é identificar automaticamente projeções dos dados de entrada em um subconjunto dos atributos, com essas projeções incluindo regiões de alta densidade.

Esse algoritmo encontra regiões de alta densidade por meio do particionamento do espaço de dados em células (hiper-retângulos) e da localização das células densas. Um *cluster* corresponde à união de todas as células de alta densidade adjacentes. CLIQUE é baseado na seguinte propriedade dos *clusters*: uma vez que um *cluster* representa uma região densa em algum subespaço do espaço de atributos, haverá áreas densas correspondentes ao *cluster* em todos os subespaços de menor dimensão. Com base nisso, o algoritmo inicia encontrando todas as áreas densas em espaços unidimensionais correspondentes a cada atributo. Em seguida, o algoritmo gera um conjunto de células bidimensionais que podem ser densas. Isso é realizado a partir das células unidimensionais densas. Cada célula bidimensional deve ser associada a um par de células unidimensionais densas. Da mesma forma são construídas células densas para as demais dimensões. Os *clusters* são ob-

tidos encontrando um conjunto maximal de unidades densas em um determinado número de dimensões.

Efetivamente, o algoritmo CLIQUE resulta na seleção de atributos (seleciona vários subespaços) e produz uma visão dos dados em diferentes perspectivas.

12.7 Considerações Finais

Neste capítulo foram descritos de forma sucinta algoritmos representantes das principais categorias de algoritmos de agrupamento encontrados na literatura de AM. As categorias consideradas foram a dos algoritmos hierárquicos (Seção 12.1), particionais baseados em erro quadrático (Seção 12.2), baseados em densidade (Seção 12.3), baseados em grafo (Seção 12.4), baseados em redes neurais (Seção 12.5) e baseados em *grid* (Seção 12.6). Mais especificamente, foram brevemente introduzidos os algoritmos DENCLUE, baseado em densidade, HSC e CLICK, baseados em grafo, SOM, baseado em redes neurais e CLIQUE, baseado em *grid*.

Além disso, foram apresentados maiores detalhes de dois dos algoritmos mais tradicionais e amplamente empregados em análise de agrupamento: os algoritmos hierárquicos aglomerativos baseados em métricas de integração e o k -médias. Nessas descrições, foram apresentados os procedimentos utilizados por esses algoritmos para encontrar *clusters* em conjuntos de dados.

A ideia deste capítulo foi apenas introduzir alguns dos algoritmos mais tradicionais. Vários outros algoritmos foram mencionados, mas este texto não tem a intenção de fornecer uma lista exaustiva dos algoritmos existentes. Como já mencionado, a quantidade de algoritmos existentes é bastante extensa, e novos algoritmos estão constantemente sendo propostos, tanto como variações e melhorias dos algoritmos tradicionais, tanto envolvendo novas abordagens, como as abordagens multiobjetivo e estratégias de combinação, detalhadas nos modelos múltiplos apresentados no Capítulo 13, e também estratégias de agrupamento não exclusivo, de agrupamento simultâneo de objetos e atributos e de agrupamento e subespaços dos atributos (Jain, 2010).

Capítulo 13

Modelos Múltiplos Descritivos

Como já mencionado no Capítulo 11, a análise de agrupamento engloba vários aspectos e tem uma série de dificuldades. Na tentativa de superar as limitações discutidas, diversas abordagens que combinam diferentes agrupamentos, ou consideram distintos critérios de forma combinada, têm sido propostas na literatura. Essas abordagens mostram-se robustas perante diferentes conformações dos dados.

Atualmente, três focos principais são seguidos para o desenvolvimento de abordagens que superem as limitações e dificuldades encontradas na análise de agrupamento tradicional:

- *Ensembles* de agrupamentos (Fred, 2001; Fred e Jain, 2002; Strehl e Ghosh, 2002; Ghosh et al., 2002; Fred e Jain, 2003; Topchy et al., 2003, 2004; Fern e Brodley, 2004; Law et al., 2004; Kuncheva et al., 2006) e
- Agrupamento multiobjetivo (Handl e Knowles, 2004, 2005a,b, 2007)
- *Ensemble* multiobjetivo (Coelho et al., 2010; Faceli et al., 2009)

Da mesma maneira que a abordagem tradicional de análise de agrupamento, os *ensembles* são direcionados à obtenção de uma única estrutura que melhor se ajuste aos dados e necessitam de ajustes de parâmetros. Além disso, a presença de um grande número de partições iniciais de baixa qualidade influencia negativamente o resultado da combinação, uma vez que todas elas são consideradas simultaneamente para gerar a partição consenso. Por sua vez, a abordagem multiobjetivo fornece como resultado final um conjunto de estruturas alternativas e não requer muitos ajustes de parâmetros (Handl e Knowles, 2004). Porém, quanto maior o número de alternativas, mais difícil é sua análise por parte dos especialistas no domínio (Handl e Knowles, 2004). A abordagem *ensemble* multiobjetivo tenta superar as dificuldades e aproveitar os benefícios dos *ensembles* e abordagem multiobjetivo.

Os *ensembles* de agrupamentos atuam por meio da combinação posterior de um conjunto de agrupamentos obtidos previamente, seja com um único algoritmo de agrupamento ou com vários algoritmos complementares (Handl e Knowles, 2004). A Seção 13.1 contém uma descrição dos principais conceitos relacionados aos *ensembles* de agrupamentos, uma comparação entre várias técnicas existentes e uma descrição mais detalhada de algumas delas. Os *ensembles* são mais robustos e fornecem soluções de melhor qualidade que os algoritmos tradicionais de agrupamento, porém não exploram todo o potencial do uso de múltiplos critérios, pois *clusters* que não podem ser detectados por algum dos membros do

ensemble provavelmente não aparecerão na solução final (Handl e Knowles, 2004). Alternativas que tratam dessa limitação incluem as abordagens de agrupamento multiobjetivo, que realizam a otimização direta de vários objetivos (critérios de agrupamento) simultaneamente, detalhada na Seção 13.2, e as técnicas que integram aspectos dos *ensembles* e da otimização de múltiplos objetivos, que serão apresentadas na Seção 13.3.

Existem outros trabalhos que também se nomeiam agrupamento multiobjetivo, como o de Law et al. (2004), mas, apesar de utilizarem vários objetivos (critérios de agrupamento), esses objetivos não são otimizados diretamente. Como nos *ensembles* de agrupamentos, os autores partem de um conjunto de partições iniciais geradas com diferentes algoritmos de agrupamento. Os diferentes algoritmos é que contemplam os diferentes critérios de agrupamento. A partir dessas partições iniciais, Law et al. (2004) selecionam os *clusters* mais adequados para compor a partição final. Apesar de Law et al. (2004) designarem sua técnica como agrupamento multiobjetivo e explicitamente diferenciá-la dos *ensembles*, ela se encaixa mais nos *ensembles* heterogêneos (partições geradas por diferentes algoritmos) do que na otimização direta de diferentes critérios. Nesse trabalho, os autores utilizam o termo *ensembles* apenas para os *ensembles* homogêneos (partições geradas com um mesmo algoritmo).

13.1 *Ensembles* de Agrupamentos

A combinação de estimadores independentes em comitês, ou *ensembles*, é uma técnica comumente empregada em problemas de classificação e regressão para melhorar a precisão de estimadores individuais, aproveitando as características intrínsecas de cada um (LeBlanc e Tibshirani, 1993; Krogh e Vedelsby, 1995; Merz, 1998; Sharkey, 1999; Gama, 1999; Kuncheva, 2004). O Capítulo 8 contém uma descrição detalhada de *ensembles* aplicados a esses problemas. Contudo, a aplicação direta das técnicas de combinação de estimadores aos algoritmos de agrupamento não é possível (Topchy et al., 2003). Os algoritmos de classificação e regressão pertencem ao paradigma de aprendizado supervisionado, em que a resposta correta acompanha cada objeto utilizado na construção do modelo.

Já os algoritmos de agrupamento pertencem ao paradigma de aprendizado não supervisionado. Nesse caso, os dados não possuem uma resposta correta associada, ou seja, não existe informação sobre a “resposta verdadeira” que se espera obter com a aplicação do algoritmo. Essa ausência da “resposta verdadeira” é um dos aspectos que impedem a aplicação direta das técnicas de combinação de estimadores (Topchy et al., 2003). Assim, os *ensembles* de agrupamentos precisam de técnicas novas ou da adaptações das técnicas utilizadas em combinação de estimadores. Por isso, as técnicas utilizadas para combinação em aprendizado supervisionado servem apenas de inspiração para o desenvolvimento de técnicas que permitam a combinação de algoritmos de agrupamento.

Uma definição simplificada do problema de combinação de algoritmos de agrupamento é apresentada por Topchy et al. (2003): dado um conjunto de n^I partições, $\Pi = \{\pi^1, \pi^2, \dots, \pi^{n^I}\}$, de um conjunto de dados \mathbf{X} resultantes de várias aplicações de um ou mais algoritmos de agrupamento, encontrar uma partição final π^F (**partição consenso**) de melhor qualidade do que as partições iniciais, chamadas também de **partições base**.

Essa “melhor qualidade” depende do objetivo que se deseja atingir com a combina-

ção, que pode estar relacionado a robustez (Strehl e Ghosh, 2002; Topchy et al., 2004; Fern e Brodley, 2004), novidade (Topchy et al., 2003; Law et al., 2004), estabilidade e estimação da confidência (Monti et al., 2003; Topchy et al., 2004), computação distribuída, paralelização e escalabilidade (Strehl e Ghosh, 2002; Topchy et al., 2004), reuso de conhecimento (Strehl e Ghosh, 2002; Ghosh et al., 2002; Topchy et al., 2004), consistência (Fred, 2001; Fred e Jain, 2002, 2003) e desempenho e custo (Topchy et al., 2003). Os objetivos mais comuns para a utilização dos *ensembles* são:

- **Robustez:** desenvolver uma forma de agrupamento mais robusta, que tenha um desempenho médio melhor para diversos domínios e conjuntos de dados, de preferência sem a necessidade de muitos ajustes manuais.
- **Novidade:** encontrar uma partição final que não possa ser obtida com nenhum algoritmo, individualmente.
- **Estabilidade:** obter soluções de agrupamento com menor sensibilidade a ruídos, *outliers*, variações de amostragem ou à variabilidade dos algoritmos.

A partição obtida pela combinação das várias partições iniciais deve ser consistente ou concordar de alguma forma com essas partições, não ser sensível a pequenas variações nas partições individuais e estar de acordo com informações externas sobre a estrutura dos dados, se essas informações estiverem disponíveis Fred e Jain (2003).

As principais tarefas, ou desafios, inerentes ao problema de combinação de múltiplos agrupamentos são: geração dos agrupamentos a serem combinados e determinação de uma função consenso para combinar os agrupamentos (Topchy et al., 2004; Kuncheva et al., 2006; Hadjitorov et al., 2006). Assim como existem diferentes objetivos para combinar várias partições, também existem várias maneiras para lidar com essas tarefas. Nas Seções 13.1.1 e 13.1.2 são resumidas as abordagens mais comuns na literatura para gerar a diversidade necessária nas partições iniciais e para encontrar a partição consenso. Na Seção 13.1.3 são detalhadas duas técnicas de *ensemble*.

13.1.1 Geração dos Agrupamentos Iniciais

As partições a serem combinadas devem ser diferentes, de forma a acrescentar informações úteis para a partição final, ou seja, deve haver diversidade no conjunto das partições a serem combinadas. Por isso, ao se buscar uma partição consenso, é preciso ter muito claro o objetivo do agrupamento final e conhecer bem os algoritmos a serem combinados, a fim de garantir que os algoritmos escolhidos possam contribuir para atingir esse objetivo. Assim, de acordo com o objetivo da combinação, deve-se escolher algoritmos de agrupamento, ou formas de aplicação de um algoritmo, que forneçam a diversidade necessária. As formas atualmente empregadas para obter essa diversidade são (Kuncheva, 2004; Kuncheva et al., 2006; Hadjitorov et al., 2006):

- Utilizar diferentes algoritmos convencionais de agrupamento para gerar as partições (Qian e Suen, 2000; Kellam et al., 2001; Strehl e Ghosh, 2002; Zeng et al., 2002; Weingessel et al., 2003).
- Executar várias vezes um mesmo algoritmo convencional, com diferentes inicializações (Fred, 2001; Fred e Jain, 2002, 2003; Frossyniotis et al., 2002; Weingessel et al., 2003; Topchy et al., 2004).

- Empregar algoritmos mais simples do que os convencionais, chamados de algoritmos de agrupamento fracos (Topchy et al., 2003).
- Avaliar diferentes conjuntos de dados, que podem ser referentes aos mesmos objetos, cada um com um subconjunto dos atributos originais (Strehl e Ghosh, 2002), ou com projeções do conjunto original em um espaço de dimensão menor (Fern e Brodley, 2004), ou ainda referentes a subconjuntos de objetos (reamostragem), com todas as características originais (Strehl e Ghosh, 2002; Monti et al., 2003; Fern e Brodley, 2004).

Nos casos em que as partições base são geradas por um mesmo algoritmo, o *ensemble* é dito homogêneo. Nesses casos, dependendo das características de cada algoritmo, deve ser estabelecido um conjunto de condições iniciais com as quais o algoritmo deve ser executado para gerar as partições. Por exemplo, o algoritmo k -médias depende da inicialização dos centroides e tem como parâmetro o valor de k . Assim, para esse algoritmo, pode-se gerar partições com vários valores de k , ou partições com um mesmo valor de k , mas geradas com diferentes inicializações dos centroides.

Já um *ensemble* heterogêneo combina partições geradas com algoritmos diferentes. Neste caso, devem ser considerados algoritmos com características bastante distintas, para que as partições geradas possam apresentar uma grande diversidade.

13.1.2 Determinação da Função Consenso

A utilização de uma função consenso é a forma empregada para encontrar uma partição consenso (partição final gerada pela combinação) a partir de partições iniciais (base). Ela constitui a essência da combinação, pois diz como as partições são combinadas.

Vários aspectos dificultam a definição de uma função consenso (Topchy et al., 2003). A ausência de rótulos nos objetos a serem agrupados faz com que não haja uma correspondência explícita entre os *clusters* das diversas partições. Isso é agravado quando as partições possuem diferentes números de *clusters*, resultando em um problema computacional intratável (problema de correspondência de rótulos¹). De fato, o problema de combinação de agrupamentos é equivalente ao problema de encontrar uma partição mediana em relação às partições dadas, que é um problema comprovadamente NP-completo (Topchy et al., 2003). Assim, como o critério de agrupamento dos algoritmos convencionais, as funções consenso são heurísticas propostas para a resolução do problema formal de obtenção de uma partição consenso. Uma divisão possível das funções consenso é a de Topchy et al. (2004):

Funções Baseadas em Coassociação

A similaridade entre dois objetos pode ser estimada pelo número de *clusters* compartilhados por eles em todas as partições base. As partições são representadas por uma matriz em que essa similaridade é utilizada para representar a força de coassociação entre os objetos. A partição consenso é obtida pela aplicação de um algoritmo de agrupamento qualquer, que seja baseado em similaridade, a essa matriz de coassociação (Kellam et al., 2001; Strehl e Ghosh, 2002; Fred e Jain, 2002, 2003; Monti et al., 2003).

¹Rótulo dado pelo algoritmo de agrupamento para identificar um *cluster*.

Alguns dos problemas dessa abordagem são: falta de uma metodologia para a definição do algoritmo de agrupamento a ser utilizado para a combinação e a baixa confiabilidade da estimativa dos valores de coassociação quando um pequeno número de partições é utilizado.

Funções Baseadas em Grafo/Hipergrafo

Nesse caso, as partições base são representadas por um grafo ou por um hipergrafo, e a participação consenso é encontrada empregando uma técnica de particionamento de grafos ou hipergrafos (Strehl e Ghosh, 2002; Fern e Brodley, 2004).

Funções Baseadas em Informação Mútua

A função consenso é formulada em termos da informação mútua entre os rótulos na participação consenso e os rótulos nas partições iniciais. Strehl e Ghosh (2002) definem a informação mútua normalizada média entre uma participação qualquer e um conjunto de partições iniciais. A participação consenso é dada pelo máximo dessa função, considerando o número de *clusters* desejado. Porém, a dificuldade de otimização dessa função fez com que os autores utilizassem heurísticas baseadas em coassociação e hipergrafo.

Fred e Jain (2003) também definem formalmente uma função consenso baseada em informação mútua, mas resolvem o problema com uma heurística baseada em coassociação.

Topchy et al. (2003) definem uma função consenso baseada na informação mútua generalizada, que é equivalente à variância intracluster em um espaço de rótulos dos *clusters* especialmente transformado. A função é então otimizada com o algoritmo *k*-médias.

Funções Baseadas em Votação

Um mecanismo de votação é utilizado para atribuir os objetos aos *clusters* da participação consenso, dado que o problema de correspondência dos rótulos seja solucionado para todas as partições base. Entretanto, esse problema de correspondência dos rótulos é de difícil solução, sendo às vezes intratável. Porém, pode-se obter uma aproximação heurística de uma rotulação consistente. Todas as partições podem ser rerrotuladas com base em sua melhor concordância com uma participação referência, que pode ser uma das partições base ou um novo agrupamento do conjunto de dados. Esse procedimento é utilizado por Fred (2001) e Weingessel et al. (2003).

Frossyniotis et al. (2002) propõem a construção das partições juntamente com um processo de renumeração dos *clusters* seguido de votação. A partir disso, são estabelecidas relações de vizinhança entre os *clusters*. Essas informações são utilizadas para fundir os *clusters* mais próximos, resultando na participação final.

Além desses tipos de função consenso, Topchy et al. (2004) definem uma função consenso baseada em uma solução para o problema de probabilidade máxima para um modelo misto finito do conjunto de partições iniciais. Esse conjunto de partições é modelado como uma mistura de distribuições multinomiais no espaço dos rótulos dos *clusters*. O problema de probabilidade máxima pode ser resolvido com o algoritmo EM.

A Tabela 13.1 contém um resumo das principais características das abordagens citadas, utilizadas para *ensemble* de agrupamentos. Nessa tabela estão resumidos a forma de representação das partições base, a função consenso, o objetivo da combinação, os algoritmos empregados e a maneira como foram utilizados para gerar diversidade para as partições base.

Tabela 13.1 Comparação das formas de combinação de agrupamentos

Artigo	Representação das partição	Função consenso	Objetivo da combinação	Algoritmos combinados	Diversidade
Kellam et al. (2001)	Matriz de concordância	Os clusters finais são aqueles que possuem os mesmos objetos em todas as partição	Clusters robustos (clusters em que os objetos aparecem junto em todas as partição)	Hierárquico, k -médias, SOM e algoritmos genéticos, com o coeficiente de correlação de Pearson	Vários algoritmos
Fred (2001)	Matriz de coassociação	Votação	Consistência	k -médias	Mesmo algoritmo com diferentes inicializações
Fred e Jain (2002)	Matriz de coassociação	Ligaçāo simples com um novo critério para determinar a partição final	Consistência	k -médias	Mesmo algoritmo com diferentes inicializações
Srehl e Ghosh (2002)	Hipergrafo	Particionamento de grafo de similaridade, particionamento de corte mínimo e metacusters	Reuso de Conhecimento, Computação distribuída e Robustez	Particionamento de grafo e k -médias, com várias similaridades SOM e particionamento de hipergrafo	Vários algoritmos e mesmo algoritmo com dados diferentes
Frossyntis et al. (2002)	Tabela de votação e tabela de relação de vizinhanga	Votação	Robustez e estabilidade	k -médias e greedy-EM	Mesmo algoritmo com diferentes inicializações
Monti et al. (2003)	Matriz consenso	LM determinando k com base na estabilidade dos agrupamentos	Estabilidade	LM e SOM	Mesmo algoritmo com dados diferentes
Fred e Jain (2003)	Matriz de coassociação, usando votação	Ligaçāo simples (pode ser qualquer função baseada em similaridade)	Consistência, estabilidade e robustez	k -médias	Mesmo algoritmo com diferentes inicializações
Weingessel et al. (2003)	Conjunto de matrizes de pertinência das partição iniciais	Votação/fusão	Robustez	k -médias, hard competitive learning e aprendizado competitivo fuzzy não supervisionado	Vários algoritmos e mesmo algoritmo com diferentes inicializações
Topchy et al. (2003)	Novo conjunto de características dos padrões	Baseada no k -médias aplicado no novo espaço de características	Desempenho e custo	Algoritmos fracos que usam projeções ou divisões aleatórias dos dados	Mesmo algoritmo com diferentes inicializações
Topchy et al. (2004)	Novo conjunto de características dos objetos	Probabilidade máxima encontrada com o método EM	Robustez, estabilidade e reuso do conhecimento	k -médias	Mesmo algoritmo com diferentes inicializações
Fern e Brodley (2004)	Grafo	Particionamento de grafo	Robustez	k -médias	Mesmo algoritmo com dados diferentes
Law et al. (2004)	Conjunto com todos os clusters	Clusters mais estáveis	Novidade e robustez	k -médias, EM, hierárquico com ligação simples e spectral clustering	Vários algoritmos

13.1.3 Algumas Técnicas Ilustrativas

As abordagens para construção de *ensembles* de agrupamentos baseadas em grafos podem ser mais robustas que as técnicas baseadas em matriz de coassociação (Topchy et al., 2005). Uma das técnicas de *ensembles* de agrupamentos mais populares é o algoritmo MCLA (do inglês *Meta-Clustering Algorithm*), proposta por Strehl e Ghosh (2002). Outra abordagem para *ensemble* baseada em particionamento de grafos é o algoritmo HBGF (do inglês *Hybrid Bipartite Graph Formulation*), proposto por Fern e Brodley (2004). Nas duas abordagens, não são necessários os atributos originais dos objetos, apenas os rótulos dos *clusters* de cada objeto nas partições a serem combinadas. Essas duas abordagens são detalhadas a seguir.

Ensemble de Strehl e Ghosh

Strehl e Ghosh (2002) definem formalmente a combinação de algoritmos de agrupamento como um problema de otimização de uma função consenso baseada na informação mútua, compartilhada entre as soluções individuais. Entretanto, como a otimização dessa função é um problema combinatorial difícil, eles propõem três algoritmos de combinação baseados em heurísticas: CSPA (do inglês *Cluster-based Similarity Partitioning Algorithm*), HGPA (do inglês *HiperGraph-Partitioning Algorithm*) e MCLA (do inglês *Meta-Clustering Algorithm*). Como esses algoritmos têm um custo computacional baixo, Strehl e Ghosh (2002) aplicam os três algoritmos, cada um gerando uma partição consenso, e utilizam uma função supraconsenso baseada na informação mútua para escolher qual delas será a partição consenso final. Assim, a partição consenso final será aquela, dentre as três, que tem a melhor informação mútua compartilhada.

A definição formal do problema como de otimização de uma função consenso baseada na informação mútua compartilhada entre as soluções individuais é feita da seguinte maneira. Seja a informação mútua normalizada (*NMI*, do inglês *Normalized Mutual Information*) estimada entre duas partições π^a e π^b , dada pela Equação 13.1, em que $|\cdot|$ indica o número de objetos de um conjunto, k^a e k^b são os números de *clusters* das partições π^a e π^b , respectivamente, \mathbf{C}_h^a é o h -ésimo *cluster* de π^a , \mathbf{C}_l^b é o l -ésimo *cluster* de π^b e n é o número de objetos do conjunto de dados (Strehl e Ghosh, 2002).

$$\phi^{(NMI)}(\pi^a, \pi^b) = -\frac{\sum_{h=1}^{k^a} \sum_{l=1}^{k^b} |\mathbf{C}_h^a \cap \mathbf{C}_l^b| \log\left(\frac{n|\mathbf{C}_h^a \cap \mathbf{C}_l^b|}{|\mathbf{C}_h^a||\mathbf{C}_l^b|}\right)}{\sqrt{\left(\sum_{h=1}^{k^a} |\mathbf{C}_h^a| \log\left(\frac{|\mathbf{C}_h^a|}{n}\right)\right) \left(\sum_{l=1}^{k^b} |\mathbf{C}_l^b| \log\left(\frac{|\mathbf{C}_l^b|}{n}\right)\right)}} \quad (13.1)$$

Com base nessa medida de informação mútua entre duas partições, Strehl e Ghosh (2002) definem uma medida de informação mútua entre uma partição π^i e um conjunto Π de r partições, como a informação mútua normalizada média (*ANMI*), dada pela Equação 13.2.

$$\phi^{(ANMI)}(\pi^i, \Pi) = \frac{1}{r} \sum_{q=1}^r \phi^{(NMI)}(\pi^i, \pi^q) \quad (13.2)$$

em que $\phi^{(ANMI)}$ é a função objetivo, e a partição consenso $\pi^{F(k-opt)}$ é aquela com infor-

mação mútua normalizada média ($\phi^{(ANMI)}$) máxima, em relação às partições individuais em Π , dado que o número de *clusters* desejado para a partição consenso é k . $\pi^{F(k-opt)}$ é dada pela Equação 13.3, em que π^i corresponde a todas as possíveis partições com k *clusters*.

$$\pi^F = \arg \max_{\pi^i} \sum_{q=1}^r \phi^{(NMI)}(\pi^i, \pi^q) \quad (13.3)$$

Como já mencionado, a $\phi^{(ANMI)}$ não é otimizada. Os três algoritmos baseados em heurísticas são executados, e a $\phi^{(ANMI)}$ é utilizada para selecionar a melhor partição dentre as geradas com eles.

Os três algoritmos, baseados em heurísticas e propostos para encontrar uma partição consenso, partem de uma representação inicial das partições na forma de um hipergrafo. No CSPA, o problema não depende dessa representação, mas ela é utilizada para calcular facilmente uma matriz de similaridade que servirá como entrada para um algoritmo de agrupamento. No HGPA o hipergrafo é empregado diretamente. Já no MCLA, ele é utilizado para a construção de um metagrafo a ser particionado e posterior determinação da partição consenso, como descrito mais adiante.

O hipergrafo utilizado é representado por uma matriz de adjacências. Cada coluna da matriz é uma hiperaresta, que representa um *cluster*. Essa matriz é construída pela concatenação das matrizes binárias de pertinência de cada partição. As linhas da matriz de pertinência de uma partição correspondem aos objetos, e as colunas correspondem aos seus *clusters*. Cada célula da matriz contém o valor 1 se o objeto pertence ao *cluster* e 0 em caso contrário.

O algoritmo CSPA utiliza a heurística mais simples, porém possui complexidade quadrática no número de objetos. A função consenso resultante desse algoritmo está classificada entre as funções baseadas em coassociação. Como já mencionado, esse algoritmo se baseia na construção de uma nova matriz de similaridade a partir das partições originais. As entradas dessa matriz denotam a fração das partições nas quais dois objetos pertencem ao mesmo *cluster*. A matriz de similaridade gerada é então utilizada para reagrupar os objetos por meio de um algoritmo de agrupamento qualquer, que seja baseado em similaridade. Strehl e Ghosh (2002) utilizam o algoritmo METIS² (Karypis e Kumar, 1999) para particionar o grafo de similaridade induzido.

No algoritmo HGPA, a combinação é tratada como um problema de particionamento de um hipergrafo definido apropriadamente, no qual as hiperarestas representam *clusters*. Esse particionamento é feito cortando um número mínimo de hiperarestas. Para isso é utilizado o pacote de particionamento de hipergrafos HMETIS.³

O algoritmo MCLA trata a combinação como um problema de correspondência dos *clusters* das partições iniciais. Ele se baseia no agrupamento desses *clusters*. Um metagrafo em que cada vértice corresponde a um *cluster* é construído. Em seguida, o metagrafo é particionado de maneira que os *clusters* que permaneceram em um mesmo grupo (metacluster) sejam correspondentes. Os objetos são então atribuídos aos metaclusters com os quais eles estão mais fortemente associados.

Em experimentos controlados realizados por Strehl e Ghosh (2002) para comparar os três algoritmos, foi observado que o MCLA apresentou melhor resultado na presença de

²<http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>

³<http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview>

uma quantidade média para alta de ruído, o que ocorre com frequência em problemas reais. Em relação à complexidade, o algoritmo MCLA também é o mais vantajoso. Em outros experimentos, Strehl e Ghosh (2002) observaram que cada algoritmo tem um desempenho melhor para uma situação diferente. O algoritmo MCLA é melhor quando há menos diversidade nas partições iniciais, o que está de acordo com a suposição inicial do MCLA de que existe uma correspondência entre os *clusters* das partições a serem combinadas. O MCLA, apresentado no Algoritmo 13.1, será descrito em mais detalhes a seguir. Como nesse algoritmo não é garantido que todo metacluster tenha pelo menos um objeto, a partição π^F tem no máximo (e não exatamente) k *clusters*.

Algoritmo 13.1 Algoritmo MCLA

Entrada: Um conjunto de partições base $\Pi = \{\pi^1, \pi^2, \dots, \pi^{n^I}\}$

Um conjunto de dados $\mathbf{X}_{n \times d}$

Número máximo de *clusters* de π^F , k

Saída: Uma partição consenso π^F

- 1 Construir um metagrafo $G = (V, W)$:
 - 2 $V \leftarrow$ vértices representando os *clusters* de todas as partições do conjunto de partições base Π // as hiperarestas do hipergrafo descrito
 - 3 **para cada** aresta ligando os vértices i e $j \in V$ **faça**
 - 4 $w(i, j) \leftarrow |\mathbf{C}_i \cap \mathbf{C}_j| / |\mathbf{C}_i \cup \mathbf{C}_j|$ // razão do número de objetos na interseção e na união dos *clusters* \mathbf{C}_i e \mathbf{C}_j , pertencentes às partições em Π
 - 5 **fim**
 - 6 Agrupar as hiperarestas (*clusters*), particionando o metagrafo em k metaclusters balanceados // Cada metacluster resultante do particionamento representa um grupo de *clusters* correspondentes
 - 7 Unir os *clusters* de cada metacluster:
 - 8 **para cada** metacluster \mathbf{C}_i^M **faça**
 - 9 Transformar as hiperarestas em uma única meta-hiperaresta
 - 10 Calcular um vetor de associação descrevendo o nível de associação de cada objeto com o metacluster // O vetor de associação é obtido pelo cálculo da média dos vetores que representam as hiperarestas de um metacluster. Assim, o nível de associação de um objeto a um metacluster é dado pela média do número de *clusters* desse metacluster, que contém o objeto.
 - 11 **fim**
 - 12 Determinar o metacluster final de cada objeto:
 - 13 **para cada** objeto x **faça**
 - 14 Associar x ao metacluster para o qual ele possui o maior valor de associação // Desempates são decididos aleatoriamente
 - 15 **fim**
 - 16 $\pi^F \leftarrow$ partição dos objetos indicada pelos metaclusters
-

Para o particionamento do metagrafo mencionado na linha 6 do Algoritmo 13.1, o

pacote de particionamento de grafos METIS⁴ (Karypis e Kumar, 1999) pode ser utilizado. Essa fase de particionamento permite encontrar os *clusters* das partições iniciais que são correspondentes.

Quanto à diversidade das partições iniciais, dependendo do cenário de aplicação considerado, foi definida uma alternativa diferente para gerar as partições iniciais. No primeiro cenário, as partições originais foram formadas com a aplicação de um único algoritmo de agrupamento a diferentes subconjuntos de atributos dos dados. No segundo cenário, as partições foram obtidas também com um único algoritmo, porém aplicado a diferentes subconjuntos de objetos, considerando sempre todos os atributos. Finalmente, no terceiro cenário, as partições iniciais foram geradas a partir da execução de diferentes algoritmos, empregando diferentes medidas de proximidade ao mesmo conjunto de dados.

Para ilustrar o funcionamento do algoritmo MCLA, considere os agrupamentos mostrados na Tabela 13.2 (Strehl e Ghosh, 2002). O hipergrafo que representa esses agrupamentos pode ser observado na matriz de adjacências da Tabela 13.3. Nessa tabela, cada objeto \mathbf{x}_i corresponde a um vértice e cada hiperaresta h_j representa um dos *clusters* de um dos agrupamentos. Com essas informações é construído o metagrafo em que cada hiperaresta é um vértice e cujos pesos das arestas entre os vértices podem ser observados na Tabela 13.4. O particionamento desse grafo em três partes resulta nos metaclusters $\mathbf{C}_1^M = \{h_3, h_4, h_9\}$, $\mathbf{C}_2^M = \{h_2, h_6, h_8, h_{10}\}$ e $\mathbf{C}_3^M = \{h_1, h_5, h_7, h_{11}\}$. Assim, os *clusters* representados por h_3 , h_4 e h_9 , por exemplo, são correspondentes.

Tabela 13.2 Exemplo do MCLA - partições

Partição	Clusters
π^1	$\mathbf{C}_1^1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, $\mathbf{C}_2^1 = \{\mathbf{x}_4, \mathbf{x}_5\}$, $\mathbf{C}_3^1 = \{\mathbf{x}_6, \mathbf{x}_7\}$
π^2	$\mathbf{C}_1^2 = \{\mathbf{x}_6, \mathbf{x}_7\}$, $\mathbf{C}_2^2 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, $\mathbf{C}_3^2 = \{\mathbf{x}_4, \mathbf{x}_5\}$
π^3	$\mathbf{C}_1^3 = \{\mathbf{x}_1, \mathbf{x}_2\}$, $\mathbf{C}_2^3 = \{\mathbf{x}_3, \mathbf{x}_4\}$, $\mathbf{C}_3^3 = \{\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7\}$
π^4	$\mathbf{C}_1^4 = \{\mathbf{x}_1, \mathbf{x}_4\}$, $\mathbf{C}_2^4 = \{\mathbf{x}_2, \mathbf{x}_5\}$, objetos \mathbf{x}_3 , \mathbf{x}_6 e \mathbf{x}_7 não agrupados

Tabela 13.3 Exemplo do MCLA - hipergrafo

Vértices	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}
	\mathbf{C}_1^1	\mathbf{C}_2^1	\mathbf{C}_3^1	\mathbf{C}_1^2	\mathbf{C}_2^2	\mathbf{C}_3^2	\mathbf{C}_1^3	\mathbf{C}_2^3	\mathbf{C}_3^3	\mathbf{C}_1^4	\mathbf{C}_2^4
\mathbf{x}_1	1	0	0	0	1	0	1	0	0	1	0
\mathbf{x}_2	1	0	0	0	1	0	1	0	0	0	1
\mathbf{x}_3	1	0	0	0	1	0	0	1	0	0	0
\mathbf{x}_4	0	1	0	0	0	1	0	1	0	1	0
\mathbf{x}_5	0	1	0	0	0	1	0	0	1	0	1
\mathbf{x}_6	0	0	1	1	0	0	0	0	1	0	0
\mathbf{x}_7	0	0	1	1	0	0	0	0	1	0	0

Em seguida, os *clusters* de cada metacluster são unidos, formando as meta-hiperarestas h_i^M representadas na Tabela 13.5. Nessa tabela também estão representados os respectivos

⁴<http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>

Tabela 13.4 Exemplo do MCLA - pesos

Vértices	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}
h_1	-	-	-	0,00	0,50	0,00	0,40	0,20	0,00	0,20	0,20
h_2	-	-	-	0,00	0,00	0,50	0,00	0,25	0,20	0,25	0,25
h_3	-	-	-	0,50	0,00	0,00	0,00	0,00	0,40	0,00	0,00
h_4	0,00	0,00	0,50	-	-	-	0,00	0,00	0,40	0,00	0,00
h_5	0,50	0,00	0,00	-	-	-	0,40	0,20	0,00	0,20	0,20
h_6	0,00	0,50	0,00	-	-	-	0,00	0,25	0,20	0,25	0,25
h_7	0,40	0,00	0,00	0,00	0,40	0,00	-	-	-	0,25	0,25
h_8	0,20	0,25	0,00	0	0,20	0,25	-	-	-	0,25	0,00
h_9	0,00	0,20	0,40	0,40	0,00	0,20	-	-	-	0,00	0,20
h_{10}	0,20	0,25	0,00	0,00	0,20	0,25	0,25	0,25	0,00	-	-
h_{11}	0,20	0,25	0,00	0,00	0,20	0,25	0,25	0,00	0,20	-	-

vetores de associação $a(h_i^M)$. Com base nessas informações, o metacluster final de cada objeto é determinado. Assim, \mathbf{x}_1 , por exemplo, vai pertencer ao metacluster \mathbf{C}_3^M , pois é o metacluster com o qual \mathbf{x}_1 tem o maior valor de associação (0,75). A partição consenso π^F fica composta pelos clusters $\mathbf{C}_1^F = \{\mathbf{x}_6, \mathbf{x}_7\}$, $\mathbf{C}_2^F = \{\mathbf{x}_4, \mathbf{x}_5\}$ e $\mathbf{C}_3^F = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$.

Tabela 13.5 Exemplo do MCLA - meta-hiperarestas e vetores de associação

Vértices	$\mathbf{C}_1^M = \{h_3, h_4, h_9\}$		$\mathbf{C}_2^M = \{h_2, h_6, h_8, h_{10}\}$		$\mathbf{C}_3^M = \{h_1, h_5, h_7, h_{11}\}$	
	h_1^M	$a(h_1^M)$	h_2^M	$a(h_2^M)$	h_3^M	$a(h_3^M)$
\mathbf{x}_1	0,00	0,00	1,00	0,25	1,00	0,75
\mathbf{x}_2	0,00	0,00	0,00	0,00	1,00	1,00
\mathbf{x}_3	0,00	0,00	1,00	0,25	1,00	0,50
\mathbf{x}_4	0,00	0,00	1,00	1,00	0,00	0,00
\mathbf{x}_5	1,00	0,33	1,00	0,50	1,00	0,25
\mathbf{x}_6	1,00	1,00	0,00	0,00	0,00	0,00
\mathbf{x}_7	1,00	1,00	0,00	0,00	0,00	0,00

Ensemble de Fern e Brodley

A abordagem de Fern e Brodley (2004) utiliza particionamento de um grafo bipartido para, dado um conjunto de partições base, encontrar uma partição consenso. Nessa abordagem, os autores constroem um grafo bipartido a partir do conjunto de partições a serem combinadas. Para isso, modelam simultaneamente tanto objetos quanto clusters como vértices do grafo e, posteriormente, particionam o grafo com uma técnica tradicional de particionamento de grafos. O algoritmo de Fern e Brodley (2004), chamado HBGF (do inglês *Hybrid Bipartite Graph Formulation*), é descrito no Algoritmo 13.2.

Para o particionamento do grafo mencionado na linha 14 do algoritmo, Fern e Brodley (2004) utilizam duas técnicas distintas: *Spectral Graph Partitioning* (Ng et al., 2002) e METIS (Karypis e Kumar, 1999).

Algoritmo 13.2 Algoritmo HBGF

Entrada: Um conjunto de partições base Π
 Um conjunto de dados $\mathbf{X}_{n \times d}$
 Número de *clusters* k

Saída: Uma partição consenso π^F

- 1 Construir um grafo $G = (V, W)$ a partir do conjunto de partições base, da seguinte maneira:
- 2 $V^C \leftarrow$ vértices representando os *clusters* do conjunto de partições base Π
- 3 $V^O \leftarrow$ vértices representando os objetos do conjunto de dados \mathbf{X}
- 4 $V \leftarrow V^C \cup V^O$
- 5 **para cada** aresta ligando os vértices i e $j \in V$ **faca**
- 6 **se** $i, j \in V^C$ **ou** $i, j \in V^O$ // ambas representam *clusters* ou ambas representam objetos
- 7 **então**
- 8 $w(i, j) \leftarrow 0$
- 9 **senão se** o objeto \mathbf{x}_i pertence ao cluster \mathbf{C}_j **então**
- 10 $w(i, j) \leftarrow 1$
- 11 **senão**
- 12 $w(i, j) \leftarrow 0$
- 13 **fim**
- 14 Particionar o grafo $G = (V, W)$ em k *clusters* utilizando qualquer técnica de particionamento de grafos tradicional
- 15 Compor a partição consenso π^F de acordo com a divisão dos objetos, resultante do particionamento do grafo

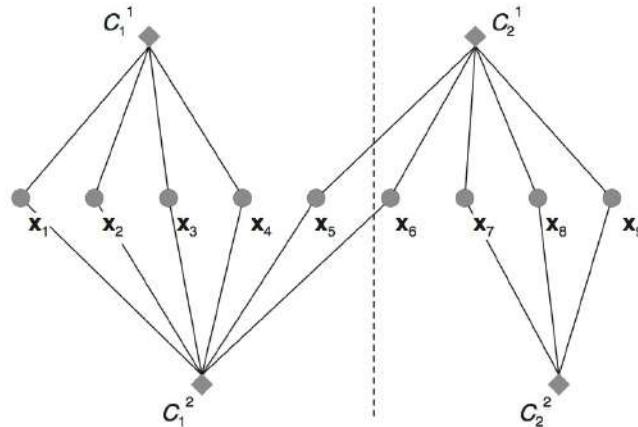
Para gerar as partições base, Fern e Brodley (2004) consideraram duas abordagens. A primeira aplica um algoritmo a diferentes subconjuntos dos dados (reamostragem). A segunda aplica um algoritmo ao conjunto de dados completo (todos os objetos), porém composto de projeções dos objetos em um espaço de dimensão menor do que o espaço de atributos original.

Nos experimentos apresentados em Fern e Brodley (2004), os autores comentam que sua abordagem, HBGF, apresentou um desempenho equivalente ou superior àqueles obtidos pelas abordagens de Strehl e Ghosh (2002).

Para ilustrar o funcionamento do algoritmo HBGF, considere os agrupamentos ilustrados na Tabela 13.6 (Fern e Brodley, 2004). Inicialmente é construído o grafo bipartido mostrado na Figura 13.1. Os vértices de V^C , representados por um losango, correspondem aos *clusters* de π^1 e π^2 , e os vértices de V^O , representados por um círculo, correspondem aos objetos. Todas as arestas representadas têm peso 1 e ligam um objeto a um *cluster*, indicando que o objeto pertence àquele *cluster*. A linha tracejada destaca uma partição desse grafo em duas partes, obtida com algum algoritmo de particionamento de grafos tradicional. A divisão dos objetos resultante desse particionamento é a partição consenso π^F , composta pelos *clusters* $\mathbf{C}_1^F = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}$ e $\mathbf{C}_2^F = \{\mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8, \mathbf{x}_9\}$.

Tabela 13.6 Exemplo do HBGF - partições

Partição	Clusters
π^1	$C_1^1 = \{x_1, x_2, x_3, x_4\}$, $C_2^1 = \{x_5, x_6, x_7, x_8, x_9\}$
π^2	$C_1^2 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, $C_2^2 = \{x_7, x_8, x_9\}$

**Figura 13.1** Exemplo do HBGF - grafo bipartido.

13.2 Agrupamento Multiobjetivo

A ideia básica do agrupamento multiobjetivo é otimizar simultaneamente dois ou mais critérios de agrupamento que sejam complementares (Handl e Knowles, 2004, 2005a,b, 2007). Um dos primeiros algoritmos de agrupamento multiobjetivo foi o MOCK (*Multi-Objective Clustering with automatic K-determination*), proposto por Handl e Knowles (2004, 2005a,b, 2007).

O algoritmo MOCK é um algoritmo evolutivo multiobjetivo, capaz de otimizar simultaneamente dois objetivos complementares e identificar automaticamente as melhores partições do fronte de Pareto, determinando de forma automática o número de *clusters*. Com base na forma do fronte de Pareto, MOCK retorna não apenas um conjunto de partições com diferentes compromissos em um intervalo de números de *clusters*, mas também uma indicação de quais dessas partições são as melhores. MOCK tenta encontrar o fronte de Pareto mais completo possível para, posteriormente, reduzir esse conjunto a uma única solução, por meio de uma pontuação calculada com base em frontes de referência (*attainment score*).

O MOCK é baseado no algoritmo evolutivo multiobjetivo PESA-II (Corne et al., 2001). Esse algoritmo mantém duas populações de soluções: uma interna, de tamanho fixo, e uma externa, de tamanho variável e limitado. O propósito da população externa é tirar proveito das boas soluções. Para isso, PESA-II utiliza elitismo, mantendo um conjunto grande e diverso de soluções não dominadas. A população interna é usada para investigar novas soluções por meio dos processos padrão de recombinação e mutação. As soluções presentes na população externa são mantidas em nichos. É mantido um registro do número de soluções que ocupam cada nicho, e esse registro é utilizado para fazer com que as soluções cubram todo o espaço de objetivos, ao invés de se agruparem todas em uma única região. Para isso, as soluções não dominadas que entrariam em uma população externa cheia apenas o farão se elas ocuparem um nicho menos cheio do que algumas ou-

tras soluções. Além disso, quando a população interna é construída a partir da população externa, os indivíduos são selecionados uniformemente dentre os nichos povoados (todos os nichos contribuem igualmente). A política de seleção baseada em nichos do PESA-II utiliza uma faixa adaptável de equalização e normalização dos valores das funções objetivos. Isso torna desnecessário o ajuste de parâmetros, que muitas vezes é complicado, e faz com que funções objetivo com variações diferentes possam ser prontamente utilizadas. Além disso, qualquer número de objetivos pode ser utilizado.

A representação dos indivíduos utilizada no MOCK é a representação de adjacência baseada em lócus (*locus-based adjacency representation*) (Park e Song, 1998). Nessa representação, cada indivíduo g consiste em n genes, g_1, g_2, \dots, g_n . Cada gene g_i pode assumir um valor j no intervalo $[1, n]$, significando que existe uma ligação entre os objetos \mathbf{x}_i e \mathbf{x}_j , ou seja, os objetos \mathbf{x}_i e \mathbf{x}_j estão no mesmo *cluster*. A decodificação dessa representação requer a identificação de todos os subgrafos. Todos os objetos pertencentes ao mesmo subgrafo são associados ao mesmo *cluster*.

Como operador de recombinação, é utilizado cruzamento uniforme. Para mutação, os autores originalmente empregaram um operador especializado que reduz significativamente o tamanho do espaço de busca (Handl e Knowles, 2004), a mutação dos vizinhos mais próximos. Nessa mutação, cada objeto pode ter sua ligação alterada apenas para um dos seus v vizinhos mais próximos. Consequentemente, $g_i \in \{nn_{i1}, \dots, nn_{iv}\}$, em que nn_{il} é o l -ésimo vizinho mais próximo do objeto \mathbf{x}_i . Nesse caso, todos os genes têm uma probabilidade de mutação igual ($\frac{1}{n}$). Posteriormente, os autores propuseram uma modificação na mutação que altera a probabilidade de mutação de ligações individuais, $i \rightarrow j$, para $p_m = \frac{1}{n} + (\frac{l}{n})^2$, em que $j = nn_{il}$ (Handl e Knowles, 2005b).

O procedimento para inicialização da população originalmente utiliza árvore geradora mínima (MST, do inglês *Minimum Spanning Tree*) para gerar os indivíduos (Handl e Knowles, 2004). No início, é gerada uma MST completa utilizando o algoritmo de Prim (Wilson e Watkins, 1990). O i -ésimo indivíduo da população inicial é inicializado pela MST com as $(i - 1)$ -ésimas ligações mais longas removidas. Porém, esse procedimento tende a gerar soluções boas na região do fronte de Pareto em que a conectividade é baixa e, quando os *clusters* não são bem separados, gerar soluções muito parecidas (Handl e Knowles, 2005b). Para melhorar o espalhamento das soluções iniciais, Handl e Knowles (2005b) propõem uma nova inicialização baseada em uma mistura de soluções geradas com o algoritmo k -médias e com MST.

Para as soluções baseadas na MST, inicialmente é construída uma MST e são identificadas todas as suas b ligações interessantes. Uma ligação $i \rightarrow j$ é considerada interessante se e somente se $i = nn_{jl}$ e $j = nn_{ik}$, com $l > v$ e $k > v$, em que v , o número de vizinhos mais próximos, é dado pelo usuário. O grau de interessabilidade é $gi = \min(l, k)$. As b ligações interessantes são ordenadas pelo seu grau de interessabilidade. O conjunto de partições baseadas na MST é construído da seguinte maneira: para cada $g \in [0, \min(b, 0, 5n^I)]$, em que n^I é o tamanho da população inicial, é gerado um agrupamento π^g removendo as g primeiras ligações interessantes. As ligações perdidas são substituídas por uma ligação com um vizinho j escolhido aleatoriamente, com $j = nn_{il}$ e $l \leq v$. Para as soluções baseadas no k -médias, o algoritmo é executado, gerando partições com números de *clusters* $k \in [2, n^I - (\min(b, 0, 5n^I) + 1)]$. As partições obtidas dessa maneira são convertidas para a representação apropriada.

Como funções objetivo, MOCK emprega dois objetivos complementares, a variância intracluster (*var*) (Equação 14.1) e a conectividade (*con*) (Equação 14.2), ambos definidos

na Seção 14.2.1. Essas medidas foram escolhidas por representar dois aspectos fundamentalmente diferentes de qualidade de um agrupamento. Esses objetivos contrabalançam suas tendências de aumentar ou diminuir com o número de *clusters*. Isso é importante para explorar bem o espaço de soluções, evitando a convergência para soluções triviais (n *clusters* com um único objeto, no caso da variância intracluster, e apenas um *cluster* com n elementos, no caso da conectividade) (Handl e Knowles, 2004).

A aplicação desse algoritmo gera um conjunto de soluções não dominadas com diferentes compromissos dos dois objetivos e com diferentes números de *clusters*. Para encontrar a melhor solução, Handl e Knowles geram o fronte de Pareto mais completo possível e, posteriormente, fazendo uso de várias considerações específicas do domínio, reduzem o conjunto de soluções a uma única solução (Handl e Knowles, 2004, 2005a).

O procedimento para selecionar a melhor solução é baseado na intuição de que a estrutura dos dados está refletida na forma do fronte de Pareto (Handl e Knowles, 2004, 2005a). Das tendências observadas nos objetivos empregados, é possível afirmar que, incrementando o número de *clusters*, k , obtém-se uma melhora na variância, δV , ao custo de uma degradação na conectividade, δC . Para um número de *clusters* k menor do que o verdadeiro, espera-se que a razão $R = \delta V / \delta C$ seja grande, pois a separação de dois *clusters* causa uma grande diminuição na variância, com pouco ou nenhum aumento na conectividade. Para os números de *clusters* maiores do que o verdadeiro, essa razão se torna menor, pois a diminuição na variância é menor, mas ao preço de um aumento maior da conectividade (um *cluster* verdadeiro está sendo dividido). Pela tendência das medidas, as soluções no fronte de Pareto estão aproximadamente ordenadas por k .

Assim, a mudança evidente em R que ocorre no número correto de *clusters* pode ser observada como um ponto de inflexão no gráfico do fronte de Pareto. Para determinar corretamente esse ponto, os autores utilizam distribuições aleatórias de dados como referência (Handl e Knowles, 2004, 2005a,b). Esses dados são agrupados com o MOCK, gerando um conjunto de frontes de referência. O fronte solução é normalizado, e, para cada ponto nesse fronte, é calculado um *attainment score*, dado pela sua distância até as *attainment surfaces* dos frontes de referência. Em seguida, é traçado um gráfico dos *attainment scores* em função de k . A solução correspondente ao máximo dessa curva é selecionada como a melhor solução.

Além de uma melhor solução, também é possível identificar outros possíveis máximos locais, que podem revelar estruturas em outros níveis. Além disso, o *attainment score* também serve como uma estimativa da qualidade de cada uma das soluções individuais.

Em Handl e Knowles (2004, 2005a), MOCK é comparado a três algoritmos de agrupamento convencionais (k -médias e algoritmos hierárquicos com ligação simples e média) e também ao *ensemble* de agrupamentos proposto por Strehl e Ghosh (2002), utilizando os três algoritmos e a função supraconsenso citados anteriormente. Os autores mostraram que MOCK é mais robusto do que as outras abordagens em relação à variedade de estruturas encontradas em conjuntos de dados diferentes e é capaz de encontrar certas estruturas que outros métodos não conseguem.

13.3 Ensemble Multiobjetivo

O *ensemble* multiobjetivo combina as duas abordagens previamente apresentadas. Ele cria um conjunto diverso de partições base que são depois combinadas para a determina-

ção do consenso, utilizando para isso estratégias multiobjetivo. Com isso, os *ensembles* multiobjetivo encontram um conjunto de partições consenso, que representam diferentes compromissos entre os critérios considerados na combinação. Dentre as abordagens de *ensembles* multiobjetivo estão o MOCLE (do inglês *Multi-Objective Clustering Ensemble*) (Faceli et al., 2009) e o MCHPF (*Multi-objective Clustering with Hierarchical Partitions Fusions*) (Coelho et al., 2010).

O MOCLE integra a saída (*output*) de diversos algoritmos de agrupamento, técnicas de validação e *ensemble* de agrupamentos em uma abordagem multiobjetivo, para encontrar um conjunto de estruturas que podem conter informações relevantes para os especialistas no domínio dos dados.

O algoritmo MOCLE, como qualquer *ensemble*, pode ser dividido em dois blocos:

- Geração de um conjunto diverso de partições iniciais a serem combinadas;
- Determinação do consenso.

O MOCLE difere dos *ensembles* tradicionais em dois aspectos, relacionados à obtenção do consenso. Em primeiro lugar, o MOCLE busca por um conjunto de partições consenso, em lugar de uma única partição. Na verdade, o conjunto de soluções que o MOCLE retorna pode conter tanto partições que resultam da combinação de outras partições quanto partições de alta qualidade que já apareciam entre as partições iniciais. A segunda diferença do MOCLE em relação aos demais *ensembles* é que ele combina pares de partições, iterativamente, em um processo de otimização que garante diferentes compromissos de qualidade das soluções. Com isso, o MOCLE consegue evitar a influência negativa das partições iniciais de baixa qualidade que afeta as abordagens tradicionais de *ensemble*.

Mais precisamente, o MOCLE deve ser iniciado com a geração de um conjunto de partições iniciais por meio da aplicação de vários algoritmos de agrupamento conceitualmente diferentes aos dados, também considerando várias configurações de parâmetros. Isso garante a diversidade das partições iniciais do *ensemble*. Em seguida, essas partições iniciais são utilizadas como população inicial para um algoritmo genético multiobjetivo baseado em Pareto. Esse algoritmo vai selecionar e combinar as partições iniciais por meio de duas características particulares: (1) um operador de recombinação especial, que encontra o consenso entre duas partições pais, e (2) a otimização de funções objetivo que representam diferentes medidas de qualidade de uma partição.

O operador de recombinação fornece a característica de *ensemble* ao MOCLE, o que o diferencia da abordagem de agrupamento multiobjetivo pura.

Com essas características, o MOCLE faz uma seleção automática das partições mais significativas, dentre as iniciais e as combinações, sem que sejam necessários muitos ajustes de parâmetros e nem conhecimento profundo em análise de agrupamento. Com isso, ele supera algumas das dificuldades da análise de agrupamento tradicional. Além disso, a integração das abordagens de *ensemble* e agrupamento multiobjetivo permite superar algumas das dificuldades individuais de ambas as abordagens.

Em resumo, o MOCLE constitui uma abordagem robusta para lidar com diferentes tipos de estrutura que podem estar presentes nos dados, fornecendo como resultado um conjunto conciso e estável de estruturas alternativas de elevada qualidade, sem a necessidade de conhecimento prévio dos dados e nem conhecimento profundo em análise de agrupamento.

O MCHPF, por sua vez, utiliza programação genética para evoluir uma população de *ensembles* e identificar as melhores maneiras de combinar subconjuntos de partições a fim de obter partições relevantes. Nessa abordagem, um conjunto de partições base é inicialmente gerado, da mesma maneira que no MOCLE. Com essas partições, são criados os *ensembles* iniciais, que são os indivíduos da população inicial. Cada *ensemble* é uma hierarquia de fusão gerada aleatoriamente, em que uma ou mais funções consenso tradicionais (como as descritas na Seção 13.1) são aplicadas a um subconjunto das partições base. A população de *ensembles* passa então pelo processo de evolução considerando uma estratégia multiobjetivo, em que são otimizadas as mesmas funções objetivo do MOCLE. Os *ensembles* resultantes do processo de evolução podem, posteriormente, ser executados para a geração das partições consenso. As principais vantagens do MCHPF em relação ao MOCLE se devem ao uso das hierarquias de fusão como soluções, em vez das partições propriamente ditas, aliado ao uso de diferentes funções consenso para a construção de um *ensemble*. Isso permite que as combinações sejam feitas de diferentes maneiras e que se tenha conhecimento das combinações feitas em cada *ensemble*, garantindo a identificação das partições que contribuíram no resultado final, bem como da maneira como as partições foram combinadas.

13.4 Considerações Finais

Neste capítulo, foram apresentados diversas abordagens de modelos múltiplos descritivos, mais especificamente relacionados ao tema de análise de agrupamento. As abordagens apresentadas permitem a superação de uma série de dificuldades da aplicação tradicional da análise de agrupamento.

Mais especificamente, foram apresentados os *ensembles* de agrupamentos, os algoritmos de agrupamento multiobjetivo e abordagens que combinam ambas as ideias.

Os *ensembles* de agrupamentos atuam por meio da combinação de um conjunto de agrupamentos obtidos previamente, enquanto os algoritmos multiobjetivo realizam a otimização direta de mais de um critério de agrupamento, resultando em um conjunto de soluções.

Os *ensembles* são mais robustos e fornecem soluções de melhor qualidade que os algoritmos tradicionais de agrupamento, porém não exploram todo o potencial de se usar múltiplos critérios. Já os algoritmos de agrupamento e os *ensembles* multiobjetivo têm uma maior facilidade para encontrar diferentes tipos de *clusters*, bem como para lidar com estruturas heterogêneas.

Capítulo 14

Avaliação de Modelos Descritivos

A análise e a comparação de resultados em análise de agrupamento podem ser consideradas sob o ponto de vista de dois objetivos diferentes: avaliação e comparação de algoritmos de agrupamento e validação das estruturas encontradas por algoritmos de agrupamento. Esses dois objetivos têm em comum o fato de estarem ligados ao tema de validação de agrupamentos.

Na avaliação e comparação de estratégias de agrupamento, que podem envolver tanto apenas algoritmos individuais quanto modelos múltiplos, é preciso garantir a corretude, a validade e a reproduzibilidade dos experimentos realizados e das conclusões obtidas a partir de seus resultados, da mesma maneira que é necessário nas técnicas preditivas.

Na análise exploratória, uma análise prévia dos dados pode ser de grande utilidade para guiar as escolhas feitas em todas as etapas do processo de agrupamento, previamente descritas no Capítulo 11. Tal análise pode ser feita com técnicas da Estatística Descritiva, como as apresentadas no Capítulo 2, ou com técnicas de visualização dos dados. Entretanto, mesmo feitas as escolhas adequadas, é preciso checar a validade das estruturas encontradas.

Para uma melhor compreensão das discussões apresentadas nas seções seguintes, é importante entender a diferença entre a validação feita para avaliar/comparar algoritmos de agrupamento e aquela feita para validar as estruturas encontradas na análise exploratória dos dados. Levando em conta que a análise de agrupamento é uma tarefa não supervisionada, deve-se ter sempre em mente que não existe um resultado correto, que seja o objetivo final a ser atingido com qualquer algoritmo de agrupamento. Isso é especialmente importante na análise exploratória. Entretanto, na avaliação/comparação de algoritmos, principalmente na avaliação da efetividade de algoritmos novos, é importante ter mecanismos para identificar se o algoritmo está realmente encontrando uma estrutura apropriada, ou se é comparável a outros existentes. Nesses casos, muitas vezes são utilizados dados para os quais se conhecem uma ou mais estruturas, e os algoritmos são avaliados com respeito à sua habilidade em encontrar essas estruturas conhecidas.

Um outro aspecto importante que é preciso considerar quando se avaliam agrupamentos é que nem sempre existe uma solução única, considerando as possíveis definições do que seja um *cluster*, as diferentes maneiras como dois objetos podem ser considerados similares e a possibilidade de existência de várias estruturas, quer homogêneas quer heterogêneas, em um conjunto de dados.

Essas características tornam a avaliação dos resultados em agrupamento uma tarefa complexa, pois não existe uma resposta esperada com a qual comparar os resultados obtidos pelos algoritmos, e, muitas vezes, não existe uma resposta única.

De qualquer maneira, é preciso seguir procedimentos rigorosos, considerando todos os

aspectos descritos previamente no Capítulo 11, para garantir resultados efetivos e úteis, e também a reproduzibilidade das análises. Como já mencionado, a avaliação do resultado de um agrupamento deve ser objetiva, visando determinar se a estrutura encontrada é válida, ou seja, se não ocorreu por acaso, ou se é “rara” em algum sentido, já que qualquer algoritmo de agrupamento encontrará *clusters*, independentemente de se existe ou não estrutura nos dados. Entretanto, mesmo que essa estrutura exista, alguns algoritmos podem encontrar *clusters* mais adequados que outros. Diversas técnicas para a validação em análise de agrupamento têm sido discutidas na literatura. Alguns desses estudos podem ser encontrados em Jain e Dubes (1988); Gordon (1999); Halkidi et al. (2001); Handl et al. (2005).

A validação do resultado de um agrupamento, em geral, é baseada em índices estatísticos, que julgam, de uma maneira qualitativa, o mérito das estruturas encontradas. Um índice quantifica alguma informação a respeito da qualidade de um agrupamento. A maneira pela qual um índice é aplicado para validar um agrupamento é dada pelo critério de validação. Assim, um critério de validação expressa a estratégia utilizada para validar uma estrutura de agrupamento, enquanto um índice é uma estatística pela qual a validade é testada. Existem três tipos de critérios para investigar a validade de um agrupamento: critérios internos, externos e relativos. A Seção 14.1 descreve detalhadamente cada um desses critérios. Neste livro, os índices que são mais comumente empregados em critérios externos, internos e relativos serão denominados simplificadamente índices externos, internos e relativos, respectivamente. Posteriormente, nas Seções 14.2, 14.3 e 14.4, serão detalhados alguns dos índices empregados, respectivamente, com critérios relativos, internos e externos, bem como as metodologias mais frequentemente empregadas para a aplicação desses índices.

14.1 Critérios de Validação

Conforme já dito, um critério de validação expressa a estratégia utilizada para validar um agrupamento, enquanto um índice é uma estatística pela qual a validade é testada. Ou, de outra forma, o critério de validação indica a maneira pela qual um índice é aplicado para validar um agrupamento. Existem três tipos de critérios para investigar a validade de um agrupamento:

- **Critérios relativos:** comparam diversos agrupamentos com respeito a algum aspecto (qual é mais estável ou qual é o mais adequado aos dados, por exemplo). Podem ser utilizados para comparar diversos algoritmos de agrupamento ou para determinar o valor mais apropriado para um ou mais parâmetros de um algoritmo, como o número de *clusters*. Com isso, esses critérios permitem medir quantitativamente qual dentre dois algoritmos melhor se ajusta aos dados ou determinar o número de *clusters* mais apropriado para um agrupamento produzido por um determinado algoritmo.
- **Critérios internos:** medem a qualidade de um agrupamento com base apenas nos dados originais (matriz de objetos ou matriz de similaridade). Por exemplo, um critério interno pode medir o grau com que uma partição obtida por um algoritmo de agrupamento é justificada pela matriz de similaridade.

- **Critérios externos:** avaliam um agrupamento de acordo com uma estrutura estabelecida previamente, que pode refletir, por exemplo, a intuição do pesquisador sobre a estrutura presente nos dados. Essa estrutura pré-especificada pode ser uma partição previamente conhecida para os dados ou um agrupamento sugerido por um especialista da área baseado em conhecimento prévio. Por exemplo, um critério externo pode medir o grau de correspondência entre o número *clusters* obtido com o agrupamento e rótulos já conhecidos para os dados.

Esses critérios de validação podem ser utilizados para avaliar vários tipos de estrutura como hierarquias, partições (*hard* ou *fuzzy*) e *clusters* individuais. A seguir, serão discutidas as principais abordagens para utilização dos três critérios no contexto de avaliação de partições.

Os critérios relativos são utilizados para encontrar o melhor algoritmo de agrupamento (e/ou conjunto de valores de parâmetros para o mesmo algoritmo) em relação a um grupo de outros algoritmos (e/ou conjuntos de valores de parâmetros diferentes para o mesmo algoritmo).

Existem vários índices que podem ser empregados com critérios relativos. Alguns dos índices mais comuns, assim como alguns propostos recentemente, são apresentados na Seção 14.2.1. Esses índices, em geral, podem ainda ser empregados em critérios internos (Jain e Dubes, 1988). O que distingue a utilização de um índice em um ou outro critério é a maneira como o índice é aplicado. A forma mais comum de aplicação de um índice com um critério relativo é o cálculo do seu valor para vários agrupamentos que estão sendo comparados, obtendo-se uma sequência de valores. O melhor agrupamento é determinado pelo valor que se destaca nessa sequência, como um valor máximo, mínimo ou uma inflexão na curva do gráfico construído com a sequência.

A Figura 14.1 resume a metodologia mais utilizada para aplicação do critério relativo de validação. Como indica a Figura 14.1, nesse critério, vários algoritmos, ou um mesmo algoritmo com diferentes valores para seus parâmetros, são aplicados ao mesmo conjunto de dados. O índice é calculado para cada uma das partições obtidas. Esses valores do índice são comparados, em geral com o auxílio de um gráfico, para determinar o melhor algoritmo ou o melhor valor para um ou mais parâmetros de um algoritmo.

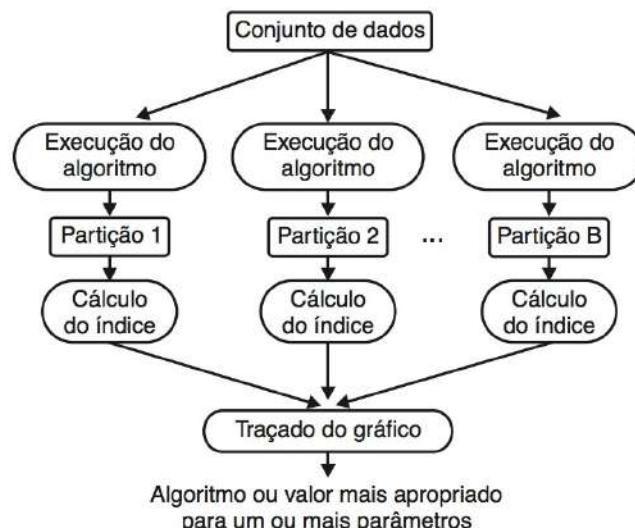


Figura 14.1 Critério relativo de validação.

Além da forma previamente descrita de validação relativa, outras abordagens têm sido exploradas na literatura. Algumas dessas abordagens serão descritas na Seção 14.2.2. Serão detalhadas a análise de replicação, proposta por McIntyre e Blashfield (1980) e Morey et al. (1983), similar à validação cruzada previamente descrita no Capítulo 9, a abordagem de Law e Jain (2003), que calcula a variabilidade do algoritmo utilizando *bootstrapping*, e as abordagens de Yeung et al. (2001) e Tibshirani et al. (2001a), que se baseiam no poder preditivo do algoritmo.

Os critérios externos e internos de validação são baseados em testes estatísticos e têm um alto custo computacional (Halkidi et al., 2001). Seu objetivo é medir o quanto o resultado obtido confirma uma hipótese pré-especificada. Nesse caso, são utilizados testes de hipótese para determinar se uma estrutura obtida é apropriada para os dados, o que é feito verificando se o valor do índice utilizado é extraordinariamente grande ou pequeno. Isso requer o estabelecimento de uma população base ou de referência. O mesmo índice pode ser utilizado em um critério externo e interno, embora as distribuições de referência do índice sejam diferentes (Jain e Dubes, 1988).

A proposição de um índice para validação é fácil, porém é muito difícil estabelecer limiares que permitam afirmar que o valor do índice é grande ou pequeno o suficiente para se considerar o agrupamento “raro” e potencialmente útil ou válido.

Os índices de validação são funções dos dados que contêm informações úteis, como o erro quadrático de um agrupamento ou a compactação de seus *clusters*. É importante observar que um índice é uma variável aleatória. Sua distribuição descreve a frequência relativa com a qual seus valores são gerados sob alguma hipótese. Uma hipótese é uma afirmação sobre a frequência relativa de eventos no espaço amostral que expressa um conceito.

No caso de validação de agrupamentos, a hipótese nula, H_0 , é uma afirmação de aleatoriedade ou falta de estrutura nos dados. Jain e Dubes (1988) e Gordon (1999) descrevem algumas hipóteses nulas comumente utilizadas para a validação de agrupamentos. Jain e Dubes (1988) discutem ainda aplicações de cada uma dessas hipóteses. A seleção da hipótese nula depende do tipo dos dados e do aspecto que está sendo analisado sobre os dados.

Jain e Dubes (1988) resumem os principais aspectos relacionados à utilização de um índice de validação:

- **Definição do índice:** o índice deve fazer sentido intuitivamente, deve ter uma base teórica e deve ser prontamente computável.
- **Distribuição de probabilidade-base:** uma distribuição-base é uma distribuição derivada de uma população que não possui estrutura. Uma população referência é definida ou implicada pela distribuição-base.
- **Teste para verificar estrutura não aleatória:** o valor de um índice de validação é comparado a um limiar que estabelece um dado nível de significância. O limiar é definido a partir da distribuição-base, que, em teoria, raramente é conhecida.
- **Teste para verificar um tipo de estrutura:** a habilidade do índice de validação em recuperar uma estrutura conhecida indica seu poder estatístico. A escolha da estrutura depende da aplicação.

As Figuras 14.2 e 14.3 resumem, respectivamente, os critérios externos e internos de validação. Conforme dito anteriormente, nesses critérios é utilizado um teste de hipótese que depende da distribuição do índice sob uma hipótese nula, H_0 . A diferença entre esses critérios está nas informações utilizadas. Nos critérios externos, pode ser utilizada uma partição dos dados conhecida previamente, chamada de “partição real” ou “estrutura conhecida”, no cálculo do índice. Já nos critérios internos, o cálculo do índice depende somente dos próprios dados, seja na forma de matriz de objetos (conjunto de dados original) ou de matriz de similaridade.

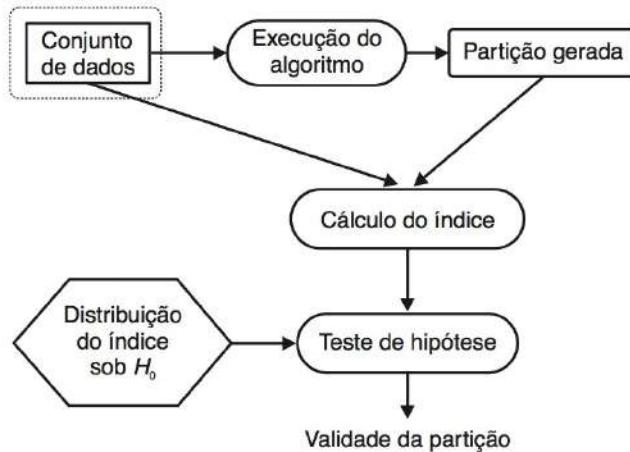


Figura 14.2 Critério interno de validação.

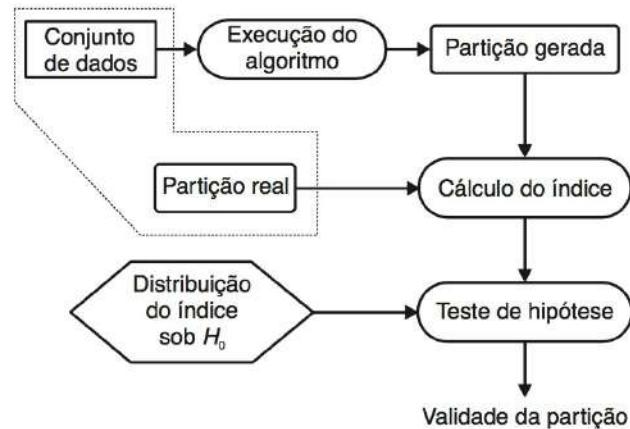


Figura 14.3 Critério externo de validação.

Um grande problema relacionado à validação externa e interna de agrupamentos é o estabelecimento da distribuição dos índices (estatísticas) sob a hipótese nula e, consequentemente, a determinação dos limiares que informam se uma partição é adequada de acordo com o índice. Assim, na prática, os testes de validação são geralmente definidos utilizando ferramentas estatísticas, como análise de Monte Carlo e *bootstrapping*.

Análise de Monte Carlo é um método para estimar parâmetros e taxas de probabilidade por meio de amostragem por computador. É utilizada quando tais valores são difíceis ou impossíveis de ser calculados diretamente.

Umas das formas mais comuns de utilização de análise de Monte Carlo para validação de agrupamentos é no estabelecimento da distribuição referência de um índice sob a hipótese nula. Inicialmente, é gerada uma grande quantidade de conjuntos de dados sintéticos

de acordo com a distribuição considerada na hipótese nula H_0 . Em seguida, cada um desses conjuntos é agrupado e o valor do índice é calculado em cada caso. Com esses valores do índice, é traçado um gráfico de dispersão, que é uma aproximação da função de densidade de probabilidade do índice. Dado o valor do índice para o agrupamento que está sendo validado e a distribuição estimada, determina-se se a hipótese H_0 deve ser aceita ou rejeitada. Com a utilização da análise de Monte Carlo, o limiar para determinar se o valor do índice é grande ou pequeno o suficiente (ou seja, rejeitar H_0) pode ser visto como um intervalo de valores, ao invés do valor único obtido quando a distribuição sob H_0 é conhecida (Jain e Dubes, 1988). A Figura 14.4 resume essa forma de aplicação de análise de Monte Carlo para validação de agrupamentos.

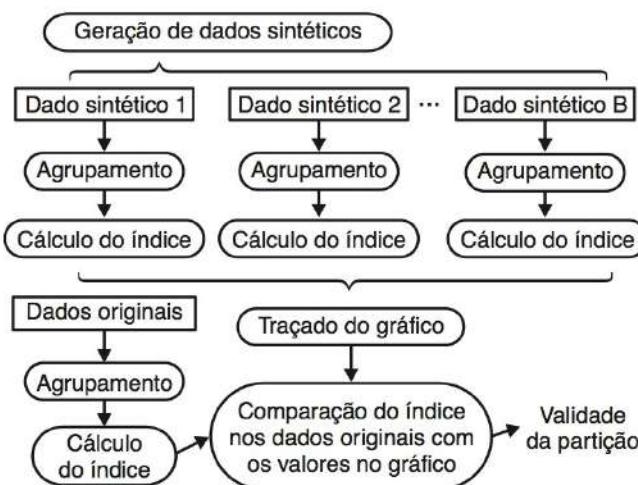


Figura 14.4 Análise de Monte Carlo para validação de um agrupamento.

Não é fácil definir o modelo nulo. Existe uma grande quantidade de tipos de modelos nulos, cada um com suas vantagens e desvantagens (Gordon, 1996). Segundo Filho (2003), Gordon (1996) sugere a utilização de mais de um modelo nulo, o que torna o processo de validação ainda mais complexo e demorado. Segundo Jain e Dubes (1988), a parte mais difícil na análise de Monte Carlo é a obtenção de uma amostra de distribuição arbitrária. Geralmente existem geradores de números aleatórios que geram amostras pseudoaleatórias de distribuição $U(1, 0)$ e existem técnicas para transformar essas amostras uniformes em amostras de outras distribuições.

Outro problema da análise de Monte Carlo é a necessidade de uma grande quantidade de recursos computacionais, uma vez que é necessário um grande número de replicações para construir a distribuição de referência. Entretanto, com os avanços tecnológicos recentes, esse problema é cada vez menor.

As técnicas de *bootstrapping* utilizam reamostragem dos dados, com substituição, para criar um conjunto “falso” de dados, que simula uma replicação de um experimento de Monte Carlo (Jain e Dubes, 1988). Nesse caso, as amostras *bootstrap* são utilizadas para construir o modelo nulo. As amostras podem ser obtidas pela reamostragem dos objetos ou dos atributos do conjunto de dados. Com *bootstrapping*, evitam-se os problemas da análise de Monte Carlo relacionados à escolha do modelo nulo. Existem outras formas de aplicação de *bootstrapping* em validação, algumas das quais são empregadas em índices descritos mais adiante.

14.2 Critérios Relativos

Conforme já foi dito, o objetivo da validação relativa é encontrar um agrupamento que melhor se ajuste aos dados, a partir de vários agrupamentos obtidos com um algoritmo sob certas suposições e valores para seus parâmetros, ou encontrar o algoritmo mais apropriado para agrupar os dados analisados de forma a encontrar as estruturas desejadas.

Uma das formas mais comuns de utilização dos critérios relativos é na determinação do número mais adequado de *clusters*. Nesse caso, o algoritmo de agrupamento é executado para todos os possíveis números de *clusters*, k , entre k_{min} e k_{max} , fornecidos. Em seguida, os valores do índice obtidos a partir dessas execuções são apresentados, na forma de um gráfico, como função de k . O melhor número de *clusters* é dado pelo mínimo, máximo ou inflexão na curva observada. Halkidi et al. (2002b) descrevem resumidamente a utilização de índices em critérios relativos para determinar outros parâmetros de algoritmos de agrupamento.

Um grande número de índices que podem ser empregados com critérios relativos pode ser encontrado na literatura da área. A título de ilustração, a Seção 14.2.1 apresenta alguns desses índices.

Além da aplicação tradicional da validação relativa (cálculo de um índice para várias partições e observação do gráfico do índice em função de k), existem algumas outras abordagens descritas na literatura, algumas das quais serão apresentadas na Seção 14.2.2.

14.2.1 Índices Empregados em Critérios Relativos

Variância Intracluster, $var(\pi)$

A variância intracluster de uma partição π é dada pela Equação 14.1, em que $\bar{\mathbf{x}}^{(k)}$ é o centroide do *cluster* (Handl et al., 2005). Ela mede a qualidade de um agrupamento em termos da compactação de seus *clusters*. Essa medida apresenta valores no intervalo $[0, \infty]$, e quanto menor o valor de var , melhor a partição.

$$var(\pi) = \sqrt{\frac{1}{n} \sum_{C_k \in \pi} \sum_{\mathbf{x}_i \in C_k} d(\mathbf{x}_i, \bar{\mathbf{x}}^{(k)})} \quad (14.1)$$

Conectividade, $con(\pi)$

A conectividade está ligada ao conceito de encadeamento e reflete o grau com que os objetos vizinhos são colocados no mesmo *cluster*. Os vizinhos mais próximos são aqueles com menor distância (ou maior similaridade) (Handl et al., 2005). A conectividade de uma partição é dada pela Equação 14.2, em que v é o número de vizinhos mais próximos que contribuem para a conectividade e nn_{ij} é o j -ésimo vizinho mais próximo ao objeto \mathbf{x}_i . Quanto menor o valor do índice con , melhor a partição. Os valores desse índice variam no intervalo $[0, v]$.

$$con(\pi) = \sum_{\mathbf{x}_i \in \mathbf{X}} \sum_{j=1}^v f(\mathbf{x}_i, nn_{ij}) \quad (14.2)$$

$$f(\mathbf{x}_i, nn_{ij}) = \begin{cases} 1/j & \text{se } \mathbf{x}_i \in \mathbf{C}_k, nn_{ij} \notin \mathbf{C}_k \\ 0 & \text{caso contrário} \end{cases} \quad (14.3)$$

Família de Índices Dunn, $D(\pi)$

A família de índices Dunn é representada pela Equação 14.4, em que $d(\mathbf{C}_a, \mathbf{C}_b)$ é uma função de distância entre os *clusters* \mathbf{C}_a e \mathbf{C}_b e $d(\mathbf{C}_a)$ é a distância intracluster do *cluster* \mathbf{C}_a , que mede a dispersão do *cluster* (Halkidi et al., 2002b). No índice Dunn original, $d(\mathbf{C}_a, \mathbf{C}_b)$ é dada pela Equação 14.5, que é a mesma medida usada pelo algoritmo hierárquico com ligação simples, e $d(\mathbf{C}_a)$ é dada pela Equação 14.6. Nesse caso, o índice mede a razão da separação dentro dos *clusters* e entre os *clusters* (Pakhira et al., 2004). Outras variações do índice consideram diferentes funções para $d(\mathbf{C}_a, \mathbf{C}_b)$ e $d(\mathbf{C}_a)$.

$$D(\pi) = \min_{a=1,\dots,k} \left\{ \min_{b=a+1,\dots,k} \left\{ \frac{d(\mathbf{C}_a, \mathbf{C}_b)}{\max_{l=1,\dots,k} d(\mathbf{C}_l)} \right\} \right\} \quad (14.4)$$

$$d(\mathbf{C}_a, \mathbf{C}_b) = \min_{\substack{\mathbf{x}_i \in \mathbf{C}_a, \\ \mathbf{x}_j \in \mathbf{C}_b}} d(\mathbf{x}_i, \mathbf{x}_j) \quad (14.5)$$

$$d(\mathbf{C}_a) = \max_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{C}_a} d(\mathbf{x}_i, \mathbf{x}_j) \quad (14.6)$$

O índice Dunn é apropriado para a identificação de *clusters* compactos e bem separados (Halkidi et al., 2002b). Valores altos do índice indicam a presença desse tipo de *cluster*. O índice $D(\pi)$ não apresenta nenhuma tendência em relação ao número de *clusters*. O ponto de máximo no gráfico de $D(\pi)$ contra k , em que diversas partições π foram obtidas com esses valores de k , pode ser uma indicação do número de *clusters* que mais se ajusta aos dados.

Os problemas do índice Dunn original são a sua complexidade e sensibilidade a ruído. Outros índices da família Dunn são mais robustos à presença de ruídos. Bezdek e Pal (1998) propõem várias generalizações desse índice e as comparam com outros índices. Azuaje (2002) apresenta uma ferramenta que emprega 18 índices baseados no índice Dunn para determinar o melhor número de *clusters*. Esses índices são baseados em diferentes combinações de diferentes distâncias entre *clusters* e intracluster. Esse índice não é apropriado para *clusters* de formas arbitrárias.

Silhueta, $sil(\pi)$

A medida silhueta se baseia na proximidade entre os objetos de um *cluster* e na distância dos objetos de um *cluster* ao *cluster* mais próximo (Rousseeuw, 1987). Ela pode ser utilizada para avaliar uma partição; para isso, avalia também a adequação de cada objeto ao seu *cluster* e a qualidade de cada *cluster* individualmente. O valor de uma medida silhueta é limitado pelo intervalo $[-1, 1]$, em que a melhor partição de acordo com a silhueta é aquela com valor 1.

As siluetas de um *cluster* e de uma partição para medidas de distância entre objetos são dadas pelas Equações 14.10 e 14.11, respectivamente, em que n é o número de objetos agrupados e $sil(\mathbf{x}_i)$ é a silhueta do objeto \mathbf{x}_i , dada pela Equação 14.7. Nessa equação, $a(\mathbf{x}_i, \mathbf{C}_i)$, definida pela Equação 14.8, representa a distância média do objeto \mathbf{x}_i em relação

a todos os outros objetos do *cluster* \mathbf{C}_i (*cluster* ao qual o objeto \mathbf{x}_i pertence) e $b(\mathbf{x}_i)$, dada pela Equação 14.9, a menor distância média de \mathbf{x}_i em relação a todos os demais *clusters*. Rousseeuw (1987) apresenta uma versão do índice para quando uma medida de similaridade é utilizada, em vez de uma medida de distância.

$$sil(\mathbf{x}_i) = \begin{cases} 1 - a(\mathbf{x}_i, \mathbf{C}_i)/b(\mathbf{x}_i), & a(\mathbf{x}_i, \mathbf{C}_i) < b(\mathbf{x}_i) \\ 0, & a(\mathbf{x}_i, \mathbf{C}_i) = b(\mathbf{x}_i) \\ b(\mathbf{x}_i)/a(\mathbf{x}_i, \mathbf{C}_i) - 1, & a(\mathbf{x}_i, \mathbf{C}_i) > b(\mathbf{x}_i) \end{cases} \quad (14.7)$$

$$a(\mathbf{x}_i, \mathbf{C}_k) = \frac{1}{|\mathbf{C}_k|} \sum_{\substack{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{C}_k \\ \mathbf{x}_i \neq \mathbf{x}_j}} d(\mathbf{x}_i, \mathbf{x}_j) \quad (14.8)$$

$$b(\mathbf{x}_i) = \min_{\substack{\mathbf{x}_j \in \mathbf{C}_i, \\ \mathbf{C}_i \neq \mathbf{C}_j}} a(\mathbf{x}_i, \mathbf{C}_j) \quad (14.9)$$

Quando a silhueta é calculada para cada objeto, seu valor será próximo de 1 se o objeto está bem situado dentro do seu *cluster*. Um valor próximo de -1 indica que o objeto deveria ser associado a outro *cluster* (Yeung, 2001).

A medida silhuetas depende apenas da partição dos dados, independe portanto do algoritmo de agrupamento empregado. Ela pode ser utilizada para melhorar os resultados de uma análise de *cluster* ou para comparar os resultados de diferentes algoritmos aplicados ao mesmo conjunto de dados.

Além da silhueta de cada objeto, pode ser calculada a silhueta de cada *cluster*, conforme a Equação 14.10 e a largura média da silhueta, $sil(\pi)$, utilizando a Equação 14.11. Um modo de escolher o melhor valor de k é selecionar aquele que resulta no maior valor de $sil(\pi)$.

$$sil(\mathbf{C}_k) = \frac{1}{|\mathbf{C}_k|} \sum_{\mathbf{x}_i \in \mathbf{C}_k} sil(\mathbf{x}_i) \quad (14.10)$$

$$sil(\pi) = \frac{1}{n} \sum_{i=1}^n sil(\mathbf{x}_i) \quad (14.11)$$

O coeficiente silhueta, SC , é o máximo $sil(\pi)$ para π gerada com $k = 2, 3, \dots, (n-1)$. SC é uma medida que quantifica a estrutura descoberta por um algoritmo de agrupamento. Uma interpretação para SC é: $SC \leq 0,25$ significa que não foi encontrada uma estrutura substancial, $0,26 \leq SC \leq 0,5$ indica que a estrutura encontrada é fraca e pode ser artificial, $0,5 \leq SC \leq 0,7$ significa que uma estrutura razoável foi encontrada, e $0,71 \leq SC \leq 1$ indica que foi encontrada uma estrutura forte.

As medidas de silhueta são apropriadas para a identificação de *clusters* compactos e bem separados. Assim, funcionam melhor com *clusters* aproximadamente esféricos (Rousseeuw, 1987). Uma vez que a definição da silhueta favorece objetos que estão associados a *clusters* com a similaridade média mais alta, as larguras das silhuetas são tendenciosas contra *clusters* verdadeiros potencialmente sobrepostos, favorecendo agrupamentos disjuntos.

Existem outros índices para avaliação de partições que, para tornar o texto mais coeso, não serão discutidos em detalhes, como o desvio total (*dev*) (Handl e Knowles, 2007), o

índice Davies-Bouldin (*DB*) (Jain e Dubes, 1988), o índice de Calinski-Harabasz (*CH*) (Calinski e Harabasz, 1974), o índice de Krzanowski e Lai (*KL*) (Krzanowski e Lai, 1985), o índice de Hartigan (*H*) (Hartigan, 1975), o índice *S_Dbw* (do inglês *compose density between and within clusters*) (Halkidi e Vaziriannis, 2001) e o índice de Pakhira et al. (2004), *PBM*. Descrições e comparações entre uma variedade grande de índices de validação podem ser encontradas em Milligan e Cooper (1985) e Vendramin et al. (2010).

Além desses índices, existem vários outros particularmente indicados para a avaliação de partições *fuzzy*. Nesse tipo de agrupamento, cada objeto apresenta um grau de pertinência em relação a cada *cluster*, em vez de ser associado unicamente a um *cluster*. Alguns dos índices mais comuns empregados para validação de agrupamentos *fuzzy* são: coeficiente de partição (*PC*) (Pal e Bezdek, 1995), entropia da partição (*PE*) (Pal e Bezdek, 1995), índice de Xie-Beni (*XB*) (Xie e Beni, 1991; Pal e Bezdek, 1995), índice de Xie-Beni estendido (Pal e Bezdek, 1995), índice de Fukuyama-Sugeno (*FS*) (Pal e Bezdek, 1995), separação da partição (*S*) (Yang e Wu, 2001) e índice de Pakhira et al. (2004) fuzzificado, *PBMF*.

Dentre os índices citados para validação de partições *fuzzy*, os índices *PC* e *PE* utilizam apenas o valor de pertinência dos objetos aos *clusters* para calcular seu valor. Já os demais índices utilizam, além do valor de pertinência, o próprio conjunto de dados.

14.2.2 Outras Abordagens de Validação Relativa

Nesta seção são descritas outras abordagens para validação relativa, em particular a análise de replicação, proposta por McIntyre e Blashfield (1980) e Morey et al. (1983), a abordagem de Law e Jain (2003), que calcula a variabilidade do algoritmo utilizando *bootstrapping*, e a abordagem de Yeung et al. (2001), que se baseia no poder preditivo do algoritmo. Tibshirani et al. (2001a) também apresentam uma abordagem baseada no poder preditivo, que não será detalhada.

A análise de replicação (McIntyre e Blashfield, 1980; Morey et al., 1983) se baseia em um argumento semelhante ao procedimento de *cross-validation* comumente usado em aprendizado supervisionado, descrito no Capítulo 9. A ideia é medir a estabilidade (ou replicabilidade) de um algoritmo de agrupamento. Essa estabilidade é medida comparando a partição obtida pelo algoritmo a uma partição obtida em um subconjunto independente dos dados.

A forma de validação de um agrupamento proposta por Law e Jain (2003) é fundamentada na interpretação de um algoritmo de agrupamento como um estimador estatístico e na avaliação da variabilidade desse estimador por meio de *bootstrapping*. Se uma partição é válida, sua variabilidade deve ser baixa.

As abordagens de Yeung et al. (2001) e Tibshirani et al. (2001a) seguem o princípio de que, se um agrupamento reflete uma estrutura verdadeira, um preditor construído com base nos *clusters* desse agrupamento deve estimar precisamente os rótulos dos *clusters* de novas amostras de teste (Jiang et al., 2004).

Análise de Replicação

A análise de replicação (McIntyre e Blashfield, 1980; Morey et al., 1983; Milligan, 1996) é ilustrada na Figura 14.5 e pode ser resumidamente descrita pelo Algoritmo 14.1. A medida de concordância mencionada na linha 10 pode ser o índice Rand corrigido

(Hubert e Arabie, 1985), que será descrito na Seção 14.4. O nível de concordância entre as duas partições reflete a estabilidade do algoritmo de agrupamento.

Algoritmo 14.1 Análise de replicação

Entrada: Um conjunto de dados \mathbf{X}

Um algoritmo de agrupamento ϕ

Saída: Estabilidade do algoritmo ϕ

- 1 Obter duas amostras dos dados, \mathbf{X}_1 e \mathbf{X}_2 , dividindo o conjunto de dados original em dois subconjuntos
 - 2 Aplicar o algoritmo de agrupamento ϕ na amostra \mathbf{X}_1 , obtendo uma partição π^1
 - 3 Aplicar o algoritmo de agrupamento ϕ na amostra \mathbf{X}_2 , obtendo uma partição π^2
 - 4 Determinar os centroides dos *clusters* da partição π^1
 - 5 Construir π'_2 , agrupando \mathbf{X}_2 com base nas características de \mathbf{X}_1 :
 - 6 Assumir os mesmos centroides de π^1 para π'_2
 - 7 **para cada** $\mathbf{x} \in \mathbf{X}_2$ **faça**
 - 8 Determinar as distâncias entre o objeto \mathbf{x} e os centroides dos *clusters* de π'_2
 - 9 Associar \mathbf{x} ao *cluster* cujo centroide é mais próximo
// Isso resulta em um agrupamento da segunda amostra com base nas características da primeira
 - 10 **fim**
 - 11 Calcular uma medida de concordância entre os dois agrupamentos da segunda amostra, π^2 e π'_2
-

Figura de Mérito

Yeung et al. (2001) propõem uma metodologia para avaliação da qualidade de um agrupamento com base no poder preditivo do algoritmo que o gerou. O poder preditivo é dado pela medida figura de mérito, *FOM* (do inglês *Figure of Merit*), definida pelos autores. A *FOM* foi proposta para o agrupamento de genes utilizando seu nível de expressão medido em diversos experimentos. A intuição por trás dessa medida é a tendência de os genes pertencentes a um mesmo *cluster* terem um nível de expressão similar em experimentos adicionais, não empregados na geração dos *clusters*, se o agrupamento for significativo do ponto de vista biológico.

Um algoritmo de agrupamento é aplicado a todos os atributos, exceto um atributo, denominado a . Esse atributo a é utilizado para estimar o poder preditivo do algoritmo, por meio da variância dentro dos *clusters* dada pela *FOM*. *Clusters* significativos devem ter menos variação nos outros experimentos do que *clusters* gerados ao acaso. Assim, quanto maior a similaridade intracluster calculada a partir da exclusão do atributo a , mais forte é o poder preditivo e melhor o esquema de agrupamento. Sejam x_i^a o valor do atributo a para o objeto \mathbf{x}_i na matriz de dados bruta e $\bar{x}^a(\mathbf{x}^a, \mathbf{C}_l)$ a média dos valores do atributo \mathbf{x}^a nos objetos pertencentes ao *cluster* \mathbf{C}_l . $FOM(\mathbf{x}^a, \pi)$ é dada pela Equação 14.12, para k *clusters* usando o atributo \mathbf{x}^a .

$$FOM(\mathbf{x}^a, \pi) = \sqrt{\frac{1}{n} \sum_{l=1}^k \sum_{\mathbf{x}_i \in \mathbf{C}_l} (x_i^a - \bar{x}^a(\mathbf{x}^a, \mathbf{C}_l))^2} \quad (14.12)$$

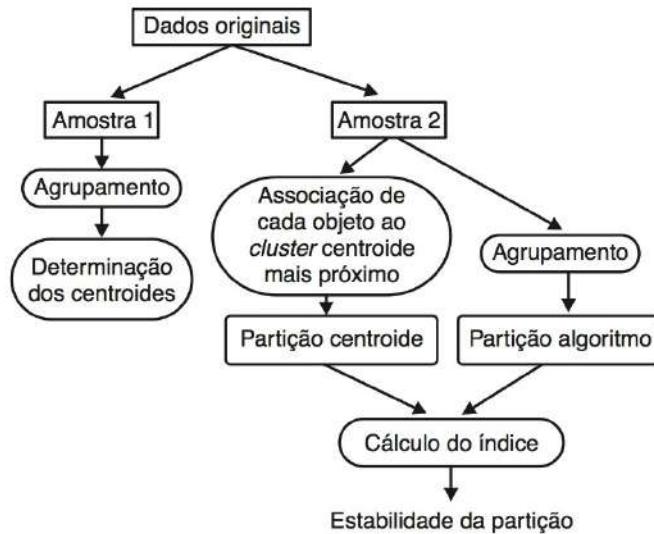


Figura 14.5 Análise de replicação para validação de um agrupamento.

Com a aplicação dessa equação a cada um dos d atributos, obtém-se a FOM agregada, dada pela Equação 14.13, que é uma estimativa do poder preditivo total de um algoritmo sobre todas as amostras para k *clusters*.

$$FOM(\pi) = \sum_{a=1}^d FOM(\mathbf{x}^a, \pi) \quad (14.13)$$

O valor da FOM tem a tendência de diminuir com o aumento do número de *clusters*. Para corrigir esse efeito, Yeung et al. (2001) utilizam a FOM ajustada, dada pela Equação 14.14.

$$FOM_{ADJ}(\pi) = \frac{\sum_{a=1}^d FOM(\mathbf{x}^a, \pi)}{\sqrt{(n - k)/n}} \quad (14.14)$$

Um valor baixo para as medidas FOM e FOM_{ADJ} indica um algoritmo com um poder preditivo elevado.

A metodologia proposta por Yeung et al. (2001) segue uma abordagem preditiva, ou seja, assume que o atributo excluído contém informação dos experimentos utilizados para produzir os *clusters*. Essa abordagem não é aplicável a todas as situações. Se todos os atributos contêm informações independentes, ela não se aplica. Além disso, não é uma abordagem segura para a comparação de agrupamentos com diferentes números de *clusters* ou obtidos com medidas de similaridade diferentes.

Variabilidade

Law e Jain (2003) propõem uma outra medida de validação para comparar os resultados de diferentes agrupamentos. Essa medida interpreta um algoritmo de agrupamento como um estimador estatístico e utiliza *bootstrapping* para estimar sua variabilidade. Segundo os autores, se uma partição é válida, sua variabilidade deve ser baixa.

Para isso, o algoritmo de agrupamento é considerado um estimador de ponto baseado em uma amostra \mathbf{X} de tamanho n , independente e identicamente distribuída. Assim,

o algoritmo é interpretado como um procedimento para estimar a melhor partição do conjunto de dados com base em uma amostra dos dados. O procedimento para o cálculo da variabilidade é apresentado no Algoritmo 14.2. A medida de variabilidade empregada nesse procedimento (linhas 7 e 9) é dada pela Equação 14.15, em $\Pi = \{\pi^1, \pi^2, \dots, \pi^B\}$ é um conjunto de B partições e $d(\pi^i, \pi^j)$ é uma medida de distância entre duas partições π^i e π^j . Law e Jain (2003) investigaram cinco medidas de distância diferentes para o cálculo da variabilidade: quatro delas baseadas nos índices Rand, Jaccard, Fowlkes e Malows e Hubert, descritos na Seção 14.4, e a medida de distância proposta em Lange et al. (2003). As quatro primeiras medidas são subtraídas de 1 para obter distâncias, uma vez que são medidas de similaridade entre duas partições.

$$V(\Pi) = \frac{1}{B(B-1)} \sum_{i=1}^B \sum_{j=i+1}^B d(\pi^i, \pi^j) \quad (14.15)$$

Algoritmo 14.2 Variabilidade

Entrada: Um conjunto de dados \mathbf{X}

Um algoritmo de agrupamento ϕ

Saída: Variabilidade do algoritmo ϕ

- 1 Gerar B amostras *bootstrap*, \mathbf{X}_b , $b = 1, \dots, B$, de tamanho n , reamostrando \mathbf{X} com reposição
 - 2 $\Pi_{alg} \leftarrow \emptyset$
 - 3 **para cada amostra \mathbf{X}_b faça**
 - 4 Aplicar o algoritmo de agrupamento ϕ na amostra, gerando a partição π^b
 - 5 $\Pi_{alg} \leftarrow \Pi_{alg} \cup \pi^b$
 - 6 **fim**
 - 7 Calcular a variabilidade $V_{alg} \leftarrow V(\Pi_{alg})$ de acordo com a Equação 14.15
 - 8 Gerar um conjunto de B partições aleatórias de tamanho n , Π_{ran}
 - 9 Calcular a variabilidade $V_{ran} \leftarrow V(\Pi_{ran})$ de acordo com a Equação 14.15
 - 10 Calcular a variabilidade ajustada $V \leftarrow V_{alg}/V_{ran}$
-

Assim como os algoritmos de AM em geral, todo algoritmo de agrupamento apresenta um viés. Em geral, existe uma explicação para esse viés, e uma partição com baixa variabilidade, mesmo se diferente dos *clusters* reais, provavelmente revela informações úteis sobre os dados (Law e Jain, 2003).

Esse método não identifica explicitamente se um conjunto de dados não apresenta estrutura, mas uma alta variabilidade para diferentes valores de k pode sugerir ausência de estrutura.

14.3 Critérios Internos

Os índices internos medem o grau em que uma partição obtida por um algoritmo de agrupamento representa uma estrutura presente nos dados, baseado apenas na matriz de objetos ou na matriz de similaridade. Eles medem o ajuste entre a partição gerada e os dados empregados. Esse tipo de validação, em geral, está relacionado à determinação do número “verdadeiro” de *clusters*.

A aplicação de índices internos para a validação de uma partição apresenta diversas dificuldades (Jain e Dubes, 1988; Halkidi et al., 2001). Uma delas decorre do fato de que os índices mais utilizados, como erro quadrático, provavelmente apresentam valores menores para dados que realmente possuem *clusters* do que para dados aleatórios, sem estrutura. Isso pode levar à conclusão errônea de que uma solução encontrada é válida, mesmo que ela não contenha o número correto de *clusters*. Uma possível solução é a aplicação dos índices internos como índices relativos, conforme já discutido. Uma outra dificuldade é a dependência desses índices dos valores utilizados para as características dos dados, tais como número de objetos, número de dimensões, número de *clusters* e espalhamento (Jain e Dubes, 1988).

Além dos índices relativos descritos anteriormente, que também podem ser empregados como internos, será apresentado, a seguir, o índice *Gap* (Tibshirani et al., 2001b). Um outro índice utilizado é o procedimento *Clest* (Dudoit e Fridlyand, 2002), que, para não tornar o texto muito extenso, não será detalhado neste livro.

14.3.1 Estatística *Gap*

O índice *Gap* (Tibshirani et al., 2001b), dado pela Equação 14.16, avalia a dispersão intracluster em relação à sua esperança sob uma distribuição de referência nula. Nessa equação, o valor de W_k é obtido pela Equação 14.17 e E_n^* é a esperança sob uma amostra de tamanho n para uma distribuição de referência. A variável D_r da Equação 14.17 é, por sua vez, definida pela Equação 14.18, que representa a dispersão intracluster.

Esse índice procura padronizar o gráfico de $\log(W_k)$ comparando-o com sua esperança sob uma distribuição de referência dos dados apropriada. A estimativa do número ótimo de *clusters* é o valor de k para o qual $\log(W_k)$ se situa o mais abaixo possível dessa curva de referência.

$$Gap_n(k) = E_n^*\{\log(W_k)\} - \log(W_k) \quad (14.16)$$

$$W_k = \sum_{r=1}^k \frac{1}{2n_r} D_r \quad (14.17)$$

$$D_r = \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_r} d(\mathbf{x}_i, \mathbf{x}_j) \quad (14.18)$$

O índice *Gap* foi proposto para estimar o número de *clusters* presentes em um conjunto de dados. O valor estimado de k é o valor que maximiza $Gap_n(k)$ após levar em consideração a distribuição da amostra.

É uma estimativa muito geral, aplicável a qualquer algoritmo de agrupamento e medida de distância. Para tornar a estatística *Gap* um procedimento operacional, é preciso encontrar uma distribuição referência apropriada e avaliar a distribuição amostral da estatística *Gap*. A população de referência pode ser determinada de duas formas:

- a. Gerar cada atributo de referência uniformemente no intervalo dos valores observados para aquele atributo ou
- b. Gerar os atributos de referência a partir de uma distribuição uniforme sobre um retângulo alinhado com os componentes principais dos dados.

Embora o método a seja mais simples, o método b leva em consideração a forma da distribuição dos dados e torna o procedimento invariante à rotação, se o método de agrupamento for invariante.

Para calcular a estatística Gap , estima-se $E_n^*\{\log(W_k)\}$ pela média de B cópias $\log(W_k^*)$, cada uma computada de uma amostra de Monte Carlo extraída da distribuição referência. Em seguida, é preciso avaliar a distribuição amostral da estatística Gap . Mais detalhadamente, o cálculo da estatística Gap pode ser feito conforme o Algoritmo 14.3.

Algoritmo 14.3 Cálculo da estatística Gap

Entrada: Um conjunto de dados \mathbf{X}

Um algoritmo de agrupamento ϕ

Saída: Estatística Gap

Melhor número de *clusters*

- 1 Geram-se B conjuntos de referência \mathbf{X}_b , utilizando o método a ou b, descritos anteriormente
- 2 **para cada** número de *clusters* $k = 1, 2, \dots, k_{max}$ **faça**
- 3 Aplica-se o algoritmo ϕ ao conjunto de dados \mathbf{X} , gerando a partição π^k
- 4 Calcula-se a medida de dispersão dentro dos *clusters* (W_k) para a partição π^k
- 5 **para cada** conjunto de referência \mathbf{X}_b **faça**
- 6 Aplica-se o algoritmo ϕ ao conjunto referência \mathbf{X}_b , gerando a partição π^{kb}
- 7 Calcula-se a medida de dispersão W_{kb}^* para a partição π^{kb}
- 8 **fim**
- 9 Computa-se a estatística Gap estimada, dada pela Equação 14.19
- 10

$$Gap_n(k) = \frac{1}{B} \sum_{b=1}^B \log(W_{kb}^*) - \log(W_k) \quad (14.19)$$

- 11 Computa o desvio padrão sd_k , dado pela Equação 14.20, em que
 $\bar{l} = (1/B) \sum_b \log(W_{kb}^*)$
- 12

$$sd_k = \sqrt{\frac{1}{B} \sum_{b=1}^B \{\log(W_{kb}^*) - \bar{l}\}^2} \quad (14.20)$$

- 13 Define-se $s_k = sd_k \sqrt{1 + 1/B}$
 - 14 **fim**
 - 15 Escolhe-se o melhor número de *clusters* como sendo o menor k tal que
 $Gap(k) \geq Gap(k+1) - s_{k+1}$
-

O cálculo da estatística Gap assume que há *clusters* uniformes bem separados. Quando existem *subclusters* menores dentro de *clusters* maiores bem separados, a estatística Gap pode exibir um comportamento não monotônico. Assim, é importante examinar a curva Gap inteira, em vez de simplesmente buscar o seu máximo.

14.4 Critérios Externos

O objetivo da validação externa é medir o quanto o agrupamento obtido confirma uma hipótese pré-especificada. Para isso são utilizados testes de hipótese, que testam se o valor do índice utilizado é incomumente grande ou pequeno, de acordo com uma distribuição de referência, conforme mencionado na Seção 14.1.

Conforme dito anteriormente, para utilizar os índices em critérios externos, aplicando testes estatísticos, é preciso conhecer suas funções de densidade de probabilidade sob a hipótese nula H_0 , que pressupõe estrutura aleatória do conjunto de dados. O cálculo da função de densidade de probabilidade desses índices é difícil. Uma alternativa comumente empregada é a aplicação de análise de Monte Carlo. O procedimento para a utilização de análise de Monte Carlo para determinar a distribuição referência de um índice externo sob a hipótese nula H_0 e responder se um agrupamento π^e é válido de acordo com ele é apresentado no Algoritmo 14.4 (Halkidi et al., 2002a; Jain e Dubes, 1988).

Algoritmo 14.4 Procedimento para análise de Monte Carlo

Entrada: Um conjunto de dados \mathbf{X}

Um algoritmo de agrupamento ϕ

Uma partição real π^r

Uma partição obtida com o algoritmo ϕ , π^e

- 1 Gerar B conjuntos de dados \mathbf{X}_b , $b = 1, \dots, B$, de tamanho n , com a mesma dimensão do conjunto de dados original \mathbf{X} , seguindo uma distribuição uniforme (Para um nível de significância α de 0,05, Jain e Dubes (1988) indicam a utilização de $B = 100$).

2 para cada conjuntos de dados \mathbf{X}_b faça

- 3 Gerar uma partição aleatória π^{rb} , associando cada objeto do conjunto a um dos *clusters* existentes em π^r , aleatoriamente e de forma a que cada *cluster* tenha o mesmo tamanho dos *clusters* em π^r

- 4 Aplicar ϕ ao conjunto \mathbf{X}_b , gerando a partição π^{eb}

- 5 Calcular o índice escolhido, $ind(\pi^{eb}, \pi^{rb})$, entre as partições π^{eb} e π^{rb}

6 fim

- 7 Calcular o valor do índice entre as partições π^e e π^r , $ind(\pi^e, \pi^r)$

- 8 Criar um gráfico de dispersão para os B valores do índice, $ind(\pi^{eb}, \pi^{rb})$ (esse gráfico é uma aproximação da função de densidade de probabilidade do índice)

- 9 Comparar o índice calculado para a partição π^e , $ind(\pi^e, \pi^r)$, aos valores do gráfico, considerando um dado nível de significância. Se s valores de $ind(\pi^{eb}, \pi^{rb})$ são maiores que o valor de $ind(\pi^e, \pi^r)$, com $s = (1 - alpha)B$, a hipótese nula é aceita, indicando que os dados são distribuídos aleatoriamente (Halkidi et al., 2002a)

Os índices externos mais tradicionais, frequentemente utilizados na validação de partições, comparam uma partição resultante da aplicação de um algoritmo, π^e , a uma partição independente dos dados, construída com base na intuição ou conhecimento *a priori* sobre a estrutura real dos dados, π^r . São os casos da estatística Rand (R), descrita na Seção 14.4.1, do coeficiente de Jaccard (J), descrito na Seção 14.4.2, do índice de Fowlkes e

Mallows (*FM*), descrito na Seção 14.4.3, e da estatística Hubert's Γ normalizada, descrita na Seção 14.4.4.

Além das formas tradicionais dos índices, equações corrigidas são frequentemente empregadas. Uma correção bastante comum é a normalização do índice para que ele apresente o valor 0 quando uma partição é selecionada ao acaso e 1 quando uma partição casa perfeitamente com a partição real (Jain e Dubes, 1988; Gordon, 1999). Um dos índices mais populares empregados em validação externa é o índice Rand corrigido, apresentado na Seção 14.4.5.

Além desses índices bastante tradicionais, outros foram propostos com o mesmo objetivo, como é o caso do variação de informação (Meila, 2007), que é um índice baseado na Teoria da Informação, e o índice proposto por Dom (2002). Como representante dos índices mais recentes, a variação de informação será descrita na Seção 14.4.6.

Os índices R , J , FM e Γ se baseiam nas seguintes informações a respeito da relação entre os objetos de π^e e π^r . Um par de objetos, $(\mathbf{x}_i, \mathbf{x}_j)$, é dito:

- SS: se pertencem ao mesmo *cluster* de π^e e ao mesmo *cluster* de π^r .
- SD: se pertencem ao mesmo *cluster* de π^e e a *clusters* diferentes de π^r .
- DS: se pertencem a *clusters* diferentes de π^e e ao mesmo *cluster* de π^r .
- DD: se pertencem a *clusters* diferentes de π^e e a *clusters* diferentes de π^r .

Sejam $a1$, $a2$, $a3$ e $a4$ os números de pares SS, SD, DS e DD, respectivamente. Define-se $M = a1 + a2 + a3 + a4$ como o número máximo de todos os pares no conjunto de dados ($M = n(n - 1)/2$). Definem-se também $m_1 = a1 + a2$ e $m_2 = a1 + a3$.

A seguir, são descritos alguns dos índices mais tradicionalmente empregados em validação externa. Além das formas tradicionais desses índices, equações corrigidas também são frequentemente empregadas. Os índices mais tradicionais são também comparados na Seção 14.4.7.

14.4.1 Índice Rand

O índice Rand, definido pela Equação 14.21, computa a probabilidade de que dois objetos pertençam ao mesmo *cluster* ou pertençam a *clusters* diferentes nas duas partições π^e e π^r (Boutin e Hascoët, 2004).

$$R(\pi^e, \pi^r) = \frac{(a1 + a4)}{M} \quad (14.21)$$

14.4.2 Índice Jaccard

Esse índice computa a probabilidade de que dois objetos pertencentes ao mesmo *cluster* em uma das partições também pertençam ao mesmo *cluster* na outra partição (Boutin e Hascoët, 2004). O coeficiente de Jaccard (J) é dado pela Equação 14.22.

$$J(\pi^e, \pi^r) = \frac{a1}{(a1 + a2 + a3)} \quad (14.22)$$

14.4.3 Índice de Fowlkes e Mallows

O índice de Fowlkes e Mallows (FM) é dado pela Equação 14.23. Os maiores valores desse índice indicam semelhança entre as duas partições, e variam no intervalo $[0, 1]$.

$$FM(\pi^e, \pi^r) = \frac{a1}{\sqrt{(m_1)(m_2)}} \quad (14.23)$$

14.4.4 Índice Hubert normalizado

A estatística Hubert's Γ normalizada, dada pela Equação 14.24, mede o grau de correspondência linear entre duas partições. Os valores desse índice variam no intervalo $[-1, 1]$, e valores absolutos excepcionalmente grandes de Γ indicam que as duas partições concordam, o que deve ser estabelecido com algum teste de hipótese, conforme mencionado anteriormente (Seção 14.1).

$$\Gamma(\pi^e, \pi^r) = \frac{Ma1 - m_1m_2}{\sqrt{m_1m_2(M - m_1)(M - m_2)}} \quad (14.24)$$

14.4.5 Índice Rand Corrigido

A correção da estatística Rand, proposta por Hubert e Arabie (1985), resulta na estatística Rand corrigida (CR), dada pela Equação 14.25, em que n_{ij} é o número de objetos comuns aos *clusters* \mathbf{C}_i de π^e e \mathbf{C}_j de π^r , $n_{i\cdot}$ é o número de objetos no *cluster* \mathbf{C}_i de π^e , $n_{\cdot j}$ é o número de objetos no *cluster* \mathbf{C}_j de π^r e k^e e k^r são os números de *clusters* nas partições π^e e π^r , respectivamente. Esse índice é um dos mais utilizados para validação externa em agrupamentos.

$$CR(\pi^e, \pi^r) = \frac{\sum_{i=1}^{k^e} \sum_{j=1}^{k^r} \binom{n_{ij}}{2} - \left[\sum_{i=1}^{k^e} \binom{n_{i\cdot}}{2} \sum_{j=1}^{k^r} \binom{n_{\cdot j}}{2} \right] / \binom{n}{2}}{\left[\sum_{i=1}^{k^e} \binom{n_{i\cdot}}{2} + \sum_{j=1}^{k^r} \binom{n_{\cdot j}}{2} \right] / 2 - \left[\sum_{i=1}^{k^e} \binom{n_{i\cdot}}{2} \sum_{j=1}^{k^r} \binom{n_{\cdot j}}{2} \right] / \binom{n}{2}} \quad (14.25)$$

O índice Rand corrigido varia no intervalo $[-1, 1]$, e valores menores ou próximos a 0 indicam que a semelhança entre as partições se deve ao acaso; o valor 1 indica que as partições são idênticas (Hubert e Arabie, 1985). Apesar de o menor valor possível do índice ser -1 , esse valor não é atingido. Em geral, partições bastante diferentes resultam em um valor próximo de 0. Poderia ser apropriado ter o limite inferior 0 bem definido, mas a normalização necessária não oferece vantagens práticas, uma vez que valores negativos do índice não têm aplicação prática (Hubert e Arabie, 1985).

Uma outra correção muito utilizada é a correção sugerida por Morey e Agresti (1984) e utilizada por Milligan e Cooper (1986). Entretanto, segundo Hubert e Arabie (1985), essa correção contém um erro, pois assume inadequadamente que a esperança de uma variável aleatória ao quadrado é o quadrado da esperança.

Uma extensão *fuzzy* para o índice Rand corrigido e outros pode ser encontrada em Campello (2007).

14.4.6 Índice variação de informação

A variação de informação (VI), dada pela Equação 14.26, mede a quantidade de informação perdida ou ganha na mudança da partição π^e para π^r . Nessa equação, $H(\pi^a)$ é a entropia de uma partição π^a , dada pela Equação 14.27, e $I(\pi^a, \pi^b)$ é a informação mútua compartilhada entre as partições π^a e π^b , dada pela Equação 14.28, com $p(i) = \frac{|\mathbf{C}_i^a|}{n}$ e $p(i, j) = \frac{|\mathbf{C}_i^a \cap \mathbf{C}_j^b|}{n}$. Esse índice tem o valor 0 como menor valor, não sendo limitado superiormente. O valor 0 indica que as partições são idênticas.

$$VI(\pi^e, \pi^r) = H(\pi^e) + H(\pi^r) - 2I(\pi^e, \pi^r) \quad (14.26)$$

$$H(\pi^a) = - \sum_{i=1}^{k^a} p(i) \log(p(i)) \quad (14.27)$$

$$I(\pi^a, \pi^b) = \sum_{i=1}^{k^a} \sum_{j=1}^{k^b} p(i, j) \log\left(\frac{p(i, j)}{p(i)p(j)}\right) \quad (14.28)$$

14.4.7 Comparaçāo dos índices tradicionais para validação externa

Os índices R , J , FM , Γ e CR são os mais utilizados para validação externa. A maioria deles pode ser sensível ao número de *clusters* de uma partição ou à distribuição dos objetos nos *clusters* (Filho, 2003). Os índices Γ e R têm a tendência de apresentar valores mais elevados para partições com maior número de *clusters* (Milligan et al., 1983). O índice J apresenta valores mais elevados para partições com menor número de *clusters*. Já o índice CR não tem essa influência do número de *clusters*.

Valores altos de R , FM e Γ indicam forte concordância entre as duas partições π^e e π^r . A estatística Γ é uma correlação, e seu valor está compreendido no intervalo $[-1, 1]$. CR também apresenta valores no intervalo $[-1, 1]$; entretanto, raramente valores próximos de -1 são produzidos. R e J possuem valores no intervalo $[0, 1]$, e o valor máximo 1 não é atingível quando as duas partições têm números de *clusters* diferentes.

Os índices Rand, Jaccard e Fowlkes e Mallows têm como limite inferior o valor 0, mas na prática nunca vão produzir esse valor em um agrupamento de dados real. Milligan e Cooper (1986) confirmaram que o índice CR apresenta valores próximos de 0 quando os *clusters* são gerados a partir de dados aleatórios.

Nas investigações de Milligan et al. (1983) sobre alguns desses índices, foi encontrado um alto grau de consistência. Esses autores encontraram alta similaridade entre os índices J e FM , talvez devido ao fato de ambos utilizarem o mesmo numerador (número de pares de objetos que foram apropriadamente agrupados pelo algoritmo). Da mesma forma, os índices R e CR se comportaram de forma semelhante, o que é razoável, uma vez que a estatística Rand corrigida (CR) é apenas um ajuste da estatística R original. Milligan et al. (1983) recomendam a utilização de apenas um índice de cada par de índices similares, apontando para o CR e o J como as melhores escolhas de cada grupo.

14.5 Considerações Finais

Neste capítulo foram discutidas questões relativas ao planejamento de experiências e à avaliação de modelos descritivos, mais especificamente de abordagens para análise de agrupamento. Assim, tal como para os modelos preditivos, esses aspectos são de grande relevância para a realização de experimentos com algoritmos de agrupamento de dados, de forma a garantir a validade dos resultados alcançados, ou para se escolher o melhor modelo para uso em um determinado problema.

Resumidamente, foram detalhados os três tipos de critérios existentes para avaliação de agrupamentos, que são os critérios relativos, apropriados para comparar diversos agrupamentos com respeito a algum aspecto, os critérios internos, que avaliam a qualidade de um agrupamento de acordo com alguma propriedade existente nos dados originais e os critérios externos, que permitem o confronto do agrupamento obtido por um algoritmo com uma estrutura dos dados previamente conhecida.

Além disso, foram discutidos diversos índices que podem ser aplicados com cada critério, diferentes maneiras como esses índices podem ser utilizados, bem como várias abordagens recentes que consideram aspectos como estabilidade das partições geradas.

Parte IV

Tópicos Avançados

Tópicos Avançados

O futuro da área de AM aponta para sistemas autônomos que podem incorporar o conhecimento, aprender a partir de dados não estacionários distribuídos em ambientes dinâmicos, com capacidade de transferir o conhecimento entre os problemas de aprendizado. Neste capítulo apresentamos algumas linhas de pesquisa em AM que lidam com diversos tópicos avançados.

No Capítulo 15 são tratados os problemas de aprendizado em fluxos de dados. A maioria dos algoritmos estudados neste livro requer que todos os dados de treinamento estejam disponíveis em memória. No entanto, o aumento exponencial de dados armazenados em grandes bases de dados, geralmente recolhidos ao longo do tempo, durante meses ou anos, levanta novos problemas. Além disso, em algumas situações, como redes de sensores, tráfego TCP / IP, *e-commerce* e *web sites*, geram fluxos de dados em alta velocidade. Não é mais possível armazenar todos os dados na memória e executar várias passagens sobre os dados de treinamento. Para aumentar a dificuldade do problema de aprendizado, o conceito a ser aprendido pode mudar ao longo do tempo. Tudo isso demanda a realização de pesquisas sobre escalonamento de algoritmos de AM, que permita o aprendizado sequencial a partir de fluxos de dados em ambientes dinâmicos (Domingos e Hulten, 2000; Gama et al., 2003; Aggarwal et al., 2003).

Um dos problemas atuais na pesquisa em AM é o problema de seleção de algoritmos e de modelos. Essa questão pode ser resumida como: *Para um dado conjunto de dados, qual é o algoritmo/modelo mais adequado?* Qualquer algoritmo de AM tem sua própria área ou nicho de aplicação: existem problemas para os quais um algoritmo apresenta uma boa generalização, enquanto para outros problemas ele não mostra uma boa capacidade de generalização. Uma área de pesquisa promissora é a de meta-aprendizado (Brazdil et al., 2009). Meta-aprendizado caracteriza a área de aplicação de algoritmos de aprendizado, sendo útil para seleção de algoritmos e de modelos em problemas semelhantes. Esse tópico é coberto no Capítulo 16.

A maioria das aplicações de AM em problemas de classificação de dados reportadas na literatura se dá em problemas com apenas duas classes. Por causa disso, vários algoritmos de AM foram desenvolvidos para lidar apenas com esses tipos de problema. No entanto, existem vários problemas reais que apresentam mais de duas classes, conhecidos como problemas de classificação multiclasse. Como será visto no Capítulo 17, as estratégias mais eficientes para lidar com esses problemas decompõem o problema multiclasse original em vários problemas de classificação binária.

Em geral, classificadores induzidos por algoritmos de AM para problemas multiclasse atribuem uma única classe a cada objeto. Entretanto, existem problemas reais de classificação em que um objeto pode pertencer simultaneamente a mais de uma classe. Esses problemas, conhecidos como problemas de classificação multirrotulo, ocorrem com frequência

nas áreas de bioinformática e processamento de textos. Um exemplo é a classificação da função de proteínas, em que uma proteína pode ter mais de uma função. O Capítulo 18 aborda essa problemática.

Deve ser observado ainda que diversos problemas multiclasse podem ser mais bem representados utilizando uma estrutura hierárquica, em que o processo de classificação pode classificar novos objetos em diferentes níveis de uma hierarquia. Muitos problemas de classificação hierárquica são também problemas de classificação multirrótulo. Técnicas de classificação hierárquica, que serão descritas no Capítulo 19, apresentam alternativas para lidar com esses tipos de problemas.

Existem várias técnicas de AM, como as RNAs, que são inspiradas em processos que ocorrem na natureza. Essas técnicas, estudadas em uma área denominada Computação Natural, têm sido aplicadas a vários problemas de AM descriptivo e preditivo. No Capítulo 20 são apresentadas algumas técnicas de Computação Natural, com maior ênfase naquelas que possuem inspiração biológica, como Computação Evolutiva, Inteligência de Enxames e Sistemas Imunológicos Artificiais.

Ao longo do livro (com raras exceções, como os sistemas de CBR), temos assumido uma representação atributo-valor para as observações. Cada exemplo é descrito por um conjunto de variáveis (um registo na terminologia de base de dados) e um conjunto de exemplos é armazenado em uma matriz (a relação ou tabela na terminologia de Banco de Dados). Em algumas das mais desafiadoras aplicações de aprendizado de máquina, os dados são descritos por sequências (por exemplo, dados de DNA), árvores (documentos XML) e gráficos (componentes químicos, análise de rede).

Capítulo 15

Aprendizado em Fluxos Contínuos de Dados

Hoje em dia, a pesquisa e a prática de AM são confrontadas com novos problemas e desafios. A forma de coletar dados já não é manual, mas sim automática. Temos sensores e computadores que enviam informações para outros computadores. Em algumas aplicações, como as que surgem em redes de sensores, a melhor forma de processar informação não é com base em tabelas persistentes, mas sim com base em fluxos transitórios de dados. Por vezes, não é até mesmo viável armazenar toda a informação em sistemas de gerenciamento de bases de dados tradicionais, principalmente porque estes não estão projetados para suportar diretamente consultas que precisam ocorrer de forma contínua (Babcock et al., 2002).

Nas últimas décadas, a pesquisa e a prática de AM têm focado o aprendizado por lotes, geralmente utilizando pequenos conjuntos de dados. No aprendizado *offline*, todos os dados do conjunto treinamento estão disponíveis em memória, e um algoritmo aprende um modelo de decisão após iterativamente processar todos os dados. A lógica por trás dessa prática é que os exemplos são gerados aleatoriamente de acordo com alguma distribuição estacionária.

Os desafios no aprendizado em fluxos contínuos de dados relacionam-se com a capacidade de processar grandes volumes de informação que evoluem ao longo do tempo e que são gerados por distribuições não estacionárias. Nesses contextos, o objetivo consiste em manter, de forma permanente, um modelo de decisão preciso e consistente com o estado atual do processo que gera dados. Esse problema requer algoritmos de aprendizado incrementais, que podem modificar o modelo atual sempre que alguma nova informação é disponibilizada. Nesse contexto, o pressuposto de que os exemplos são gerados aleatoriamente de acordo com uma distribuição estacionária não se verifica. Na presença de uma distribuição não estacionária, o sistema de aprendizado deve incorporar mecanismos de esquecimento, de forma que permitam descartar informações e conceitos que já não refletem o estado atual do problema. Aprender a partir de dados fluxos requer algoritmos de aprendizado incrementais, que funcionam com recursos computacionais limitados, e capazes de levar em consideração mudanças e evoluções dos conceitos a aprender.

Neste Capítulo serão enumerados os principais desafios no aprendizado em fluxos contínuos de dados. Na Seção 15.2 são apresentados algoritmos para indução de árvores de decisão e análise de agrupamento para fluxos contínuos de dados. A Seção 15.3 trata do problema de dados não estacionários, um dos tópicos que mais diferencia o aprendizado *offline* do aprendizado em tempo real. O capítulo termina com a identificação de problemas em aberto e linhas de potencial investigação.

15.1 Desafios no Aprendizado em Fluxos Contínuos de Dados

O aprendizado a partir de fluxos contínuos de dados produzidos a alta velocidade em ambientes dinâmicos e não estacionários requer novas técnicas de amostragem, novos algoritmos, capacidade de processar dados na velocidade em que são disponibilizados e capacidade de esquecer dados desatualizados. Domingos e Hulten (2001) identificam as propriedades desejáveis para esses sistemas de aprendizado:

1. Incrementalidade;
2. Aprendizado em tempo real;
3. Capacidade para processar exemplos em tempo constante e usando memória limitada;
4. Acesso limitado a exemplos já processados;
5. Capacidade de detectar e adaptar o modelo de decisão a mudanças do conceito.

Exemplos de algoritmos de aprendizado projetados para processar fluxos de dados incluem:

- Aprendizado preditivo
 - Árvores de decisão: Domingos e Hulten (2000); Gama et al. (2003); Jin e Agrawal (2003);
 - Regras de decisão: Ferrer-Troyano et al. (2005);
- Aprendizado descritivo
 - Variantes do algoritmo k -médias: Zhang et al. (1996); Sheikholeslami et al. (1998);
 - *Micro Clustering*: Callaghan et al. (2002); Aggarwal et al. (2003);
 - Agrupamento hierárquico de séries temporais: Rodrigues et al. (2006);
- Regras de associação
 - *Itemsets* frequentes: Han et al. (2000);
 - Padrões frequentes: Giannella et al. (2003);
- Detecção de novidade: Markou e Singh (2003); Spinsosa et al. (2008);
- Redução de dimensionalidade: Sousa et al. (2007).

Todos esses algoritmos compartilham propriedades, como processamento de exemplos na velocidade em que chegam, com restrições de tempo e memória. Eles também mantêm modelos de decisão dinâmicos que evoluem com os dados, que podem ser utilizados em qualquer momento do processo de aprendizado, e são capazes de adaptar o modelo de decisão aos dados mais recentes.

15.2 Algoritmos Ilustrativos para Aprendizado em Fluxos de Dados

Para esclarecer como funciona o aprendizado em fluxos de dados, apresentamos três exemplos ilustrativos de algoritmos de aprendizado que continuamente mantêm um modelo de decisão que evolui ao longo do tempo. No primeiro caso, apresentamos um algoritmo de indução de árvores de decisão. Como segundo exemplo, é apresentado um algoritmo de agregação hierárquica de séries temporais. É um algoritmo projetado para lidar com milhares de séries temporais que fluem em alta velocidade. O terceiro exemplo é uma adaptação simples de redes neurais, adaptadas para aprender em fluxos de dados a alta velocidade.

15.2.1 O Algoritmo *Árvore de Decisão Muito Rápida*

Aprender a partir de grandes conjuntos de dados pode ser mais eficaz quando se utilizam algoritmos que dão maior ênfase no gerenciamento de viés. Um desses algoritmos é o sistema árvore de decisão muito rápida (VFDT, do inglês *Very Fast Decision Tree*) (Domingos, 1997b), apresentado no Algoritmo 15.1.

VFDT é um algoritmo de aprendizado de árvore de decisão que dinamicamente ajusta o seu viés sempre que novos exemplos se tornam disponíveis. Na indução de árvores de decisão, a questão principal é a decisão de quando expandir a árvore, instalando um teste de divisão e gerando novas folhas. A ideia básica do VFDT consiste na utilização de um pequeno conjunto de exemplos para selecionar o teste de divisão a ser incorporado em um nó da árvore de decisão. Se depois de considerar um conjunto de exemplos a diferença do mérito entre os dois melhores testes de divisão não satisfaz um teste estatístico (o limite de Hoeffding), o VFDT procede examinando mais exemplos. O VFDT apenas toma uma decisão (isto é, adiciona um teste de divisão nesse nó) quando há evidência estatística suficiente em favor de um teste particular. Essa estratégia garante estabilidade ao modelo (baixa variância) e controla superajustamentos, embora possa alcançar um aumento no número de graus de liberdade (baixo viés) com aumentos do número de exemplos.

No VFDT, uma árvore de decisão é construída recursivamente substituindo folhas por nós de decisão. Cada folha armazena as estatísticas suficientes sobre os valores dos atributos. As estatísticas suficientes são aquelas necessárias para a função de avaliação heurística que computa o mérito dos testes de divisão de acordo com os valores dos atributos. Quando um exemplo está disponível, ele percorre a árvore da raiz até uma folha, avaliando o atributo apropriado para cada nó e seguindo o ramo correspondente ao valor do atributo no exemplo. Quando o exemplo alcança uma folha, as estatísticas suficientes são atualizadas. Então, cada condição possível baseada no valor do atributo é avaliada. Se há um suporte estatístico suficiente em favor de um teste sobre os outros, o nó é mudado para um nó de decisão. O novo nó de decisão terá tantos nós descendentes quanto o número de valores possíveis para o atributo escolhido (entretanto, essa árvore não é necessariamente binária). Os nós de decisão somente mantêm a informação sobre o teste de divisão instalado neles.

A principal inovação do sistema VFDT é o uso do limite de Hoeffding para decidir quantos exemplos devem ser observados antes de instalar um teste de divisão em um nó. Suponha que tenham sido feitas n observações independentes de uma variável aleatória r

Algoritmo 15.1 O algoritmo da árvore de Hoeffding

Entrada: Uma sequência de exemplos de treinamento $\mathbf{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, \infty\}$
 Uma função de avaliação de divisão $H(\cdot)$
 Número mínimo de exemplos N_{min}
 δ : 1 menos a probabilidade desejada de escolher o atributo correto em qualquer nó.
 τ : Constante usada para desempate.

Saída: HT : Árvore de Decisão

```

1 Seja  $HT \leftarrow$  Folha Vazia (Raiz) ;
2 para cada exemplo  $(\mathbf{x}_i, y_i) \in \mathbf{D}$  faça
3   Atravessar a árvore  $HT$  a partir da raiz até a folha  $l$ ;
4   Atualizar as estatísticas suficientes em  $l$ ;
5   se  $O$  número de exemplos em  $l$  é maior que  $N_{min}$  então
6     Calcular  $H(at_i)$  para todos os atributos ;
7     Seja  $at_a$  o atributo com maior  $H$  ;
8     Seja  $at_b$  o atributo com o segundo maior  $H$  ;
9     Calcular  $\epsilon$  (limite de Hoeffding) ;
10    se  $(H(at_a) - H(at_b)) > \epsilon$  então
11      Substituir  $l$  por um teste de divisão baseado no atributo  $at_a$  ;
12      Adicionar uma nova folha vazia para cada possível valor de  $at_a$  ;
13    fim
14  senão
15    se  $\epsilon < \tau$  então
16      Substituir  $l$  por um teste de divisão baseado no atributo  $at_a$  ;
17      Adicionar uma nova folha para cada possível valor de  $at_a$  ;
18    fim
19  fim
20 fim
21 fim
```

cujo intervalo é R . O limite de Hoeffding diz que a média verdadeira de r, \bar{r} , é pelo menos $\bar{r} - \epsilon$, em que $\epsilon = \sqrt{R^2 \frac{\ln(\frac{1}{\delta})}{2n}}$, com probabilidade $1 - \delta$.

Seja $H(\cdot)$ a função de avaliação de um atributo. Para o ganho de informação, o intervalo R , de $H(\cdot)$, é $\log_2(k)$, em que k denota o número de classes. Sejam at_a o atributo com o maior $H(\cdot)$, at_b o atributo com o segundo maior $H(\cdot)$ e $\overline{\Delta H} = \overline{H}(at_a) - \overline{H}(at_b)$ a diferença entre os dois melhores atributos. Então, se $\overline{\Delta H} > \epsilon$ com n exemplos observados na folha, o limite de Hoeffding diz que, com probabilidade $1 - \delta$, at_a é realmente o atributo com o maior valor na função de avaliação. Nesse caso, a folha deve ser transformada em um nó de decisão que divide em at_a .

A avaliação da função de mérito para cada exemplo pode ser muito cara. Porém, não é eficiente calcular $H(\cdot)$ toda vez que um exemplo é observado. O VFDT só calcula a função de avaliação do atributo $H(\cdot)$ quando um número mínimo de exemplos já foi observado desde a última avaliação. Esse número mínimo de exemplos é um parâmetro definido pelo usuário. Quando dois ou mais atributos continuamente têm valores muito similares de $H(\cdot)$, mesmo com um número grande de exemplos, o limite de Hoeffding não decidirá

entre eles. Para resolver esse problema, o VFDT usa uma constante τ introduzida pelo usuário. Por exemplo, se $\overline{\Delta H} < \epsilon < \tau$, então a folha é transformada em um nó de decisão. O teste de divisão é baseado no melhor atributo.

Mais tarde, os mesmos autores apresentaram o algoritmo CVFDT (do inglês, *Concept-adapting Very Fast Decision Tree*) (Hulten et al., 2001), uma extensão do VFDT desenvolvida para fluxos de dados que mudam com o tempo. O CVFDT gera árvores de decisão alternativas nos nós em que há evidência de que o teste de divisão previamente instalado não é mais apropriado, tendo em conta os dados mais recentes. O sistema substitui a árvore antiga pela árvore alternativa, quando a última se torna mais precisa.

15.2.2 Análise de Agrupamentos em Séries Temporais Contínuas

A maior parte do trabalho em agrupamento incremental de fluxos contínuos de dados concentra-se em agrupamento de exemplos e não em agrupamento de variáveis (atributos). Dada uma matriz estática de dados, a diferença entre agrupamento de exemplos ou agrupamento de variáveis não é relevante. Podemos usar exatamente os mesmos algoritmos de análise de agrupamentos diretamente sobre a matriz de dados ou sobre a transposta da matriz.

Para o processamento em tempo real de fluxos de dados, a matriz de dados não é estática. Ela evolui com o tempo à medida que novos exemplos se tornam disponíveis. Nesse contexto, a transposta da matriz de dados é um operador de bloqueio.¹ Por esse motivo e no contexto de fluxos de dados, as técnicas de análise de agrupamentos de variáveis (séries temporais) requerem algoritmos diferentes dos algoritmos de análise de agrupamento de exemplos.

Um dos primeiros algoritmos para análise de agrupamento de variáveis em fluxos contínuos de dados é o ODAC (do inglês, *Online Divisive-Agglomerative Clustering*) (Rodrigues et al., 2006). O ODAC é um sistema de agrupamento de séries temporais que constrói uma hierarquia de grupos de forma incremental e divisiva. As folhas são os grupos resultantes, com um conjunto de variáveis em cada folha. A união das folhas é o conjunto total de variáveis. A interseção das folhas é o conjunto vazio, ou seja, cada variável pertence a um e apenas um dos grupos. O sistema calcula a distância entre variáveis de forma incremental e executa procedimentos de expansão e agregação da estrutura, baseado em um intervalo de confiança dado pelo limite de Hoeffding. O ODAC usa dois operadores para manter a estrutura de grupos consistente com a informação mais recente. O operador de *divisão* transforma uma folha f em não folha com duas folhas descendentes. As variáveis em f são distribuídas pelas novas folhas descendentes, formando uma estrutura de grupos mais detalhada. Esse operador é aplicado sempre que a informação acumulada numa folha é suficiente para formar grupos mais detalhados. O segundo operador é um operador de *agregação*. Esse operador agrupa duas folhas numa única folha. O operador é aplicado quando o sistema detecta que a estrutura de correlação entre as variáveis numa das folhas não é consistente com a estrutura de correlação observada no nó que gerou essa folha. Esse operador permite adaptar a estrutura de grupos quando são observadas mudanças na estrutura de correlação entre variáveis do processo que gera dados. A Figura 15.1(a) ilustra a estrutura de grupos no instante t . Inicialmente todas

¹Um operador de bloqueio só retorna resultados depois de processar toda a informação na entrada.

as variáveis estão no mesmo grupo, o grupo 1. O grupo 1 foi expandido dando origem aos grupos 2 e 3. Posteriormente, o grupo 2 foi expandido, dando origem aos grupos 3 e 4. A atual estrutura de grupos é dada pelas folhas 4, 5, e 3. O exemplo recebido no instante t , que contém o valor de todas as variáveis nesse momento, vai atualizar as estatísticas suficientes nas folhas da estrutura. A Figura 15.1(b) ilustra as diferentes janelas temporais definidas sobre o fluxo de dados. O nó 1 recebeu informação da parte inicial do fluxo de dados. Quando o nó 1 é expandido, os exemplos vão atualizar os nós descendentes. Em cada instante, as folhas da estrutura recebem a informação mais recente.

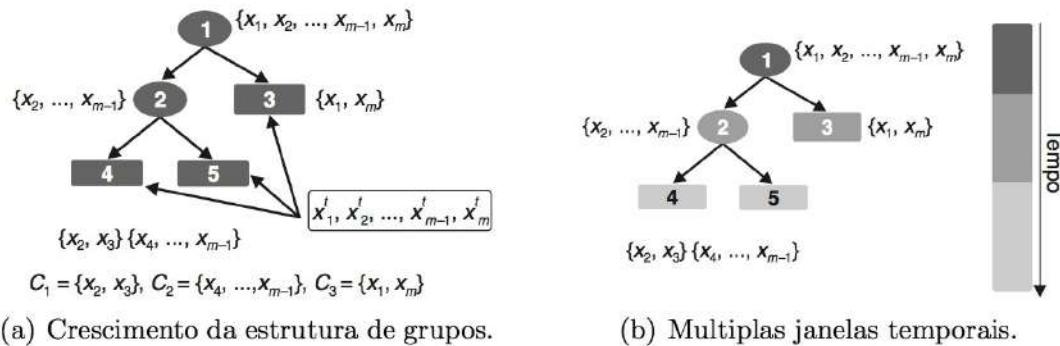


Figura 15.1 ODAC: Análise de Agrupamentos de Séries Temporais.

A medida de dissimilaridade é o *Rooted Normalized One-Minus-Correlation*. De acordo com essa medida, a dissimilaridade entre as variáveis at_a e at_b é definida por: $rnomc(at_a, at_b) = \sqrt{(1 - corr(at_a, at_b))/2}$, em que $corr(at_a, at_b)$ é a correlação de Pearson (Equação 11.6). Essa medida é calculada de forma incremental, e cada exemplo é processado apenas uma vez. A atualização em tempo real das estatísticas suficientes necessárias ao cálculo da matriz de dissimilaridade requer calcular: as somas das variáveis, as somas dos quadrados das variáveis e as somas das multiplicações par a par entre as variáveis. A matriz de distâncias para cada folha é calculada quando a folha é testada para divisão ou agregação, após receber um número mínimo de exemplos, dado pelo limite de Hoeffding. Em cada folha, se a diferença entre a máxima distância entre duas variáveis e a segunda maior distância for estatisticamente significativa, verificada pelo limite de Hoeffding, então a maior distância é o diâmetro da folha, estando ela pronta para os testes de divisão e agregação.

O critério de divisão do ODAC possui dois conceitos inerentes: quanto mais afastadas estiverem a máxima e mínima distâncias de um grupo, maior a possibilidade de ocorrer uma divisão; e também quanto mais afastada estiver a distância média da média entre as distâncias máxima e mínima, maior a probabilidade de divisão. De forma a distinguir os casos em que o grupo tem todas as variáveis equidistantes dos casos em que possui duas ou mais variáveis muito distantes, foi introduzido um parâmetro no sistema, τ , que determina o tempo dado ao sistema para descobrir o verdadeiro diâmetro, até que seja forçado o teste de divisão. Quando um ponto de divisão é reportado, as duas variáveis que definem o diâmetro, designadas *pivô*, vão dar origem a dois novos grupos. Cada uma das variáveis restantes vai para o novo grupo que contém o *pivô* mais próximo. As novas folhas começam novas computações, reiniciando as estatísticas suficientes.

Considerando a presença de alterações de conceito nos dados, o critério de agregação é baseado na comparação entre os diâmetros dos nós descendentes e o diâmetro do nó ascendente. O critério de divisão garante que sempre que a estrutura de grupos é expandida,

o diâmetro dos grupos deve diminuir. Observando que o diâmetro de uma folha é significativamente maior que o diâmetro do nó ascendente, significa uma mudança na estrutura de correlação entre as variáveis nessa folha. Por esse motivo, as folhas descendentes desse nó são agregadas, e as estatísticas suficientes nesse nó são reiniciadas.

Apesar de os requisitos de espaço do sistema serem quadráticos em relação ao número de variáveis, eles são constantes no número de exemplos. Da mesma forma, os requisitos de tempo do sistema são também quadráticos no número de variáveis, mas são também constantes no número de exemplos. Uma característica importante desse algoritmo é que, a cada vez que ocorre uma divisão em uma folha com d variáveis, o número de distâncias que é necessário calcular diminui em no mínimo $d-1$, diminuindo assim o tempo gasto, por exemplo. Os resultados experimentais, utilizando conjuntos de dados artificiais e reais, mostraram eficácia competitiva quando comparados a uma análise estática de agrupamentos, evoluindo e adaptando-se na presença de alterações de conceito.

15.2.3 Redes Neurais

RNAs são modelos computacionais que podem aproximar qualquer função contínua (Craven e Shavlik, 1997) com erro arbitrariamente pequeno. Um dos problemas das RNAs é a lentidão do processo de aprendizado. Outros problemas conhecidos são a tendência a superajuste, a variância do erro esperado e a convergência para mínimos locais. À primeira vista, esses aspectos parecem impedir a aplicação de redes neurais ao cenário de fluxo de dados.

O processo habitual para treinar uma rede neural consiste no processamento dos exemplos de treinamento por épocas. Qual a necessidade de processar o mesmo conjunto de exemplos várias vezes? A única motivação é a falta de exemplos. Uma das principais características de fluxos contínuos de dados é que os dados são abundantes. Nesse contexto, uma rede neural pode ser treinada processando cada exemplo uma única vez (Chen e Pung, 2008). Sempre que um exemplo de treinamento está disponível, seu vetor dos atributos é propagado pela rede, e o erro é retropropagado através da rede apenas uma vez.

Essa metodologia de treinamento é uma grande vantagem, permitindo a aplicação de redes neurais em processos que geram um número infinito de exemplos e em alta velocidade. Outra vantagem é a adaptação gradual em fluxos de dados dinâmicos, em que a função alvo evolui ao longo do tempo. Craven e Shavlik (1997) referem que o viés indutivo de redes neurais é o mais adequado para tarefas de previsão sequenciais e temporais.

15.3 Detecção de Mudança

Os fluxos contínuos de dados evoluem ao longo do tempo. Assumir que os exemplos são gerados aleatoriamente de acordo com uma distribuição estacionária é irrealista (Basseville e Nikiforov, 1987). Em sistemas complexos, são esperadas mudanças (suaves ou abruptas) na distribuição dos exemplos e evoluções dos conceitos a aprender. Mineração de dados nesses contextos requer algoritmos adaptativos: algoritmos incrementais que detectam e reagem a mudanças do conceito a aprender.

Mudança de conceitos (do inglês *concept drift*) (Klinkenberg, 2004) significa que o processo que gera dados evolui ao longo do tempo. Por exemplo, os interesses e objetivos

de um usuário de um programa de busca na internet evoluem ao longo do tempo. A evidência de mudanças no conceito a aprender reflete-se de alguma forma nos exemplos de treinamento. Observações mais antigas, que refletem o comportamento no passado, tornam-se irrelevantes para o estado atual dos fenômenos em observação. A natureza da mudança é muito variada. As alterações podem ocorrer quer devido a mudanças em variáveis que não são mais observadas, quer em alteração nas propriedades características das variáveis observadas. Muitos algoritmos de aprendizado utilizam métodos cegos, que adaptam o modelo de decisão em intervalos regulares, sem considerar se houve de fato alguma mudança. Muito mais interessantes são os mecanismos que detectam mudanças explicitamente. Uma das vantagens é que podem proporcionar informações significativas sobre o processo em análise, indicando pontos de mudança, ou janelas temporais onde a mudança ocorre, assim como quantificar o grau de mudança. Podemos agrupar essas abordagens em dois grandes grupos:

- Monitoram a evolução de indicadores de desempenho, por exemplo, usando técnicas utilizadas em Controle Estatístico de Processos (Gama et al., 2004).
- Monitoram a distância entre as distribuições em duas janelas temporais (Kifer et al., 2004). Nesse caso, o método acompanha a evolução de uma função de distância entre duas distribuições: uma janela de referência e em uma janela sobre os pontos mais recentes.

As métricas e os critérios para a seleção de algoritmos de detecção de mudança normalmente têm em conta a taxa de erro, a rapidez de detecção, assim como a robustez a falsos alarmes e ruído.

Em aprendizado preditivo, uma das formas mais eficazes para detectar mudanças ou desvios no conceito a aprender consiste em monitorar o processo de aprendizado dos modelos preditivos. Por exemplo, monitorando a evolução de uma medida de desempenho. Vários algoritmos de detecção de mudança, que traçam a evolução da taxa de erro, foram apresentados na literatura: o *Page-Hinkley* teste (Hinkley, 1970), o SPC (do inglês *Statistical Process Control*) (Gama et al., 2004) ou o Adwin *Adaptive Window* (Bifet e Gavaldà, 2007). Esses algoritmos monitoram a evolução do erro de um classificador. Assumem que, em processos estacionários o erro do classificador deve manter-se constante ou diminuir. Se o processo é não estacionário, o erro aumenta. No entanto, um aumento do erro pode ser motivado por ruído nos dados. É esse o problema principal dos algoritmos de detecção de mudança: distinguir mudanças estruturais no processo que gera dados de perturbações temporárias nesse processo.

Nesta seção apresentamos o algoritmo SPC, apresentado por Gama et al. (2004). O algoritmo SPC monitora a evolução da taxa de erro de um classificador. Sinaliza alertas de degradação do processo de aprendizado, e indica mudança quando essa degradação corresponde (para um determinado nível de confiança) a uma alteração estrutural no processo que gera dados. No SPC, um alerta de mudança é sinalizado quando observamos um aumento do erro, mas não há ainda evidência para sinalizar uma mudança. O SPC mantém uma memória de curto prazo de exemplos rotulados recebidos entre um sinal de alerta e o sinal de desvio. Portanto, podemos explorar os exemplos armazenados na memória de curto prazo para inicializar o modelo de decisão sempre que um desvio está sinalizado.

O algoritmo SPC, cujo pseudocódigo é apresentado no Algoritmo 15.2, mantém dois registros: p_{min} e s_{min} , em que p representa uma probabilidade de erro e s , a variância

associada à estimativa de p . p é calculado como a frequência de erro e s , a variância do erro dada por $s = \sqrt{p \times (1 - p)/n}$, em que n representa o número de exemplos processados. Sempre que um novo exemplo i é processado, esses valores são atualizados se $p_i + s_i < p_{min} + s_{min}$. É utilizado um nível de aviso para definir o tamanho ideal da janela de contexto. A janela de contexto contém os exemplos de idade que estão no novo contexto e um número mínimo de exemplos no contexto antigo. Nos experimentos, o nível de alerta é atingido se $p_i + s_i \geq p_{min} + 2 \times s_{min}$, e o nível de mudança é alcançado se $p_i + s_i \geq p_{min} + 3 \times s_{min}$. Suponha uma sequência de exemplos em que o erro do modelo aumenta atingindo o nível de alerta no exemplo k_w , e o nível de mudança no exemplo k_d . Um novo modelo é induzido usando os exemplos a partir de k_w até k_d . É possível observar um aumento do erro de atingir o nível de alerta, seguido de um decréscimo. É assumido que tais situações correspondem a um falso alarme, sem alterar o contexto.

Algoritmo 15.2 O Algoritmo SPC para detecção de mudança

Entrada: Um modelo de decisão atual \hat{f}
 Uma sequencia de exemplos $\{(\mathbf{x}_j, y_j), j = 1, \dots, n\}$

- 1 Seja (\mathbf{x}_j, y_j) o exemplo atual
- 2 Computa a previsão do modelo: $\hat{y}_j \leftarrow \hat{f}(\mathbf{x}_j)$
- 3 Computa o erro: $erro_j$
- 4 Computa média dos erros: p_j e variância s_j
- 5 **se** $p_j + s_j < p_{min} + s_{min}$ **então**
 - 6 $p_{min} \leftarrow p_j$
 - 7 $s_{min} \leftarrow s_j$
- 8 **fim**
- 9 **se** $p_j + s_j < p_{min} + \beta \times s_{min}$ **então**
 - 10 $Aviso? \leftarrow Falso$
- 11 Atualiza o modelo de decisão usando o exemplo atual: \mathbf{x}_j, y_j
- 12 **fim**
- 13 **senão**
 - 14 **se** $p_j + s_j < p_{min} + \alpha \times s_{min}$ **então**
 - 15 **se** $! Aviso?$ **então**
 - 16 $buffer \leftarrow \{(\mathbf{x}_j, y_j)\}$
 - 17 $Aviso? \leftarrow Verdadeiro$
 - 18 **fim**
 - 19 **senão**
 - 20 $buffer \leftarrow buffer \cup \{(\mathbf{x}_j, y_j)\}$
 - 21 **fim**
 - 22 **fim**
 - 23 **senão**
 - 24 Reaprende um novo modelo de decisão com os exemplos no $buffer$
 - 25 $Aviso? \leftarrow Falso$
 - 26 Reinicializa p_{min} e s_{min}
 - 27 **fim**
 - 28 **fim**

15.4 Considerações Finais

Detecção de mudança em aprendizado preditivo é um assunto bem estudado (Klinkenberg, 2004). Em outros cenários, como agrupamento de dados, poucos são os algoritmos capazes de lidar com distribuições não estacionárias. A questão principal é como incorporar mecanismos de detecção ao algoritmo de aprendizado, nomeadamente em diferentes paradigmas.

Outro aspecto relevante de qualquer algoritmo de aprendizado é a definição de métricas para a avaliação de hipóteses. A maioria dos métodos de avaliação (e métricas associadas) foi concebida para o caso de modelos estáticos e dados estacionários, ou seja, independentes e identicamente distribuídas. No caso de fluxos contínuos de dados, estamos muito mais interessados em estimar a evolução ao longo do tempo da capacidade de generalização do modelo. Resultados das estatísticas sequenciais (Wald, 1947) parecem ser muito mais apropriados.

Aprendizado em fluxos contínuos de dados é uma área de pesquisa crescente com desafios em termos de aplicações e oportunidades de pesquisa em áreas científicas como bases de dados, teoria de algoritmos, aprendizado de máquina e mineração de dados. Redes de sensores, dados científicos, monitoração de processos, análise de dados da web e análise de tráfego em redes de computadores são exemplos de aplicações relevantes e onde análise de fluxo de dados tem sido bem-sucedida. Aprendizado em tempo real, a capacidade de incorporar nova informação de forma contínua, a capacidade de esquecer, a capacidade de autoadaptação e autorreação são as principais características de qualquer sistema inteligente. São propriedades características dos algoritmos de aprendizado sobre fluxos contínuos de dados.

Capítulo 16

Meta-aprendizado

Um dos principais desafios enfrentados quando da utilização de algoritmos de AM em novos conjuntos de dados é a escolha do algoritmo (ou conjunto de algoritmos) mais apropriado. Como visto nos capítulos anteriores, diferentes algoritmos de AM podem ser utilizados em problemas reais. Embora exista um grande número de algoritmos de AM, faltam regras ou dicas que auxiliem na escolha do algoritmo mais apropriado para um dado problema. Essa escolha geralmente ocorre por tentativa e erro ou é influenciada pela disponibilidade do algoritmo na ferramenta computacional utilizada, por experiências passadas do usuário ou por sugestões de especialistas em AM. Essas abordagens, além de serem subjetivas, podem ter um custo computacional elevado.

Conforme mencionado em Wolpert (1996), não existe um único algoritmo de AM que seja sempre melhor que os demais, pois o desempenho de um algoritmo depende de como seu viés indutivo se adapta a propriedades presentes no conjunto de dados a ser utilizado. Pesquisas realizadas em AM, em uma área denominada meta-aprendizado, indicam que é possível, durante o processo de aprendizado, aprender quais são os algoritmos mais apropriados para um novo conjunto de dados. A análise de resultados experimentais obtidos em pesquisas realizadas nessa área sugerem que meta-aprendizado pode fornecer uma ferramenta poderosa de apoio à seleção de algoritmos não apenas de AM, mas de algoritmos baseados em qualquer princípio que possam ser utilizados para aproximação de funções.

Sem perda de generalidade, será assumido neste texto que meta-aprendizado será utilizado para a seleção de algoritmos de AM a serem utilizados em problemas de classificação. É fácil adaptar os conceitos apresentados para outras classes de problemas, como regressão, previsão de séries temporais, agrupamento de dados e otimização combinatorial, e outros algoritmos de aproximação de função, como técnicas das áreas de estatística e de otimização.

De acordo com Brazdil et al. (2009), meta-aprendizado é o estudo dos principais métodos que exploram metaconhecimento para obter modelos e soluções eficientes por meio da adaptação de processos de aprendizado de máquina e mineração de dados. Os mesmos autores apresentam uma definição mais específica na qual definem meta-aprendizado como o uso da abordagem de AM para gerar metaconhecimento mapeando as características de problemas (meta-atributos) ao desempenho relativo de algoritmos. Para isso, investiga como selecionar os algoritmos mais promissores para um dado conjunto de dados (Vilalta e Drissi, 2002; Brazdil et al., 2009; Smith-Miles, 2008; Prudêncio e Ludermir, 2004; de Souza, 2010).

Meta-aprendizado difere do aprendizado convencional no nível em que ocorre a adaptação. No aprendizado convencional, que ocorre em um nível inicial, denominado nível

de base, o processo de aprendizado ocorre em um conjunto de dados por vez. No meta-aprendizado, que ocorre em um nível mais avançado, chamado metanível, o aprendizado se dá por meio do acúmulo de experiências obtidas observando o desempenho de diversos algoritmos de AM quando aplicados a vários conjuntos de dados (Brazdil et al., 2009).

Dois grandes objetivos do meta-aprendizado são fornecer suporte a usuários que não possuem experiência em seleção de algoritmos de AM e estudar como utilizar o conhecimento obtido da experiência acumulada pela aplicação de diversos algoritmos de AM a vários conjuntos de dados para a geração de modelos com maior acurácia preditiva.

Um processo de meta-aprendizado começa com a criação de um grupo de conjuntos de dados e a seleção de um grupo de algoritmos de AM. Para cada conjunto são extraídas características que descrevem suas principais propriedades e anotado o desempenho dos algoritmos de AM selecionados quando aplicados ao conjunto, formando um metadado. O conjunto de metadados gerados para todos os conjuntos de dados é, posteriormente, utilizado para a construção de um sistema de recomendação, capaz de selecionar os algoritmos mais adequados para um novo problema.

Uma abordagem simples para o sistema de recomendação é utilizar um algoritmo de aprendizado baseado em instância, como o algoritmo *k*-NN, para, dadas as características do conjunto de dados associados a um novo problema, selecionar os algoritmos com melhor desempenho nos *k* conjuntos de dados mais semelhantes. A similaridade entre conjuntos de dados é definido pelas características extraídas e pode ser calculada pela distância euclidiana, por exemplo. Em uma abordagem mais sofisticada, o processo de meta-aprendizado utiliza um algoritmo de AM que, após treinado com os metadados, induz um modelo capaz de associar as características de um conjunto de dados ao desempenho obtido pelos diferentes algoritmos de AM. Esse modelo é então utilizado em um sistema de recomendação que, dado um novo conjunto de dados, recomenda os algoritmos de AM mais promissores.

O resultado produzido por um sistema de recomendação pode ser um algoritmo ou um conjunto de algoritmos. Quando um conjunto de algoritmos é selecionado, um escore pode ser associado a cada um dos algoritmos. Com esse escore, os algoritmos podem ser ordenados em um *ranking* conforme sua adequação (Soares, 2004).

Segundo Kalousis (2002), um sistema de recomendação pode ser analisado de acordo com os seguintes aspectos:

- Propriedades utilizadas para caracterização do conjunto de dados;
- Medidas de avaliação dos algoritmos;
- Formas de apresentação de sugestões;
- Métodos para construção das sugestões.

Nas seções seguintes será apresentada uma breve explicação de cada um desses aspectos.

16.1 Caracterização de Conjuntos de Dados

A caracterização de conjuntos de dados procura extrair características presentes nos dados que possam influenciar o desempenho de algoritmos de AM. Essas características podem ser utilizadas durante o processo de meta-aprendizado para, por exemplo,

recomendar os algoritmos de AM mais promissores para conjuntos de dados com tais características. Para isso, é necessário conhecer de antemão como diferentes algoritmos se comportam para conjuntos de dados com diferentes características. Por exemplo, é sabido que, em geral, o NB se sai bem quando os atributos de entrada são independentes, que SVMs lidam bem quando o número de atributos de entrada é elevado e que k -NN não se comportam bem com atributos irrelevantes. Essas relações entre características de um conjunto de dados e desempenho de algoritmos de AM são exploradas por esse critério (de Souza, 2010).

Soares (2004) sugere que as características ou medidas extraídas de conjuntos de dados devem conter informação relevante que permita estimar o desempenho relativo entre algoritmos de AM. Além disso, a extração dessas medidas deve ter um baixo custo computacional. De acordo com Vilalta et al. (2005), os estudos realizados em caracterização de dados são geralmente divididos em três abordagens:

- Caracterização direta;
- Caracterização por propriedades de modelos;
- Caracterização baseada em *landmarking*.

Os principais aspectos de cada uma dessas abordagens serão explicados a seguir.

16.1.1 Caracterização Direta

A caracterização direta é a forma mais simples de extrair propriedades de um conjunto de dados. Em um dos primeiros trabalhos a utilizar caracterização direta de dados, o projeto STATLOG (Michie et al., 1994), as medidas extraídas de cada conjunto de dados, denominadas meta-atributos, foram divididas em três classes:

- **Medidas simples:** incluem descrições gerais do conjunto de dados, como, por exemplo, número de classes, número de atributos, número de atributos binários e número de objetos.
- **Medidas baseadas em estatística:** são medidas que descrevem estatisticamente os dados, que podem ser valores de obliquidade, valores de curtose, correlação entre atributos por classe, desvio-padrão dos atributos.
- **Medidas baseadas em teoria da informação:** procura quantificar a informação presente nos dados, utilizando medidas como entropia e informação mútua.

Cada conjunto de dados pode ser representado por um vetor com valores para seus meta-atributos. Isso foi feito em diversos estudos. O projeto STATLOG extraiu valores para esses meta-atributos de 21 conjuntos de dados. Para cada conjunto de dados, foram realizados experimentos com 23 algoritmos de classificação diferentes, avaliando a acurácia preditiva de cada um deles. Um dos objetivos do projeto era identificar a classe de problemas mais adequada para cada algoritmo e as medidas de desempenho capazes de caracterizar o sucesso de um algoritmo em um dado problema. A principal medida de desempenho utilizada foi a taxa de erro de classificação. Em Sovat (2002), foi proposto um sistema que utilizava raciocínio baseado em casos para, dados os valores para um

conjunto de meta-atributos, que incluíam as medidas propostas no projeto STATLOG, selecionar uma rede neural e os valores de seus parâmetros.

Novas medidas para caracterização de dados foram propostas no projeto METAL (Brazdil et al., 2009), que disponibilizou um sistema de recomendação na internet que retornava sugestões de algoritmos para conjuntos de dados submetidos por interessados via internet. O principal objetivo do projeto METAL foi o de melhorar o uso de ferramentas de mineração de dados e gerar ganho significativo no tempo dedicado à experimentação (Vilalta et al., 2005). Em de Souza (2010), é investigado um novo conjunto de medidas, baseadas em índices de validação de agrupamentos de dados. Outras sugestões para medidas de caracterização direta podem ser encontradas em Kalousis (2002) e Soares (2004).

16.1.2 Caracterização por Propriedades de Modelos

Na caracterização via modelos, os meta-atributos gerados para um conjunto de dados são propriedades de um ou mais modelos induzidos utilizando esse conjunto (Bensusan, 1998; Bensusan et al., 2000; Peng et al., 2002). Vilalta et al. (2005) apresentam as seguintes vantagens para essa caracterização:

- O conjunto de dados é sumarizado por uma estrutura de dados que embute a complexidade e o desempenho da hipótese induzida, e não apenas a distribuição dos dados.
- A representação obtida pode ser utilizada para explicar o desempenho do algoritmo de AM.

Quando modelos são utilizados para caracterizar conjuntos de dados, ocorre uma mudança no espaço de busca do processo de meta-aprendizado, que passa do espaço de objetos para o espaço de modelos (Bensusan et al., 2000). Como o novo espaço permite uma busca eficiente no espaço de hipóteses, espera-se uma sumarização maior e mais eficiente do conjunto de dados originais, levando a melhores meta-atributos.

Vilalta et al. (2005) citam como exemplo a indução de uma árvore de decisão para representar um conjunto de dados. Como propriedades do modelo induzido que podem ser utilizadas para representar o conjunto de dados, podem ser citadas número de nós folha, formato da árvore, profundidade máxima da árvore e grau de balanceamento da árvore. De acordo com Bensusan et al. (2000), evidências empíricas mostram que propriedades dos conjuntos de dados podem estar relacionadas a estruturas de árvores de decisão não podadas.

16.1.3 *Landmarking*

Na caracterização baseada em *landmarking*, são utilizadas informações sobre o desempenho de um conjunto de algoritmos de classificação rápidos e simples, denominados *landmarkers*, para os conjuntos de dados do repositório (Pfahringer et al., 2000; Bensusan e Giraud-Carrier, 2000a).

Nessa abordagem, os conjuntos de dados são caracterizados pelos desempenhos de diferentes algoritmos aplicados a eles. É esperado que conjuntos de dados gerem meta-atributos com valores semelhantes quando o desempenho dos classificadores nesses con-

juntos for similar. Diferentes medidas de desempenho podem ser utilizados como meta-atributos, como, por exemplo, precisão, revocação e área sob a curva ROC. Outras abordagens mais sofisticadas também foram investigadas (Brazdil et al., 2009).

Para essa abordagem, é sugerido o uso de algoritmos diferentes, que sejam adequados a diferentes conformações de dados. É também aconselhável que os conjuntos de dados utilizados cubram os domínios de problemas em que os algoritmos se destacam.

Soares et al. (2001) investigaram uma variante denominada *Landmarking* baseado em amostras, que combina os conceitos de amostragem de dados e *Landmarking*. Para isso, são utilizadas amostras de conjuntos de dados para estimar o desempenho de algoritmos em conjuntos completos. As estimativas de desempenho são utilizadas para caracterizar os conjuntos de dados.

16.2 Medidas de Avaliação dos Algoritmos

A seleção do algoritmo de AM mais apropriado para um conjunto de dados pressupõe a utilização de medidas que avaliem o desempenho de algoritmos de AM. A princípio, qualquer medida de desempenho empregada para avaliar os algoritmos candidatos pode ser utilizada. Para problemas de classificação, podem ser utilizadas, por exemplo, medidas de acurácia preditiva, como taxa de classificações incorretas, revocação, precisão, medida-F e área sob a curva ROC 9. Outras medidas além da acurácia preditiva podem ser utilizadas, como, por exemplo, custo computacional para as etapas de treinamento e de teste, quantidade de memória necessária, complexidade do modelo induzido e facilidade de interpretação do modelo.

Nada impede que duas ou mais medidas que avaliem diferentes aspectos possam ser utilizadas em conjunto, atribuindo um peso a cada uma delas, definindo uma ordem de importância entre elas ou empregando técnicas de otimização multiobjetivo. Isso pode ocorrer, por exemplo, no desenvolvimento de uma ferramenta computacional baseada em AM para dispositivos com pouca capacidade de memória que necessitem de uma boa acurácia preditiva.

16.3 Formas de Apresentação de Sugestões

A forma de seleção define quantos algoritmos são selecionados e como esses algoritmos são apresentados ao usuário. De acordo com Kalousis (2002), as sugestões podem ser apresentadas de três formas:

- Sugestão do melhor algoritmo;
- Sugestão de um grupo com os melhores algoritmos;
- Sugestão de um *ranking* dos melhores algoritmos.

Soares (2004) investigou o uso de *ranking* para a sugestão de algoritmos de AM.

Na primeira forma é apresentado o algoritmo que induziu o melhor modelo para o novo conjunto de dados, de acordo com alguma medida de avaliação. Essa abordagem é utilizada em Koepf et al. (2000) e Bensusan e Giraud-Carrier (2000b). A ausência de alternativas para o algoritmo recomendado pode ser um problema quando ele não puder

ser utilizado. Isso ocorre, por exemplo, quando o algoritmo não estiver implementado na ferramenta a ser utilizada.

Esse problema é resolvido pela segunda forma, que recomenda o conjunto dos melhores algoritmos. Fazem parte desse conjunto o melhor algoritmo e os algoritmos com desempenhos estatisticamente semelhantes ao melhor. Essa forma foi utilizada no projeto STATLOG.

A terceira forma vai um passo além, ordenando os melhores algoritmos de acordo com o desempenho obtido, gerando assim um *ranking* de algoritmos. Tanto uma única medida como uma combinação de medidas podem ser utilizadas para a formação do *ranking*.

16.4 Recomendações a partir da Caracterização

Uma das principais tarefas do meta-aprendizado é relacionar as características das bases de dados ao desempenho dos algoritmos de AM, de forma a permitir a recomendação dos algoritmos mais promissores para um novo conjunto de dados. A construção desse relacionamento pode ser vista como um problema de AM, situado em um metanível. As características das bases de dados são utilizadas para gerar metaexemplos. Os metaexemplos são caracterizados por meta-atributos e possuem como classe associada alguma informação acerca do desempenho dos algoritmos de AM disponíveis. O conjunto de metaexemplos constitui o metaconjunto de dados, que pode ser utilizado por algoritmo de AM para a indução de um modelo a ser utilizado por um sistema de recomendação.

O modelo induzido para sugerir algoritmos de AM pode ser tanto um regressor, denominado metarregressor, que associa valores reais a um dos algoritmos de AM avaliado, como um classificador, denominado metaclassificador, que possui como classes os algoritmos avaliados. O resultado gerado por um metarregressor para um novo conjunto de dados pode ser a estimativa do desempenho do algoritmo de AM associado ao metarregressor para esse novo conjunto. Os resultados gerados por um metarregressor para cada um dos algoritmos avaliados pode ser utilizado para sugerir um *ranking* dos melhores algoritmos. Quando um metaclassificador é utilizado, o resultado pode ser a previsão de uma ou mais classes. Quando mais de uma classe puder ser prevista, tem-se um problema de classificação multirrótulo, coberto no Capítulo 18.

16.5 Estudo de Casos

No projeto STATLOG, foi observado o desempenho de algoritmos de classificação para uma grande variedade de problemas. Mais recentemente, foram realizados trabalhos em que meta-aprendizado foi utilizado para a seleção de algoritmos em domínios bem definidos.

Em Prudêncio e Ludermir (2004) e Prudêncio e Ludermir (2006), meta-aprendizado é utilizado para a seleção de modelos para análise de séries temporais. Foram realizados experimentos com séries apresentando diferentes características, obtidas tanto de um repositório internacional como de uma competição internacional. Foram propostas e utilizadas medidas de caracterização direta para séries temporais. Diferentes estratégias foram investigadas para a seleção de modelos. A mais simples selecionava entre dois modelos. Uma segunda abordagem criava um *ranking* de três possíveis modelos. Para isso, esses modelos foram divididos em três grupos de dois modelos cada. Uma rede MLP foi

treinada para classificar uma série em um dos modelos de cada grupo. As classificações das redes são utilizadas para criar um *ranking* de modelos (Prudêncio e Ludermir, 2004). Uma terceira abordagem também utiliza uma rede MLP, mas agora para regressão, definindo um metarregressor (Prudêncio e Ludermir, 2006). O objetivo é encontrar pesos para ponderar a participação de dois modelos de previsão em uma combinação de modelos. Resultados experimentais mostraram uma redução significativa de erros de previsão de séries temporais.

O domínio de classificação de dados de expressão gênica é abordado em de Souza et al. (2010) e de Souza (2010). Nesses trabalhos foi avaliado o desempenho de 7 algoritmos de classificação para 49 conjuntos de dados de análise de expressão gênica. Foram escolhidos algoritmos que fossem facilmente encontrados em ferramentas computacionais públicas, que possuíssem um baixo custo computacional e fossem reconhecidos por apresentar uma boa acurácia preditiva. Foram também propostas novas medidas para caracterização direta de dados, baseadas em índices de validação de agrupamentos de dados. Como dados de expressão gênica apresentam milhares de atributos, foram investigadas alternativas para reduzir esse número. A seleção de algoritmos foi realizada de diferentes formas. Foi proposto o uso de um algoritmo *k*-NN com pesos e diferentes funções *kernel*, que privilegiava metaexemplos mais próximos, o uso de metarregressores baseados em SVMs e novos métodos de *ranking* de algoritmos que empregam árvores de decisão e *bagging*. Outro estudo do uso de meta-aprendizado para dados de expressão gênica, mas para a seleção de algoritmos de agrupamento de dados, pode ser encontrado em de Souto et al. (2008).

Meta-aprendizado também tem sido utilizado para a seleção de técnicas de otimização. Em Kanda et al. (2010), meta-aprendizado é utilizado para selecionar os algoritmos de otimização mais promissores para novas instâncias do problema do caixeiro-viajante (Gutin et al., 2002). O objetivo de uma técnica de otimização quando aplicada a esse problema é encontrar a melhor sequência de viagens entre as cidades de forma a que cada cidade seja visitada apenas uma vez. O problema do caixeiro-viajante pode ser representado por um grafo, em que cada cidade é um nó do grafo e a distância entre duas cidades é associada ao tamanho da aresta que conecta as duas cidades. Para isso, são definidos novos meta-atributos baseados em propriedades de grafos. Algoritmos de classificação são utilizados para induzir modelos capazes de, dada uma nova instância do problema, prever qual a técnica de otimização mais promissora. Nesse trabalho, é permitido que diferentes técnicas de otimização fiquem empatadas quando da seleção da mais promissora. Para lidar com essa situação, são investigadas diferentes estratégias para classificação multirótulo, apresentadas no Capítulo 18.

16.6 Considerações Finais

Neste capítulo foram apresentados os principais conceitos de uma área de pesquisa recente de AM, a área de meta-aprendizado. Trabalhos nessa área ajudam a conhecer os potenciais e as limitações dos algoritmos de AM para a solução de problemas reais. Com isso, permite o desenvolvimento de sistemas de recomendações que, de acordo com características presentes em um conjunto de dados, sugerem os algoritmos mais promissores para serem aplicados a esses dados.

Pesquisas na área de meta-aprendizado têm ganhado grande impulso nos últimos anos,

com propostas em vários temas, como definição de meta-atributos para domínios de aplicação específicos, novos métodos para ordenação de algoritmos e recomendação de valores de parâmetros para algoritmos de AM.

Outro benefício da utilização de meta-aprendizado é possibilitar a construção de mecanismos para transferência de conhecimento entre diferentes domínios ou tarefas (Brazdil et al., 2009). Assim, o conhecimento adquirido quando algoritmos de AM são utilizados em problemas em um domínio pode ser indiretamente utilizado no aprendizado em outros domínios, permitindo a transferência de conhecimento entre domínios.

Alguns conjuntos de dados estão em constante expansão, com novos objetos sendo continuamente gerados. Isso ocorre em particular quando os dados são gerados em fluxos contínuos, conforme visto no Capítulo 15. Nesses casos, os algoritmos mais adequados em um dado momento podem não ser os mais adequados em um momento posterior. Como resultado, a sugestão dos algoritmos indicados precisa ser atualizada quando houver mudanças no perfil dos dados.

Capítulo 17

Decomposição de Problemas Multiclasse

Diversas técnicas de AM foram originalmente formuladas para a solução de problemas de classificação contendo apenas duas classes, também denominados binários. Entre elas podem-se citar as SVMs e as RNAs Perceptron. Muitos problemas de classificação, contudo, são multiclasse, e apresentam mais de duas classes. A generalização de técnicas de classificação binária para problemas multiclasse pode ser realizada basicamente por meio de duas estratégias. A primeira consiste na combinação de preditores gerados em subproblemas binários, enquanto na segunda realizam-se adaptações nos algoritmos originais das técnicas consideradas. A extensão direta de um algoritmo binário a uma versão multiclasse nem sempre é possível ou fácil de realizar. Para as SVMs, em particular, Hsu e Lin (2002) e Rifkin e Klautau (2004) observaram que a reformulação dessa técnica em versões multiclasse leva a algoritmos computacionalmente custosos. Dessa forma, é comum recorrer-se à alternativa de decompor o problema multiclasse em múltiplos subproblemas binários, uma estratégia denominada decomposicional. As saídas dos preditores binários gerados na solução de cada um desses subproblemas são então combinadas na obtenção da classificação multiclasse final.

Segundo Moreira e Mayoraz (1997), existe uma série de motivações para empregar estratégias decompcionais na solução de problemas multiclasse. Além de haver técnicas cuja formulação é originalmente binária, alguns algoritmos não são adequados a problemas com um número elevado de classes ou apresentam dificuldades em lidar com grandes volumes de dados de treinamento. Há ainda algoritmos capazes de processar problemas com múltiplas classes que contêm procedimentos internos restritos a problemas de duas classes (por exemplo: a regra *towing* e a divisão do subconjunto de atributos nominais no CART (Breiman et al., 1984), a divisão do critério de seleção no QUEST (Loh e Shih, 1997)).

Mesmo que o algoritmo seja capaz de trabalhar com problemas de múltiplas classes de grande escala, o uso de um procedimento decomposicional pode reduzir a complexidade computacional envolvida na solução do problema total, por meio da divisão deste em subtarefas mais simples (Furnkranz, 2002). Knerr et al. (1992), por exemplo, observaram que as classes em um problema de reconhecimento de dígitos eram linearmente separáveis quando consideradas em pares.

O emprego de técnicas decompcionais envolve dois passos: uma fase de decomposição, que ocorre antes do aprendizado, e uma fase de reconstrução, que ocorre depois da predição. Dado um problema de decisão $\mathbf{D} = \{(\mathbf{x}_j, y_j), j = 1, \dots, n\}$, em que $y \in \{1, \dots, k\}$ e $k > 2$, a fase de decomposição consiste em obter múltiplos subproblemas binários na

forma de: $B_i = \{(\mathbf{x}_j, y'_j), j = 1, \dots, n\}$, em que $y'_j \in \{-1, +1\}$. Um algoritmo de aprendizado constrói um modelo de decisão para cada problema B_i . Depois, os modelos de decisão são usados para classificar os exemplos de teste. A reconstrução refere-se à forma como as saídas dos classificadores binários são combinadas na determinação da classe de um exemplo.

Na Seção 17.1 são apresentadas algumas das principais estratégias para a decomposição de problemas multiclasse descritos na literatura. Na Seção 17.2, é abordada a obtenção das previsões multiclasse em uma etapa de reconstrução. Na Seção 17.3 são apresentadas as considerações finais do capítulo.

17.1 Fase de Decomposição

Várias alternativas podem ser empregadas na decomposição de problemas multiclasse em um conjunto de problemas de classificação binária. Genericamente, essas decomposições podem ser descritas por uma abordagem proposta por Allwein et al. (2000), em que elas são representadas por uma matriz de códigos \mathbf{M} . As linhas dessa matriz contêm códigos que são atribuídos a cada classe. As colunas de \mathbf{M} definem partições binárias das k classes e correspondem aos rótulos que essas classes assumem na geração dos classificadores binários. Tem-se então uma matriz de dimensão $k \times l$, em que k é o número de classes do problema e l representa o número de classificadores binários utilizados na solução multiclasse. Na Figura 17.1 é apresentado um exemplo de matriz de códigos na qual o número de classes k é igual a quatro e o número de classificadores binários l também é quatro. Abaixo dessa matriz de códigos são indicadas as partições binárias das classes realizadas por cada classificador binário representado em suas colunas.

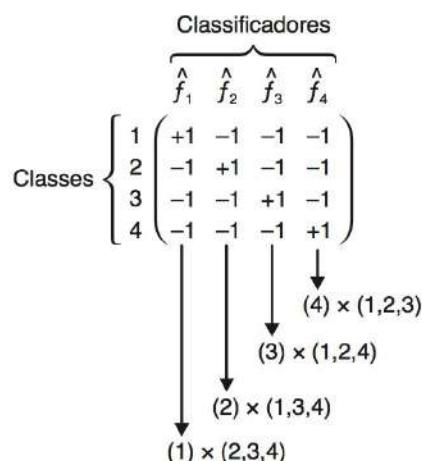


Figura 17.1 Exemplo de matriz de códigos para um problema com quatro classes.

Cada elemento da matriz \mathbf{M} assume valores em $\{-1, 0, +1\}$. Um elemento m_{ij} com valor $+1$ indica que a classe correspondente à linha i assume rótulo positivo na indução do classificador \hat{f}_j . O valor -1 designa um rótulo negativo, e um valor 0 indica que os dados da classe i não participam do processo de indução do classificador \hat{f}_j . Para cada coluna, tem-se um conjunto de treinamento diferente, e classificadores binários são então treinados de forma a aprender os rótulos representados nas colunas de \mathbf{M} .

A decomposição mais compacta de um problema com k classes pode ser realizada com

o uso de $l = \lceil \log_2(k) \rceil$ classificadores binários (Mayoraz e Moreira, 1996). Um exemplo de matriz compacta para um problema com quatro classes é apresentado na Figura 17.2(a).

$$\begin{array}{ll} \left(\begin{array}{cc} +1 & +1 \\ +1 & -1 \\ -1 & +1 \\ -1 & -1 \end{array} \right) & \left(\begin{array}{cccccc} +1 & +1 & +1 & +1 & +1 & +1 \\ -1 & -1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & +1 & -1 & -1 \\ -1 & +1 & -1 & +1 & -1 & +1 \end{array} \right) \\ \text{(a)} & \text{(b) ECOC} \\ \left(\begin{array}{cccc} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{array} \right) & \left(\begin{array}{ccccc} +1 & +1 & +1 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 \\ 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 \end{array} \right) \\ \text{(c) OAA} & \text{(d) OAO} \end{array}$$

Figura 17.2 Matrizes de diferentes decomposições para um problema com quatro classes.

O número total de diferentes preditores binários para um problema com k classes é $0,5 \times (3^k + 1) - 2^k$, considerando que $\hat{f} = -\hat{f}$, ou seja, que a inversão das classes positivas e negativas produz o mesmo classificador. Desses, $2^{k-1} - 1$ classificadores incluem todas as classes simultaneamente, ou seja, possuem somente rótulos $+1$ e -1 , sem o elemento 0. Para quatro classes, tem-se nesse caso a matriz representada na Figura 17.2(b).

Entre as estratégias decompositivas mais comuns encontradas na literatura estão a um-contra-todos (Rifkin e Klautau, 2004), a todos-contra-todos (Hastie e Tibshirani, 1998; Furnkranz, 2002, 2003; Moreira, 2000), a baseada em códigos de correção de erros (*Error Correcting Output Codes* - ECOC) (Dietterich e Bariki, 1995) e as decomposições hierárquicas, que são descritas a seguir.

17.1.1 Um-contra-todos

Na estratégia um-contra-todos (OAA, do inglês *one-against-all*), dado um problema com k classes, k classificadores binários $\hat{f}_i(\mathbf{x})$ são gerados. Cada um desses preditores é treinado de forma a distinguir uma classe i das demais. A representação dessa técnica é dada por uma matriz de dimensão $k \times k$, na qual os elementos da diagonal possuem o valor $+1$ e os demais, o valor -1 . Uma matriz do tipo OAA para um problema de quatro classes é apresentada na Figura 17.2(c).

A decomposição OAA pode apresentar desvantagens quando a proporção de exemplos de uma classe é muito pequena em relação à do conjunto formado pelos dados das outras classes. Esse tipo de desbalanceamento pode dificultar a indução de um preditor que apresente bom desempenho no reconhecimento da classe considerada.

17.1.2 Todos-contra-todos

Na decomposição todos-contra-todos, também denominada um-contra-um (OAO, do inglês *one-against-one*) e em pares (*pairwise*), dadas k classes, $\frac{k(k-1)}{2}$ classificadores binários são gerados. Cada um deles é responsável por diferenciar um par de classes (i, j) , em que $i \neq j$. A matriz de códigos nesse caso possui então dimensão $k \times \frac{k(k-1)}{2}$, e cada

coluna corresponde a um classificador binário para um par de classes. Em uma coluna representando o par (i, j) , o valor do elemento correspondente à linha i é $+1$, e o valor do membro correspondente a j é igual a -1 . Todos os outros elementos da coluna possuem o valor 0, indicando que os dados de outras classes não participam do processo de indução do classificador. Na Figura 17.2(d) tem-se uma matriz OAO para um problema com quatro classes.

Embora o número de classificadores gerados na decomposição OAO seja da ordem de k^2 , o treinamento de cada um deles envolve dados de apenas duas classes. Com isso, mesmo com um número elevado de classes, o tempo total despendido na geração dos preditores geralmente não é grande.

Um problema apontado na decomposição OAO é que a resposta de um preditor para um par de classes (i, j) na realidade não fornece informação nenhuma quando o exemplo não pertence às classes i ou j . Suponha um problema com 10 classes. Dos 45 problemas de decisão binária, somente 9 podem classificar corretamente o exemplo teste. Todos os outros 36 irão classificar o exemplo de forma errada. Portanto, nessa proposta, $(k-1)(k-2)/2$ irão classificar incorretamente qualquer exemplo, e somente $k-1$ podem prover a classificação correta.

17.1.3 Códigos de Correção de Erros de Saída

A transmissão da informação em um canal ruidoso pode envolver perda de informação. Com o desenvolvimento da tecnologia digital, é possível ao receptor da mensagem detectar e corrigir erros. A ideia base consiste em codificar a mensagem previamente à transmissão, em vez de transmiti-la em seu formato original. A codificação envolve a introdução de alguma redundância. A mensagem codificada é enviada através do canal ruidoso. O receptor decodifica a mensagem, e, devido à redundância de eventuais erros, estes podem ser detectados e corrigidos. Esse é o contexto em que os códigos de correção de erros aparecem. Shannon (1948) foi o primeiro a mostrar a possibilidade de usar esses códigos em comunicação, obtendo um bom equilíbrio entre redundância e capacidade de recuperar erros. Posteriormente, Hamming (1950) apresentou a matriz Hamming, usada para codificar 4 bits de informação usando 7 bits:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Os 7 bits usados para codificar uma mensagem de 4 bits são: $C(d_1, d_2, d_3, d_4) = (d_1 + d_2 + d_4, d_1 + d_3 + d_4, d_1, d_2 + d_3 + d_4, d_2, d_3, d_4)$. Por exemplo, a mensagem 1001 é codificada em: 0011001. Suponha que a mensagem seja recebida como: $\mathbf{m} = 0010001$. A matriz Hamming irá nos contar se houve um erro e onde o erro está. Se não há erro, $\mathbf{Hm} = 0$; senão, podemos determinar onde está o erro, computando:

$$\mathbf{Hm} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Rodando \mathbf{Hm} no sentido horário, obtemos 100, indicando o quarto bit como erro.

Dietterich e Bariki (1995) propuseram o uso de códigos de correção de erros para representar as k classes de um problema multiclasse em AM. Essa técnica possui a denominação decomposição por códigos de correção de erros (*Error Correcting Output Codes – ECOC*). As matrizes de códigos nesse caso assumem valores em $\{-1, +1\}$. Mesmo com a substituição dos valores 0 por -1 , podemos notar que a matriz de Hamming \mathbf{H} não pode ser diretamente usada no contexto de ECOC para classificação: a última coluna não definirá um problema de decisão, e várias colunas são complementares (como a primeira e a sexta colunas, por exemplo).

A possibilidade de correção de erros requer que os códigos das classes contidos em \mathbf{M} sejam bem separados segundo a distância de Hamming. Sendo d_l a distância mínima entre qualquer par de linhas de \mathbf{M} , então o classificador multiclasse final é capaz de corrigir ao menos $\lfloor \frac{d_l-1}{2} \rfloor$ bits incorretos em uma predição. Além disso, na construção de bons códigos de correção de erros, deve-se estimular também que os erros dos classificadores binários gerados sejam não correlacionados. Exige-se então a separação entre as colunas de \mathbf{M} , ou seja, a distância de Hamming entre cada par de colunas deve ser grande. Se no algoritmo de aprendizado a inversão das classes positivas e negativas produz o mesmo classificador (ou seja, $\hat{f} = -\hat{f}$), então deve-se também fazer com que a distância de Hamming entre cada coluna e o complemento das outras seja grande. Por fim, nenhuma coluna deve ser constante (composta somente por $+1$ ou -1), pois não se forma um problema de decisão.

Com base nessas observações, Dietterich e Bariki (1995) propuseram quatro técnicas para o desenvolvimento de matrizes de códigos com boa capacidade de correção de erros. A escolha de cada uma delas é determinada pelo número de classes do problema. Para $k \leq 7$, esses autores recomendam o uso de um código exaustivo, que consiste na combinação de todos $2^{k-1} - 1$ classificadores binários unicamente com rótulos $+1$ e -1 , conforme ilustrado na Figura 17.2(b) para um problema com quatro classes. A distância d_l em uma matriz gerada pelo método exaustivo é de 2^{k-2} . Se $8 \leq k \leq 11$, é aplicado um método que seleciona colunas do código exaustivo. Para $k > 11$, tem-se duas opções: um método baseado no algoritmo *hill-climbing* e a geração de códigos BCH (Boser e Ray-Chaudhuri, 1960), os quais provêm da teoria de códigos de correção de erros em comunicação. Há outros trabalhos mais recentes com estratégias alternativas para construir ECOCs, entre os quais Pimenta et al. (2007) e Tapia et al. (2010).

Allwein et al. (2000) apontam que, apesar de os códigos gerados pelo ECOC possuírem boa propriedade de correção de erros, vários dos subproblemas binários criados podem ser difíceis de aprender. Por esse motivo, as técnicas mais simples OAA e OAO têm apresentado resultados comparáveis ou superiores ao ECOC em várias aplicações (Allwein et al., 2000; Rifkin e Klautau, 2004).

17.1.4 Decomposições Hierárquicas

Uma maneira alternativa de solucionar um problema multiclasse com preditores binários pode ser conduzida com sua decomposição hierárquica. De forma geral, a introdução de uma hierarquia em uma aplicação multiclasse pode reduzir a complexidade de sua solução. A ideia é realizar inicialmente discriminações mais gerais, as quais são refinadas sucessivamente até a obtenção da classificação final.

Na Figura 17.3 são apresentados dois exemplos de decomposições hierárquicas utilizadas na solução de um problema com quatro classes. O classificador da Figura 17.3(a) possui uma estrutura de árvore direcionada binária, em que exatamente uma aresta chega

a cada nó e no máximo duas arestas partem deles. Na Figura 17.3(b) tem-se uma estrutura mais geral, de um grafo direcionado acíclico, em que mais de uma aresta pode apontar para um mesmo nó. Por definição, as árvores direcionadas binárias são um tipo de grafo direcionado acíclico. Porém, para facilitar as exposições realizadas nesta seção, essas estruturas serão tratadas como tipos distintos. Em ambas as estruturas, cada nó interno corresponde a um classificador binário que distingue dois subconjuntos de classes, enquanto os nós terminais, denominados folhas, representam as classes individuais.

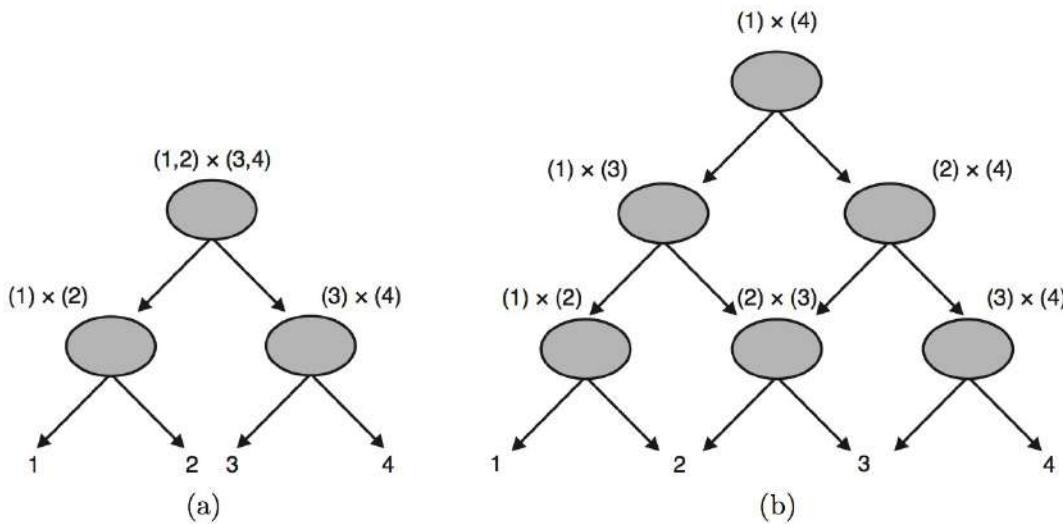


Figura 17.3 Decomposições hierárquicas para um problema com quatro classes.

Os preditores contidos nas decomposições hierárquicas também podem ser representados por uma matriz de códigos. O grafo da Figura 17.3(b), por exemplo, corresponde à mesma matriz da decomposição OAO (Figura 17.2(d)).

Para a obtenção das hierarquias, que equivale à etapa de decomposição do problema, várias estratégias podem ser empregadas. Algumas delas são discutidas a seguir. Inicialmente são apresentadas abordagens com estrutura de grafos direcionados acíclicos, seguidas de estruturas de árvores direcionadas binárias.

Grafos Direcionados Acíclicos

Platt et al. (2000) sugerem que os classificadores produzidos pela decomposição OAO sejam dispostos em um grafo de decisão direcionado acíclico (DDAG, do inglês *Decision Directed Acyclic Graph*). Logo, cada nó do grafo corresponde a um preditor binário para um par de classes. Essa estratégia será referenciada como DDAG neste documento. A Figura 17.3(b) ilustra um exemplo de DDAG para um problema com quatro classes.

Uma desvantagem do DDAG, apontada por Kijsirikul e Ussivakul (2002), é sua dependência em relação à sequência de classificadores binários contidos nos nós do grafo. Essa característica afeta a sua confiabilidade, uma vez que diferentes permutações de nós no grafo podem produzir resultados distintos. Outra deficiência do DDAG é que, dependendo da posição da classe correta no grafo, o número de avaliações com essa classe é desnecessariamente grande, resultando em um alto erro cumulativo.

Esses fatores motivaram Kijsirikul e Ussivakul (2002) a desenvolver uma nova estratégia hierárquica para combinar as saídas produzidas por classificadores obtidos pela decomposição OAO. A nova estrutura, denominada DAG Adaptável (ADAG, do inglês *Adaptive*

Directed Acyclic Graph), corresponde a um DDAG com estrutura reversa. Na Figura 17.4 é apresentado um exemplo de ADAG para um problema com oito classes. O ADAG tem $k - 1$ nós, cada qual correspondendo a um classificador binário para um par de classes. A primeira camada tem $\lceil k/2 \rceil$ nós, seguidos por $\lceil k/2^2 \rceil$ na segunda camada e assim por diante, até que uma camada com um único nó é atingida, a qual produz a saída final. Os nós da primeira camada diferenciam pares distintos de todas as classes. No caso de o número de classes ser ímpar, um dos nós contém apenas uma classe, a qual é diretamente passada para o segundo nível. Os níveis seguintes são adaptativos, sendo determinados de acordo com as previsões realizadas pelos classificadores dos níveis anteriores.

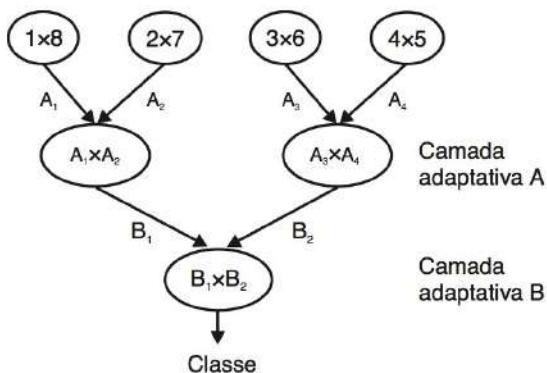


Figura 17.4 Exemplo de ADAG para problema com oito classes (Kjksirikul e Ussivakul, 2002).

O ADAG também foi proposto em um trabalho diferente como uma analogia a um torneio de tênis (Pontil e Verri, 1998). Primeiramente, todos os jogadores (classes) fazem partidas em pares. Com base no resultado dessas partidas, uma nova fase do torneio é iniciada, na qual os ganhadores das partidas anteriores são pareados de acordo com sua chave. Esse processo é repetido até que reste uma única classe.

O ADAG minimiza a ocorrência de erros cumulativos, especialmente em problemas com um número de classes elevado. Embora o ADAG também tenha demonstrado menor dependência que o DDAG em relação à ordem dos classificadores binários no grafo, ainda ocorrem diferenças de desempenho entre estruturas distintas. Logo, a definição dos classificadores que compõem o primeiro nível do ADAG afeta o seu resultado.

Alguns trabalhos investigaram heurísticas para determinar as estruturas de ADAGs e/ou DDAGs, por exemplo: Phetkaew et al. (2003), Takahashi e Abe (2003) e Lorena e Carvalho (2007). O número possível de diferentes DDAGs para um problema com k classes é $k!/2$ e de ADAGs é $k!/2^{\lfloor k/2 \rfloor}$. Na obtenção desses valores, classificadores para os pares de classes (i, j) são considerados equivalentes àqueles para os pares (j, i) , ou seja, considera-se $\hat{f} = -\hat{f}$.

Árvores Direcionadas Binárias

Diversos trabalhos empregam uma estrutura em forma de árvore direcionada binária na obtenção do classificador multiclasse hierárquico. Tais estruturas também são denominadas dicotomias embutidas. As árvores possuem $k - 1$ classificadores binários e, portanto, envolvem o treinamento de $k - 1$ preditores. Além disso, os nós de níveis inferiores envolvem cada vez menos classes e, portanto, menos dados de treinamento para os classificadores binários correspondentes.

Para um problema com $k \geq 3$ classes, existem $\prod_{i=3}^k 2i - 3$ estruturas de árvores distintas (Frank e Kramer, 2004). Duas possíveis árvores para um problema com quatro classes são ilustradas na Figura 17.5. Como no DDAG e no ADAG, a estrutura da árvore, ou seja, que classificadores são dispostos na árvore e onde eles se encontram, influencia o seu resultado. Os trabalhos nessa área diferenciam então no processo de obtenção das partições binárias das classes em cada nó da árvore e, consequentemente, na determinação de sua estrutura. Todos aplicam algum critério recursivamente sobre subconjuntos de classes, particionando-os em dois até que eles possuam apenas uma classe.

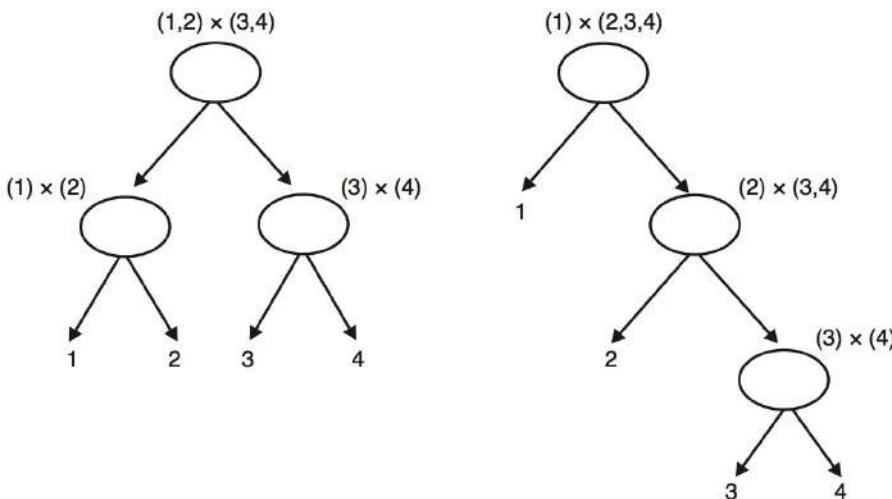


Figura 17.5 Duas árvores para um problema com quatro classes (Frank e Kramer, 2004).

Kumar et al. (2002), por exemplo, propõem que as partições das classes sejam feitas de maneira a maximizar a discriminação entre os grupos de classes formados. Esses autores usam para tal uma generalização do algoritmo de agrupamento k -médias que combina ideias de otimização por têmpora simulada (*simulated annealing*). Tibshirani e Hastie (2007) propuseram um algoritmo guloso que procura o classificador que maximize a separação entre os grupos de classes na hierarquia. Vural e Dy (2004) sugerem três métodos para a partição binária das classes na determinação de uma estrutura de árvore. O primeiro consiste em agrupar as classes com centros mais similares em cada nível da hierarquia com o algoritmo k -médias, com $k = 2$. No segundo, as classes são divididas de acordo com a distância média de seus dados à origem. No terceiro método, as classes são divididas em dois subconjuntos tal que a diferença entre o número de dados em cada um deles é mínima. Lorena e Carvalho (2008) propuseram um algoritmo para definir as partições binárias de classes em uma árvore que realiza um agrupamento hierárquico das classes de acordo com sua similaridade. Diferentes critérios podem ser utilizados para medir a similaridade das classes, tal como a distância entre seus centroides, a sua separabilidade (Lorena e Carvalho, 2010), entre outros.

17.2 Fase de Reconstrução

A fase de reconstrução define como as saídas dos classificadores binários serão combinadas na obtenção das previsões multiclasse. Nesse processo, um novo dado \mathbf{x} pode ser classificado avaliando-se as predições dos l classificadores, que geram um vetor $\hat{\mathbf{f}}(\mathbf{x})$ de

tamanho l na forma $\hat{\mathbf{f}}(\mathbf{x}) = (\hat{f}_1(\mathbf{x}), \dots, \hat{f}_l(\mathbf{x}))$. Esse vetor é então comparado às linhas de \mathbf{M} . O dado é atribuído à classe cuja linha de \mathbf{M} é mais próxima de $\hat{\mathbf{f}}(\mathbf{x})$ de acordo com alguma medida. Seja \mathbf{m}_q a q -ésima linha de \mathbf{M} , a qual apresenta o código referente à classe q . O processo de decodificação equivale a computar a Equação 17.1, em que $\hat{f}(\mathbf{x})$ representa o classificador multiclasse final e d denota uma função de decodificação.

$$\hat{f}(\mathbf{x}) = \arg \min_{1 \leq q \leq k} \left(d \left(\mathbf{m}_q, \hat{\mathbf{f}}(\mathbf{x}) \right) \right) \quad (17.1)$$

Caso mais de um código minimize a Equação 17.1, temos uma situação de empate. Escolhe-se então uma das classes envolvidas no empate aleatoriamente ou com o uso de alguma informação *a priori*.

Existem diversas funções de decodificação que podem ser utilizadas na integração dos classificadores binários na Equação 17.1 (Passerini et al., 2004; Allwein et al., 2000; Windeatt e Ghaderi, 2003). A mais simples é a de Hamming, que conta o número de ocorrências diferentes entre o vetor $\hat{\mathbf{f}}(\mathbf{x})$ e cada um dos códigos. Essa função pode ser visualizada na Equação 17.2, em que sgn corresponde à função sinal. Um rótulo 0 contribui com $\frac{1}{2}$ na computação da soma apresentada. A função descrita equivale à distância de Hamming caso \mathbf{M} possua unicamente os rótulos +1 e -1.

$$d_H \left(\mathbf{m}_q, \hat{\mathbf{f}}(\mathbf{x}) \right) = \sum_{i=1}^l \frac{1 - \text{sgn} \left(m_{qi} * \hat{f}_i(\mathbf{x}) \right)}{2} \quad (17.2)$$

Usualmente a classificação de um novo dado na estratégia OAA corresponde a escolher o classificador com maior saída. Isso equivale a usar alguma função de decodificação que considere a confiança dos classificadores nas previsões obtidas. No ECOC de Dietterich e Bariki (1995) originalmente propõe-se o uso da decodificação pela função de Hamming. Finalmente, para o OAO, a agregação mais usual é por uma votação por maioria. Dado um novo exemplo \mathbf{x} , cada classificador fornece um voto em sua classe preferida. O resultado é então dado pela classe que recebeu mais votos. Essa estratégia pode ser implementada pelo uso da distância de Hamming sobre a matriz de códigos da estratégia OAO.

No caso das estratégias hierárquicas, nem todos os classificadores binários precisam ser avaliados em uma previsão, ao contrário do apresentado anteriormente. Os classificadores consultados são determinados a partir das previsões realizadas para cada exemplo de teste, nível a nível nas hierarquias. Em geral, pode-se afirmar que as estratégias hierárquicas apresentam tempos menores de previsão do que as estratégias OAA, OAO com votação por maioria e ECOC. Isso ocorre porque, na classificação de um dado, apenas uma parte dos preditores binários é consultada.

Pode-se verificar que, na classificação de um novo exemplo com o DDAG e o ADAG, $k-1$ classificadores binários são avaliados. Logo, essas estruturas aceleram a fase de previsão da abordagem OAO tradicional usando votação por maioria. No ADAG, a classe correta é testada contra outras classes quando muito $\lceil \log_2 k \rceil$ vezes, enquanto no DDAG pode requerer até $k-1$ avaliações com essa classe. No caso das árvores direcionadas binárias, no melhor caso, dependendo da estrutura da árvore, é possível classificar o exemplo já no primeiro nó. No pior caso, os $k-1$ classificadores devem ser consultados. Logo, a fase de teste pode ser acelerada nessa estrutura perante as anteriormente discutidas.

A decodificação empregada na matriz de códigos de estruturas hierárquicas é conhecida

como eliminação (Klautau et al., 2003) e procede iterativamente. Inicialmente todos os classificadores binários (colunas da matriz) são considerados ativos. Em uma determinada iteração, um deles é consultado. As classes que “perdem” são eliminadas, assim como os classificadores binários relacionados a elas. Esse processo é repetido até que reste um único classificador binário, provendo a classificação final.

17.3 Considerações Finais

Embora algumas técnicas de aprendizado, como as SVMs, sejam originalmente formuladas para a solução de problemas de classificação binários, seu uso pode ser estendido a aplicações multiclasse. Em geral dois tipos de estratégias podem ser empregados nesse processo: decomposicionais e diretas. As decomposicionais dividem o problema total em múltiplos subproblemas binários, cujas saídas são combinadas na obtenção da previsão multiclasse. Nas estratégias diretas, reformula-se o algoritmo original da técnica de aprendizado em uma versão multiclasse. Entretanto, esse procedimento nem sempre é simples de ser realizado. Este capítulo revisou as principais estratégias decomposicionais da literatura. É importante ressaltar que o estudo das estratégias decomposicionais se aplica a qualquer técnica de aprendizado, bastando para isso utilizá-la na geração de classificadores binários.

Além das abordagens mencionadas neste capítulo, há também trabalhos que combinam as saídas dos preditores binários por meio do uso de outro classificador. Esse classificador, que pode ser produzido por outra técnica de aprendizado distinta da utilizada na obtenção dos preditores binários, é treinado de forma a ponderar as saídas dos classificadores binários na previsão multiclasse. Entre os trabalhos que fizeram uso desse tipo de estratégia, podem-se mencionar: Mayoraz e Alpaydim (1998) e Savicky e Fürnkranz (2003).

É possível também obter probabilidades de classificação a partir das previsões de vários classificadores binários. Entre os trabalhos nesse sentido, podem-se mencionar Hastie e Tibshirani (1998), Zadrozny (2001), Passerini et al. (2004) e Wu et al. (2004).

Capítulo 18

Classificação Multirrótulo

Em geral, classificadores induzidos por algoritmos de AM associam uma única classe ou rótulo a cada exemplo. Esses classificadores serão denominados neste capítulo classificadores de um único rótulo (ou simples-rótulo). Nesses problemas, um classificador é treinado em um conjunto de exemplos, em que cada exemplo \mathbf{x}_i é associado a uma única classe y_i de um conjunto R de k classes disjuntas.

Entretanto, existe um grupo de problemas reais de classificação, conhecidos como problemas de classificação multirrótulo, em que cada exemplo pode pertencer simultaneamente a mais de uma classe. Um classificador multirrótulo pode ser formalmente definido como uma função $H : X \rightarrow 2^k$ que mapeia cada objeto \mathbf{x}_i em um conjunto de classes ou rótulos \mathbf{y}_i , em que y_i é um vetor binário com k elementos. Cada elemento y_i^j tem valor 1 se o elemento pertence a j -ésima classe e 0 caso contrário. Assim, $1 < \text{sum}(\mathbf{y}_i) < k$, em que $\text{sum}(\mathbf{y}_i)$ é o número de rótulos com valor 1 em \mathbf{y}_i .

Problemas de classificação multirrótulo ocorrem com frequência nas áreas de Bioinformática e processamento de textos. O estudo de métodos de classificação multirrótulo tem sido motivado, principalmente, pelas tarefas de classificação de textos (Gonçalves e Quaresma, 2003; Lauser e Hotho, 2003; Luo e Zincir-Heywood, 2005). Em um problema de classificação de textos, cada documento pode pertencer simultaneamente a mais de uma classe (ou tópico). Um documento, por exemplo, pode ser classificado como pertencente à área de Ciência da Computação e Física. Um artigo de jornal que aborda as reações de políticos a pacotes econômicos pode ser classificado ao mesmo tempo nas categorias Política e Economia.

Um exemplo de problema de classificação multirrótulo na área de Bioinformática é a classificação da função de proteínas, pois uma proteína pode ter mais de uma função.

Problemas de classificação multirrótulo podem também ser encontrados em várias outras áreas, como diagnóstico médico (Karalic e Pernat, 1991; Tsoumakas e Katakis, 2007), classificação de imagens (Boutell et al., 2004; Shen et al., 2004) e bioinformática (Clare e King, 2001; Zhang e Zhou, 2005; Elisseeff e Weston, 2001b). Na área de diagnóstico médico, um paciente pode sofrer de diabetes e gripe ao mesmo tempo. Um problema de classificação multirrótulo de imagens é a classificação de fotografias de paisagens, quando uma mesma imagem pode ser classificada, por exemplo, como de bosque e de pôr do sol. Um exemplo de aplicação na área de Bioinformática é a definição das funções de uma proteína, uma vez que uma proteína pode ter uma ou mais funções simultaneamente associadas a ela. Portanto, o estudo de técnicas de classificação capazes de prover uma solução eficiente para esses problemas pode trazer grandes benefícios para várias áreas práticas.

A Figura 18.1 apresenta uma comparação entre um problema de classificação conven-

cional, em que exemplos podem ser associados a apenas uma classe, e um problema de classificação multirrótulo. A Figura 18.1(a) ilustra um problema de classificação em que os exemplos pertencem ou à classe \blacksquare ou à classe \blacktriangle , mas nunca às duas classes ao mesmo tempo. A Figura 18.1(b) mostra um exemplo de classificação multirrótulo em que os exemplos pertencentes simultaneamente às classes \blacksquare e \blacktriangle são representados por \star .

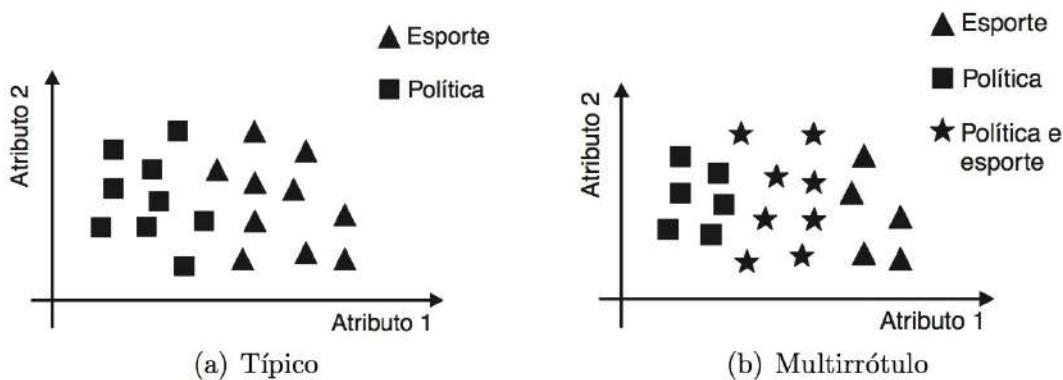


Figura 18.1 Problemas de classificação

Três diferentes abordagens foram propostas na literatura para tratar problemas de classificação multirrótulo. Em uma delas, classificadores simples-rótulo são combinados para tratar problemas de classificação multirrótulo. Na segunda, classificadores simples-rótulo são modificados por meio de adaptações em seus mecanismos internos, de forma a permitir que sejam utilizados em problemas multirrótulo. Na última abordagem, novos algoritmos são desenvolvidos especificamente para tratar problemas de classificação multirrótulo (de Carvalho e Freitas, 2009).

Este capítulo está organizado com a seguinte estrutura. Na Seção 18.1 são discutidas as principais abordagens para lidar com problemas de classificação multirrótulo. Dois conceitos importantes para classificação multirrótulo, de cardinalidade de rótulo e densidade de rótulo, são explicados na Seção 18.2. A Seção 18.3 apresenta algumas das principais medidas de avaliação utilizadas nessa área. As considerações finais deste capítulo são objeto da Seção 18.4.

18.1 Principais Abordagens

Quando um classificador é treinado, uma probabilidade pode ser associada a cada uma das classes existentes no problema e então ser utilizada para a classificação de um novo exemplo. Se o problema de classificação tem k classes, uma probabilidade p_i , com $1 \leq i \leq k$, na qual $0 \leq p_i \leq 1$, pode ser atribuída a cada classe. Quando o classificador é treinado em um problema de classificação simples-rótulo, há uma restrição que diz que $\sum p_i = 1$. Em problemas de classificação multirrótulo, essa restrição não é adotada (de Carvalho e Freitas, 2009). Problemas de classificação binária e multiclasse podem ser considerados casos especiais de problemas de classificação multirrótulo, em que o número de classes atribuídas a cada exemplo é igual a 1 (Elisseeff e Weston, 2001a).

A Figura 18.2 ilustra os diferentes métodos propostos na literatura para tratar problemas de classificação multirrótulo (de Carvalho e Freitas, 2009). De acordo com a figura, esses métodos podem ser divididos em duas grandes abordagens: abordagem independente de algoritmo e abordagem dependente de algoritmo. Como pode ser observado na figura,

a abordagem independente de algoritmo utiliza algoritmos tradicionais de classificação para tratar problemas multirrótulo, transformando o problema multirrótulo original em um conjunto de problemas simples-rótulo. A abordagem dependente de algoritmo cria algoritmos específicos para tratar o problema multirrótulo. Esses algoritmos podem ser baseados em técnicas de classificação convencionais, como SVMs e árvores de decisão, ou podem ser especificamente desenvolvidos para classificação multirrótulo.

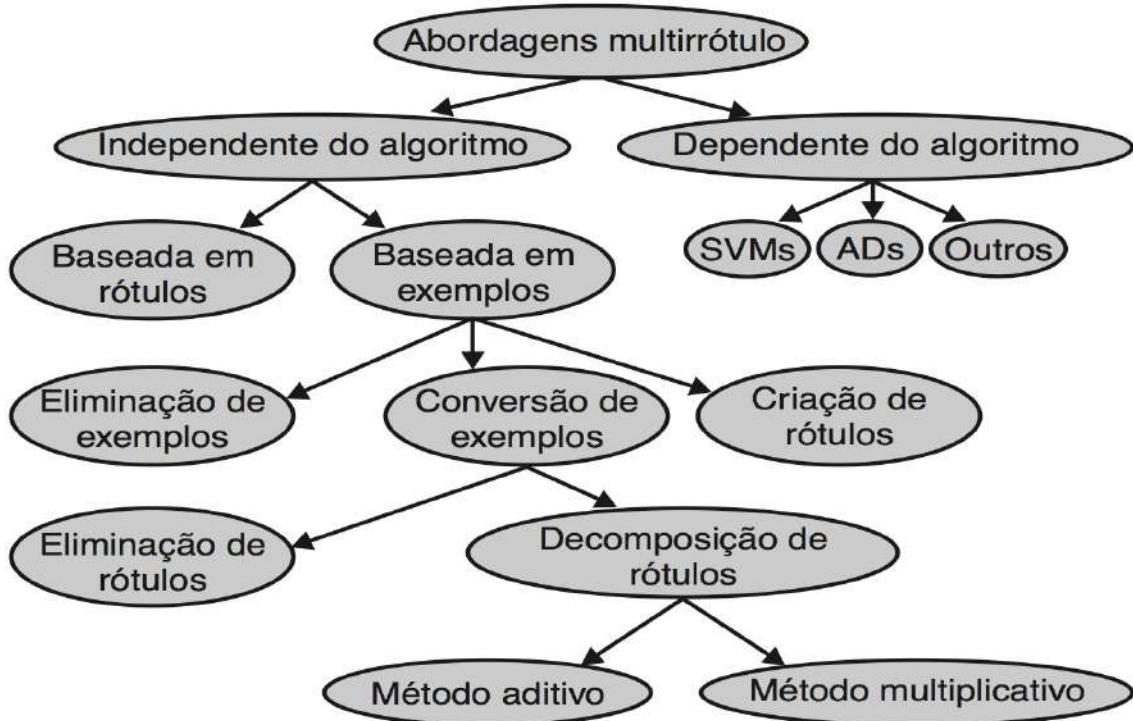


Figura 18.2 Métodos para classificação multirrótulo.

18.1.1 Abordagem Independente de Algoritmo

Nessa abordagem, qualquer algoritmo tradicional de classificação pode ser utilizado para tratar o problema. Para isso, basta transformar o problema multirrótulo original em um conjunto de problemas de classificação simples-rótulo. Essa transformação pode ser baseada nos rótulos de classes dos exemplos ou nos próprios exemplos de treinamento.

Transformação Baseada nos Rótulos das Classes

Nesse tipo de transformação, são utilizados k classificadores, e k é o número de classes do problema. Cada classificador é então associado a uma classe e treinado para resolver um problema de classificação binária, na qual é considerada a classe à qual ele está associado contra todas as outras classes envolvidas. Esse método é também chamado de método binário ou um-contra-todos (Tsoumakas e Vlahavas, 2007).

Um ponto fraco desse método é que ele assume que as classes atribuídas a um exemplo são independentes entre si. Isso nem sempre é verdade, e ignorar as possíveis correlações entre as classes pode fazer com que o método tenha pouca capacidade de generalização.

O processo de transformação desse método é reversível, ou seja, é possível recuperar as classes do problema original a partir do novo problema criado. O método utiliza k classificadores, em que k é o número de classes.

Transformação Baseada nos Exemplos

Nessa abordagem, o conjunto de classes associado a cada exemplo é redefinido, de maneira a converter o problema multirrótulo original em um ou mais problemas simples-rótulo. Ao contrário do método anterior, esse método não produz apenas problemas de classificação binária, podendo produzir problemas tanto binários quanto multiclasse.

Três diferentes estratégias são propostas para esse tipo de transformação (de Carvalho e Freitas, 2009):

- Eliminação de exemplos multirrótulo;
- Criação de novos rótulos para os exemplos multirrótulo existentes;
- Conversão de exemplos multirrótulo em exemplos simples-rótulo.

Eliminação de Exemplos Multirrótulo

A estratégia mais simples que existe da transformação baseada em exemplos, mas também a mais ineficaz, é eliminar do conjunto de dados os exemplos que são multirrótulo. A eliminação dos exemplos com mais de uma classe não resolve o problema multirrótulo original. Ela apenas muda o problema, transformando-o em outro mais simples e provavelmente não tão relevante quanto o original.

Um exemplo de perda de informação causada por essa estratégia pode ser dado por um problema de classificação de proteínas. Proteínas podem desempenhar mais de uma função, e eliminar essas proteínas do conjunto de dados reduz a significância do classificador (de Carvalho e Freitas, 2009), que não teria como predizer as outras funções de uma proteína. Essa transformação é irreversível, pois não é possível descobrir, no novo problema criado, quais exemplos foram eliminados do problema original. O número de classificadores necessários também não é alterado.

Criação de Novos Rótulos para os Exemplos Multirrótulo Existentes

Nessa estratégia, para cada exemplo, todas as classes atribuídas àquele exemplo são combinadas em uma nova e única classe. Com essa combinação, o número de classes envolvidas no problema pode aumentar consideravelmente, e algumas classes podem terminar com poucos exemplos que as representem.

Com a criação de novas classes, as classes do problema original não são perdidas. Se forem utilizados classificadores multiclasse no novo problema criado, a quantidade desses classificadores se mantém a mesma. Se forem utilizados classificadores binários, o número de classificadores necessários aumenta.

Conversão de Exemplos Multirrótulo em Exemplos Simples-rótulo

Existem duas variações para essa estratégia. Na primeira, todos os exemplos multirrótulo são convertidos em exemplos simples-rótulo, em um processo chamado de simplificação ou eliminação de rótulos. Uma segunda variação, chamada de decomposição de rótulos, decompõe todos os exemplos multirrótulo em um conjunto de exemplos simples-rótulo.

Na simplificação de rótulos, quando um exemplo possui mais de uma classe, uma de suas classes é escolhida e as outras são eliminadas. Essa escolha pode ser feita de maneira determinística, selecionando, dentre as classes às quais o exemplo pertence, aquela que possui mais chance de ser a classe verdadeira, ou pode ser feita de maneira aleatória, selecionando uma classe randomicamente.

Se na escolha das classes for adotado um critério determinístico, é possível retornar ao problema multirrótulo original a partir do problema simples-rótulo criado. Se as classes forem escolhidas de maneira randômica, esse retorno não é possível. O número de classificadores utilizados no problema multirrótulo e no simples-rótulo geralmente é o mesmo.

No processo de decomposição de rótulos, um problema multirrótulo com k classes e n exemplos é dividido em ps conjuntos de problemas simples-rótulo. O valor de ps varia de 1, quando nenhum exemplo possui mais do que uma classe, a $(k - 1)^n$, se todos os exemplos possuem $k - 1$ classes. O processo de decomposição pode ser dividido em dois métodos: aditivo e multiplicativo (de Carvalho e Freitas, 2009).

No método aditivo, o número de classificadores é igual ao número de classes que rotulam pelo menos um exemplo multirrótulo. Esse método permite que o problema multirrótulo original seja recuperado a partir do problema simples-rótulo criado.

No método multiplicativo, por outro lado, é utilizada uma combinação de todos os possíveis problemas simples-rótulo. Esse método é similar ao método todos-contra-todos (explicado no Capítulo 17), utilizado para dividir um problema multiclasse em um conjunto de problemas binários. O número de classificadores utilizados no método multiplicativo é igual a $\prod k_i$, que é o produtório do número de classes presentes em cada exemplo do conjunto de dados.

O número de classificadores desse método cresce exponencialmente com o número de classes de cada exemplo do conjunto de dados. É possível notar também que o método aditivo produz um subconjunto de problemas simples-rótulo produzidos pelo método multiplicativo. O método multiplicativo também é reversível, permitindo a recuperação do problema multirrótulo original. O método multiplicativo também minimiza a deficiência de métodos que eliminam ou combinam rótulos e perdem informação. Porém, ele não leva em consideração possíveis relações entre os rótulos de classes de um mesmo exemplo.

Muitos dos métodos propostos para tratar problemas multirrótulo de forma independente fazem uso de SVMs (Cristianini e Shawe-Taylor, 2000). No trabalho de Pavlidis e Grundy (1999) foram utilizadas SVMs para a classificação multirrótulo de funções de genes, utilizando decomposição binária. Para os experimentos, os autores utilizaram uma base de dados heterogênea, consistindo em dados de expressão gênica e perfis filogenéticos. De acordo com os autores, esses dois tipos de dados proporcionam uma visão mais exata de subconjuntos sobrepostos de categorias de funções gênicas presentes em uma célula, resultando em um melhor desempenho na classificação. Foi observado também que essa melhoria não é uniformemente distribuída entre as classes presentes no problema, portanto essa combinação só deve ser utilizada se há evidências de seu benefício.

Em Su et al. (2005), um problema multirrótulo foi decomposto em um conjunto de problemas binários utilizando a estratégia um-contra-todos. Os experimentos foram feitos utilizando um conjunto de dados de predição da localização subcelular de proteínas.

18.1.2 Abordagem Dependente de Algoritmo

Na abordagem dependente de algoritmo, como o próprio nome sugere, novos algoritmos são propostos para tratar os problemas de classificação multirrótulo como um todo, em uma única etapa. Um algoritmo específico, feito para um determinado problema de classificação real e difícil, pode apresentar resultados melhores do que métodos que seguem a abordagem independente de algoritmo. Embora diversas soluções baseadas nessa abordagem tenham sido propostas, as principais dizem respeito a algoritmos de indução de árvores de decisão.

Um novo método de classificação baseado em árvores de decisão, chamado de “Árvore de Decisão Alternada” (ADT), foi proposto por Freund e Mason (1999). Esse método é uma generalização das árvores de decisão, e seu princípio indutivo é baseado no método *boosting* (Freund e Schapire, 1999). Uma extensão do método ADT foi proposta por de Comite et al. (2003), e é baseada nos métodos *AdaBoost* (Freund e Schapire, 1995) e *ADTBoost* (Freund e Mason, 1999). Esse algoritmo estende o ADT pela decomposição de problemas multiclasse usando a abordagem um-contra-todos (de Carvalho e Freitas, 2009).

Outro trabalho que utiliza árvores de decisão foi proposto por Clare e King (2001). Nesse trabalho, os autores modificaram o algoritmo C4.5 (Quinlan, 1993) para a classificação de proteínas de acordo com suas funções. O algoritmo C4.5 define os nós da árvore de decisão através de uma medida chamada entropia. Os autores modificaram a fórmula dessa medida, originalmente elaborada para problemas simples-rótulo, de maneira a permitir seu uso em problemas multirrótulo. Outra modificação feita pelos autores foi a utilização dos nós folha da árvore para representar conjuntos de rótulos de classes. Quando um nó folha, alcançado na classificação de um exemplo, contém um conjunto de classes, uma regra separada é produzida para cada classe (de Carvalho e Freitas, 2009).

Em Zhang e Zhou (2005) é proposto um novo método para classificação multirrótulo baseado no algoritmo k -NN, chamado ML- k -NN. Nesse método, para cada exemplo, as classes associadas aos k exemplos vizinhos mais próximos são recuperadas, e é feita uma contagem dos vizinhos associados a cada classe. Então, o princípio *maximum a posteriori* (Saridis, 1983) é utilizado para definir o conjunto de classes de um novo exemplo.

Em Schapire e Singer (1999) e Schapire e Singer (2000) são propostas duas extensões para o algoritmo *Adaboost* (Freund e Schapire, 1995), de maneira a permitir seu uso em problemas multirrótulo. Na primeira, é feita uma modificação na maneira de se avaliar o desempenho preditivo do modelo induzido, verificando sua capacidade de predizer um conjunto correto de classes para um dado exemplo. Na segunda, uma mudança no algoritmo faz com que ele passe a predizer um *ranking* de classes para cada exemplo de entrada.

O número de exemplos multirrótulo e classes de uma base de dados pode influenciar o desempenho dos métodos de classificação. A próxima seção faz algumas considerações a respeito disso.

18.2 Densidade e Cardinalidade de Rótulo

Os conjuntos de dados não são todos igualmente multirrótulo. Em alguns casos, o número de classes de cada exemplo é pequeno se comparado ao número total de exemplos n , enquanto em outros, esse número é grande. Esse número pode ser um parâ-

metro que influencia o desempenho dos diferentes métodos de classificação multirrótulo (Tsoumakas e Katakis, 2007). Nesta seção são apresentados os conceitos de cardinalidade de rótulo e densidade de rótulo em um conjunto de dados. Sendo \mathbf{X} um conjunto de dados multirrótulo consistindo em n exemplos multirrótulo $(\mathbf{x}_i, \mathbf{y}_i)$, com $i = 1, 2, \dots, n$, pode-se definir cardinalidade e densidade como:

- A cardinalidade de rótulo de \mathbf{X} é dada pelo número médio de rótulos dos exemplos de \mathbf{X} :

$$CR(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^n sum(\mathbf{y}_i) \quad (18.1)$$

- A densidade de rótulo de $|D|$ é dada pelo número médio de rótulos dos exemplos de \mathbf{X} dividido por $|k|$, o número total de classes:

$$DR(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \frac{sum(\mathbf{y}_i)}{|k|} \quad (18.2)$$

Nessas equações, $sum(\mathbf{y}_i)$ é o número de rótulos do objeto \mathbf{x}_i . A cardinalidade de rótulo é independente do número de possíveis classes, k , e é utilizada para quantificar o número de rótulos alternativos que caracterizam os exemplos de um conjunto de dados multirrótulo. A densidade de rótulo leva o número de possíveis rótulos em consideração. Dois conjuntos de dados com a mesma cardinalidade de rótulo, mas com uma grande diferença no número de rótulos (diferentes densidades de rótulo), podem apresentar propriedades diferentes, que podem afetar o desempenho dos algoritmos de classificação multirrótulo. Supor, por exemplo, dois conjuntos de dados com a mesma cardinalidade de dois rótulos por objeto e com diferentes densidades, dois rótulos por objeto dentre quatro possíveis classes e dois rótulos por objeto dentre 40 possíveis classes. O número de possíveis combinações no segundo caso é bem maior que no primeiro caso. Essas duas métricas são relacionadas uma com a outra: $CR(\mathbf{X}) = k \times DR(\mathbf{X})$ (Tsoumakas e Katakis, 2007).

Para avaliar o desempenho dos classificadores multirrótulo, algumas medidas foram propostas na literatura. A próxima seção apresenta algumas delas.

18.3 Medidas de Avaliação

A avaliação de classificadores multirrótulo requer métricas diferentes das utilizadas em problemas de classificação simples-rótulo. Diferentemente da classificação simples-rótulo, em que um exemplo é classificado de maneira errada ou correta, na classificação multirrótulo, um exemplo pode ser classificado de maneira parcialmente errada ou parcialmente correta. Esse casos acontecem quando um classificador atribui corretamente a um exemplo pelo menos uma das classes a que ele pertence, mas também não atribui ao exemplo uma ou mais classes às quais ele pertence. Pode acontecer também de o classificador atribuir a um exemplo uma ou mais classes às quais ele não pertence. Nesta seção serão apresentadas algumas das métricas propostas na literatura para avaliar classificadores multirrótulo.

Basicamente, o critério de avaliação utilizado pode ser baseado na classificação multirrótulo feita pelo classificador, que utiliza os rótulos atribuídos por um classificador a

um dado exemplo, ou baseado em uma função de *ranking*, em que, para cada exemplo, o classificador produz um *ranking* de rótulos.

Seja \mathbf{X} um conjunto de dados multirrótulo consistindo em n exemplos multirrótulo $(\mathbf{x}_i, \mathbf{y}_i)$, com $i = 1, 2, \dots, n$ e $\text{sum}(\mathbf{y}_i) < k$, em que k é o conjunto de possíveis classes. Sejam ainda \hat{f} um classificador multirrótulo e $\mathbf{z}_i = \hat{f}(\mathbf{x}_i)$ um vetor binário com k elementos representando o conjunto de classes preditas por \hat{f} para um dado exemplo \mathbf{x}_i .

Para a avaliação baseada na classificação, uma medida muito comum é o *Hamming Loss*. No trabalho de Schapire e Singer (2000), essa medida foi utilizada. Ela é definida na Equação 18.3.

$$\text{HammingLoss}(\hat{f}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \frac{a(\mathbf{y}_i, \mathbf{z}_i)}{k} \quad (18.3)$$

Nessa medida, o $a(\mathbf{y}_i, \mathbf{z}_i)$ representa a distância de Hamming entre dois vetores, e corresponde à operação *XOR* da lógica booleana (Tsoumakas e Katakis, 2007). Quanto menor for o valor do *Hamming Loss*, melhor é a classificação. A situação perfeita ocorre quando o seu valor é igual a zero.

No mesmo trabalho, os autores utilizaram outras medidas de avaliação baseadas em *ranking*. Essas medidas são chamadas na literatura de *one-error*, *coverage* e *average precision*. A medida *one-error* calcula o número de vezes que um rótulo com a melhor posição no *ranking* de rótulos não está presente no conjunto de rótulos corretos de um dado exemplo de entrada (Shen et al., 2004). Na medida *coverage*, é calculado o quanto longe, em média, se deve percorrer o *ranking* de rótulos ordenados de maneira a atribuir, a um exemplo de entrada, todos os rótulos que deveriam ser atribuídos a ele (de Carvalho e Freitas, 2009). O terceiro método, originalmente proposto para a tarefa de recuperação de informação, avalia a proporção média de rótulos que ocupam, no *ranking*, uma posição superior a um dado rótulo particular e que pertencem ao conjunto de rótulos desejados (de Carvalho e Freitas, 2009).

Outras métricas (acurácia, precisão e revocação) foram utilizadas no trabalho de Godbole e Sarawagi (2004). Elas são apresentadas nas Equações 18.4, 18.5 e 18.6.

$$\text{acurácia}(\hat{f}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \frac{\mathbf{y}_i \text{AND} \mathbf{z}_i}{\mathbf{y}_i \text{OR} \mathbf{z}_i} \quad (18.4)$$

$$\text{precisão}(\hat{f}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \frac{\mathbf{y}_i \text{AND} \mathbf{z}_i}{\text{sum}(\mathbf{z}_i)} \quad (18.5)$$

$$\text{revocação}(\hat{f}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \frac{\mathbf{y}_i \text{AND} \mathbf{z}_i}{\text{sum}(\mathbf{y}_i)} \quad (18.6)$$

em que AND e OR representam as operações booleanas *bitwise AND* e *OR* aplicadas a dois vetores binários.

No trabalho de Boutell et al. (2004), é utilizada uma versão mais generalizada da acurácia, apresentada na Equação 18.4. Nessa versão, é utilizado um parâmetro $\alpha \geq 0$, chamado de “taxa de perdão” (*forgiveness rate*). Essa taxa reflete o quanto penalizar erros cometidos na predição dos rótulos de classes. Quanto menor o valor de α , mais os erros são tolerados, e quanto maior o valor de α , a penalização de erros é maior. O cálculo é apresentado na Equação 18.7.

$$\text{acurácia}(\hat{f}, \mathbf{X}) = \frac{1}{n} \sum_{i=1}^n \left(\frac{\mathbf{y}_i AND \mathbf{z}_i}{\mathbf{y}_i OR \mathbf{z}_i} \right)^\alpha \quad (18.7)$$

18.4 Considerações Finais

Este capítulo apresentou os conceitos de classificação multirrótulo em que um exemplo pode pertencer simultaneamente a mais de uma classe. Vários problemas reais apresentam esse perfil, principalmente problemas de classificação de textos e de Bioinformática.

Após uma contextualização desse grupo de problemas de classificação, foram descritos os conceitos básicos e apresentadas as principais abordagens utilizadas para lidar com esses problema de classificação. Essas abordagens combinam técnicas tradicionais de classificação ou desenvolvem uma nova técnica, que pode ser obtida pela alteração de procedimentos utilizados por técnicas convencionais. Em seguida, foram discutidos os conceitos de cardinalidade e densidade e mostradas algumas medidas de desempenho que podem ser utilizadas para avaliação de classificadores multirrótulo. Alguns problemas de classificação multirrótulo são também problemas de classificação hierárquica. O próximo capítulo apresenta os conceitos de classificação hierárquica.

Capítulo 19

Classificação Hierárquica

Nos problemas de classificação hierárquica, as classes podem apresentar uma relação de taxonomia ou dependência, formando subclasses e superclasses. A classificação plana, em que não se tem uma relação hierárquica entre as classes, é o tipo mais comum de classificação. Contudo, se a hierarquia das classes é conhecida ou pode ser construída, sua consideração pode levar à indução de classificadores com maior acurácia preditiva.

Como exemplo de aplicação, as funções exercidas por uma proteína no meio celular podem ser organizadas hierarquicamente. A indução de um classificador para a previsão da função de uma nova proteína, dadas suas características, configura assim um problema de classificação hierárquica (Clare e King, 2003). Outro exemplo são os problemas de categorização de textos. Classes de textos correlatos são comumente agrupadas em tópicos, os quais, por sua vez, também podem ser agrupados em temas principais (Sun e Lim, 2001). Assim, um texto que trata de uma partida de futebol pode ser classificado na seção de esportes, que por sua vez pode também incluir tópicos como basquete, futebol, tênis, entre outros ligados ao tema. Desse modo, o modelo induzido deve ser capaz de classificar o texto mencionado tanto na categoria futebol quanto em sua supercategoria esporte. Silla e Freitas (2010) também reportam o uso de hierarquias na classificação de gêneros musicais, de fonemas, de objetos 3D, de animais e de imagens.

É importante também destacar que, embora uma hierarquia de classes também tenha sido introduzida como uma possibilidade de solução de problemas multiclasse (Seção 17.1.4), esse último não é considerado um problema de classificação hierárquica, que se configura apenas quando as classes possuem uma relação taxonômica inerente.

Ao se induzir modelos de classificação para problemas hierárquicos, a relação hierárquica entre as classes deve ser levada em consideração. Este capítulo apresenta os principais conceitos relacionados a problemas de classificação hierárquica (Seção 19.1), as possíveis abordagens para a sua solução (Seção 19.2) e algumas técnicas específicas para avaliação de classificadores hierárquicos (Seção 19.3), tendo como base os trabalhos de Freitas e de Carvalho (2007) e Silla e Freitas (2010).

19.1 Tipos de Problemas

É possível distinguir diferentes tipos de problemas de classificação hierárquica, que variam de acordo com três características (Silla e Freitas, 2010):

1. Tipo de hierarquia em que as classes se organizam, que pode ser em forma de árvore ou de um DAG.

2. Se os dados podem ou não seguir mais de um caminho na hierarquia. Caso possam, tem-se um problema de classificação hierárquica com múltiplos rótulos.
3. A profundidade das rotulações dos dados. Tem-se dois casos: todos os objetos possuem rotulação até os nós folha, que representam os níveis mais profundos da hierarquia; ou ao menos um dos objetos possui uma rotulação parcial, que não atinge um nó folha.

Em relação ao tipo de hierarquia, as classes em problemas hierárquicos podem se organizar de duas formas, de árvore e de DAG, ilustradas nas Figuras 19.1(a) e 19.1(b), respectivamente. Em ambas as estruturas, cada nó corresponde a uma classe. Iniciando pelo nó raiz, as classes são particionadas recursivamente em subclasses.

Um nó que aponta para outro nó é chamado de pai do segundo, que é então filho do primeiro. Como exemplo, considerando a hierarquia da Figura 19.1(a), o nó 2 é pai dos nós 2.2 e 2.3, que são seus filhos. Nós que compartilham o mesmo pai são chamados de irmãos. Logo, 2.2 e 2.3 são irmãos na Figura 19.1(a). Quando um nó possui outros nós conectados a si em um nível mais baixo, ele é chamado de nó interno ou não folha, e, quando isso não ocorre, ele é chamado de nó folha. Nessa figura, todos os nós do terceiro nível da hierarquia são folhas.

O conteúdo dos nós indica a classe associada a ele. Para isso, é utilizada uma lista numérica para os diferentes níveis, em que, a cada passagem de nível, as superclasses correspondentes são adicionadas à notação. Logo, para um dado nó, o número de listagem mais à direita indica sua classe no nível em que ele está, e o(s) número(s) à esquerda indica(m) seu(s) pai(s) em níveis mais altos. Pode-se notar, por exemplo, no DAG da Figura 19.1(b), que o nó 2.2.2 – 2.3.1 no segundo nível tem dois nós do primeiro nível, 2.2 e 2.3, como pais. O nó raiz indica qualquer classificação, pois um item nesse nível pode pertencer a qualquer uma das classes.

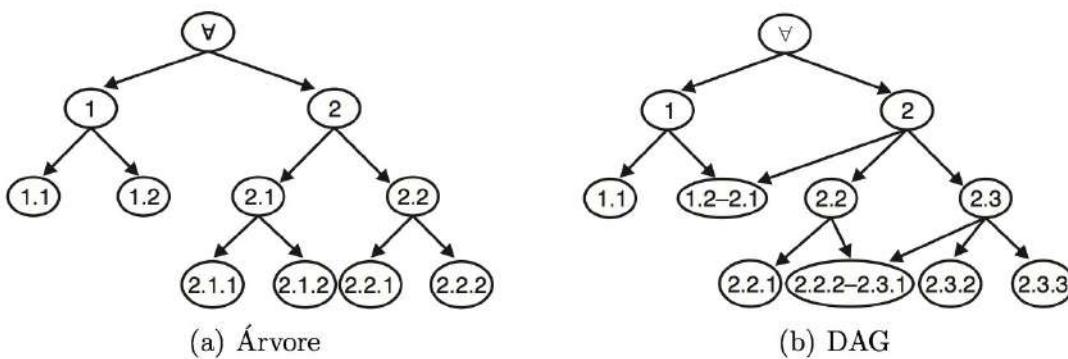


Figura 19.1 Exemplos de hierarquias.

A distinção entre o DAG e a árvore está no fato de que na primeira estrutura um nó pode possuir mais de um pai (como o nó 1.2 – 2.1). As árvores na realidade são um caso particular de DAG em que todos os nós têm um único pai. Os ancestrais de um determinado nó na hierarquia são os nós que se encontram no(s) caminho(s) da raiz a esse nó, incluindo ele próprio. Por exemplo, os ancestrais do nó 2.1.1 na Figura 19.1(a) são os nós 2.1.1, 2.1, 2 e a raiz.

Embora nos problemas de classificação hierárquica os dados naturalmente apresentem múltiplos rótulos, uma vez que um objeto pertencente a uma subclasse também pertence

às suas superclasses, é comum nomear-se como problemas de classificação hierárquica multirrótulo aqueles em que as previsões podem seguir múltiplos caminhos na hierarquia. Por exemplo, se um dado pertencer simultaneamente às classes 2.1 e 2.2 (ou 1.1 e 2.1) da hierarquia da Figura 19.1(a).

Em grande parte dos problemas de classificação hierárquica, é comum que todos os exemplos possuam rótulos que correspondem a nós folha. Esse caso corresponde a uma rotulação completa em profundidade, ou seja, todo objeto possui rótulos de classes em todos os níveis em um caminho da hierarquia, da raiz a uma folha. Porém, pode haver exemplos para os quais a classificação para em um nó interno da hierarquia, tipicamente porque a classificação em níveis mais profundos ainda é desconhecida para eles. Tem-se então uma rotulação parcial em profundidade.

As características que distinguem os tipos de problemas de classificação hierárquica são resumidas a seguir. Cada combinação de valores de característica leva a um problema distinto. Por exemplo, um determinado problema pode ser caracterizado por possuir uma hierarquia estruturada em árvore, em que os dados possuem rótulos que podem seguir múltiplos caminhos na hierarquia (multirrótulo) e há dados para os quais os rótulos correspondem a nós internos (não folha). A partir da determinação dessas características, é possível então escolher uma determinada estratégia algorítmica apropriada à solução do problema.

- *Tipo de hierarquia:*
 - Árvore;
 - DAG.
- *Caminho das rotulações:*
 - Múltiplo (multirrótulo);
 - Único;
- *Profundidade das rotulações:*
 - Completa (nós folha);
 - Parcial.

19.2 Algoritmos para Classificação Hierárquica

Anteriormente discorreu-se sobre os tipos de problemas de classificação hierárquica, distinguidos de acordo com características associadas à hierarquia que seguem e às rotulações dos objetos em seus conjuntos de dados. Com relação aos algoritmos existentes na literatura para a solução de problemas de classificação hierárquica, também é possível fazer uma distinção, de acordo com quatro características (Silla e Freitas, 2010):

1. O tipo de hierarquia com que o algoritmo pode lidar, árvore ou DAG.
2. Se ele pode rotular os dados de acordo com múltiplos caminhos na hierarquia (multirrótulo) ou não.

3. A profundidade das predições do algoritmo, se são sempre em nós folha ou não.
4. Os tipos de classificadores que são usados na resolução do problema hierárquico: locais ou globais. Para classificadores locais, tem-se ainda as possibilidades de usar um classificador por nó, um classificador por nó pai ou um classificador por nível.

Os itens 1 a 3 claramente se relacionam com aos itens 1 a 3 que distinguem os problemas de classificação hierárquica descritos na seção anterior, evidenciando a importância da determinação das características do problema para a escolha de uma abordagem de solução apropriada.

Algoritmos capazes de realizar predições para hierarquias do tipo DAG também são capazes de lidar com hierarquias de árvores, uma vez que elas são DAGs simplificadas. Contudo, algoritmos capazes de lidar apenas com hierarquias de árvores devem ser significativamente estendidos para lidar com DAGs. Pode-se considerar, portanto, problemas com a hierarquia em DAG mais complexos que aqueles com estrutura de árvore.

A rotulação em múltiplos caminhos na hierarquia implica a possibilidade de solução de problemas de classificação hierárquica multirrotulo. Existem algoritmos que realizam predições seguindo apenas um caminho na hierarquia. Contudo, é possível também realizar transformações no problema de múltiplos caminhos, transformando-o em um ou mais problemas de caminho único, usando abordagens como as descritas no Capítulo 18, adaptadas para o caso hierárquico.

A profundidade de predição do algoritmo está relacionada a ele sempre realizar predições em nós folha (predição mandatória em nós folha) ou a ele ser capaz realizar predições em qualquer nível da hierarquia, incluindo as folhas (predição não mandatória em nós folha). Embora a confiabilidade da predição possa ser maximizada na predição não mandatória em nós folha, a classificação torna-se menos específica e pode assim ter menor utilidade.

Na Figura 19.2 são ilustradas as diferentes abordagens empregadas na solução de problemas de classificação hierárquica de acordo com o tipo de classificadores usados. Nós destacados com bordas pontilhadas adicionais representam os classificadores usados em cada caso. Por simplicidade, considera-se nesses exemplos uma hierarquia estruturada como árvore. Existe ainda uma quinta abordagem bastante simplificada para resolver problemas de classificação hierárquica, que consiste em usar um único classificador plano,¹ geralmente distinguindo somente as classes em nós folha. A predição em uma classe 1.2, por exemplo, também indica que esse exemplo em particular pertence à classe 1. Isso corresponde a simplesmente transformar o problema hierárquico em um único problema de classificação plana e pode ser usado tanto no caso de árvores quanto de DAGs. Essa abordagem possui a desvantagem de poder ter que distinguir um grande número de classes (todas as folhas) sem explorar as informações de relacionamentos hierárquicos presentes. Além disso, as predições nesse caso são obrigatoriamente mandatórias em nós folha.

A seguir são descritas as abordagens mais usuais para classificação hierárquica, ilustradas na Figura 19.2.

¹Classificadores planos são aqueles induzidos na solução de problemas de classificação planos, em que não há uma relação hierárquica de classes.

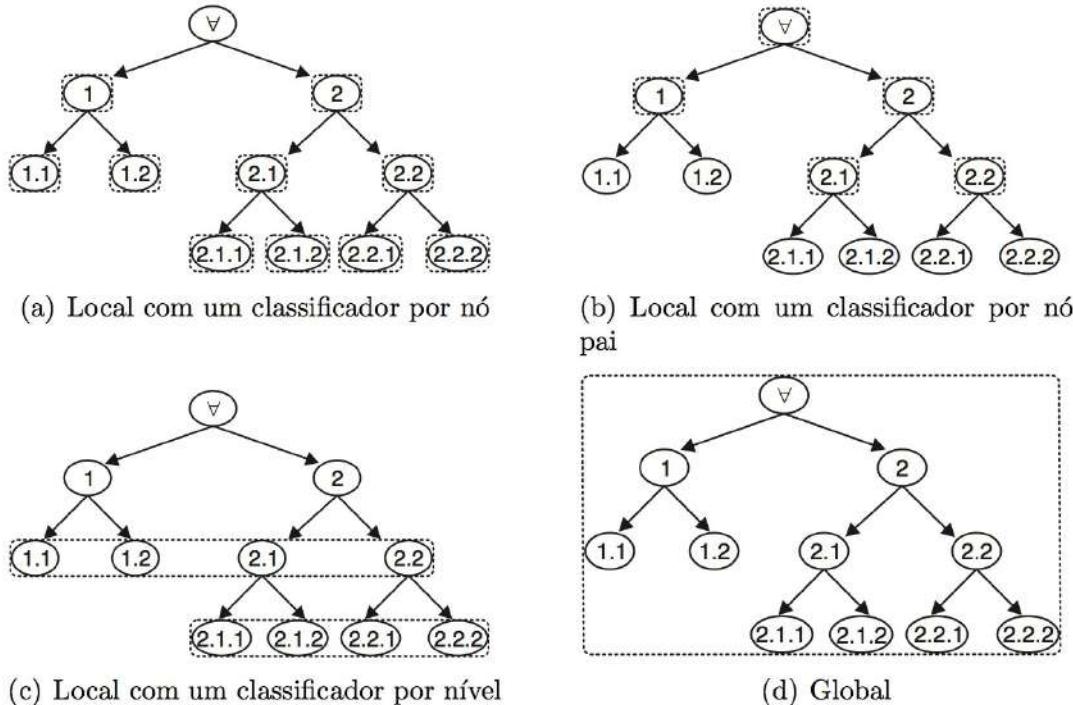


Figura 19.2 Abordagens para a classificação hierárquica.

19.2.1 Classificadores Locais

Inicialmente, destacam-se abordagens que usam informações locais da hierarquia no processo de solução (casos a, b e c na Figura 19.2). Há três maneiras padrão de usar informação local, empregando um classificador plano por: (a) nó da hierarquia, exceto a raiz; (b) nó pai da hierarquia; (c) nível da hierarquia, com exceção da raiz. O uso de informação local permite o emprego direto de classificadores planos na solução do problema hierárquico.

O uso de um classificador plano por nó que representa uma classe na hierarquia é a estratégia mais usada e implica a combinação de classificadores binários, um por nó da hierarquia, a menos da raiz (Figura 19.2(a)). Diferentes estratégias podem ser usadas para determinar que exemplos serão considerados positivos e negativos pelo algoritmo de classificação utilizado em cada nó (Eisner et al., 2005; Fagni e Sebastiani, 2007). Uma das políticas mais usadas para essa definição é a de irmãos, em que, para um dado nó representando a classe c_j , os exemplos positivos são aqueles com rótulo c_j e de suas subclasses, enquanto os exemplos negativos são os de nós irmãos de c_j , segundo a hierarquia, e de suas respectivas subclasses. Considerando o classificador para a classe 2.1 na Figura 19.2(a), por exemplo, os objetos positivos na política de irmãos são os de rótulo 2.1, 2.1.1 e 2.1.2, enquanto os negativos são os de rótulo 2.2, 2.2.1 e 2.2.2. Outras políticas podem ser consultadas em Silla e Freitas (2010).

O uso de um classificador plano por nó pai é ilustrado na Figura 19.2(b). Para cada um desses nós, devem-se distinguir as subclasses contidas em seus nós filhos, usando classificadores multiclasse ou combinações de classificadores binários, como as apresentadas no Capítulo 17. Essa estratégia não costuma ser empregada em hierarquias do tipo DAG, uma vez que pode haver muita redundância nos conjuntos de treinamento de diferentes classificadores.

Na Figura 19.2(c) tem-se o uso de um classificador plano multiclasse por nível da

hierarquia (portanto, três classificadores no exemplo considerado). Essa abordagem é a menos utilizada na literatura da área, pois podem haver conflitos nas predições realizadas para os diferentes níveis.

Na etapa de teste, uma estratégia comumente adotada para prever a classe de um novo exemplo, utilizando classificadores locais, é denominada *top-down* (de cima para baixo). Outras estratégias são revistas em (Silla e Freitas, 2010). Nessa estratégia, partindo do(s) classificador(es) em níveis iniciais da hierarquia, as predições obtidas são usadas para escolher um novo classificador até atingir um nó folha ou por fim decidir parar a predição em um nível intermediário da hierarquia. Uma desvantagem dessa estratégia é que, quando um exemplo é erroneamente classificado em um determinado nível, esse erro será provavelmente propagado para os níveis seguintes. Isso sempre ocorre em uma estrutura em árvore, mas pode não acontecer para uma hierarquia DAG.

Se o problema não exige predições mandatórias em nós folha, algum mecanismo pode ser empregado para parar a classificação em algum nível. Por exemplo, o classificador no próximo nível é consultado somente se a confiança na predição atual é maior que um determinado limiar. Problemas de classificação multirrótulo podem também ser tratados pela consideração de múltiplos caminhos na hierarquia de acordo com os níveis de confiança das predições obtidas.

19.2.2 Classificadores Globais

Os classificadores globais consideram a hierarquia como um todo na etapa de treinamento e não possuem a modularidade característica das abordagens locais, como ilustrado na Figura 19.2(d). O modelo final obtido nessa abordagem, também denominada *big-bang*, pode ser menor do que aquele formado por múltiplos classificadores locais. Contudo, esse classificador tende a ser mais complexo do que os individuais induzidos na abordagem local. Logo, na abordagem *big-bang*, geralmente um único modelo é induzido. Na etapa de teste, esse modelo pode ser empregado potencialmente na predição de qualquer classe da hierarquia.

Em Kiritchenko et al. (2006), por exemplo, o problema hierárquico é transformado em um plano multirrótulo. Para tal, são adicionados a cada exemplo de treinamento os rótulos dos nós antecessores.

Existem ainda diferentes algoritmos de AM que foram adaptados para considerarem a hierarquia como um todo durante a etapa de treinamento, gerando um único classificador capaz de realizar predições hierárquicas. Uma adaptação do algoritmo C4.5 para a indução de árvores de decisão é apresentada em Clare e King (2003). Nesse trabalho, o algoritmo C4.5 (Quinlan, 1993) foi modificado para a tarefa de classificação hierárquica. A ideia básica dessa modificação foi substituir a fórmula da entropia, utilizada para decidir qual atributo será selecionado para um dado nó da árvore, por uma entropia ponderada. Essa entropia ponderada leva em consideração que classes em níveis mais elevados da hierarquia tendem a ter menores valores de entropia do que classes em níveis mais profundos. Contudo, classes em níveis mais profundos são preferíveis, pois fornecem conhecimento mais específico. Silla e Freitas (2009) apresentam uma adaptação do algoritmo *naive Bayes*.

Há também algoritmos especialmente desenvolvidos para a classificação hierárquica, como o Clus-HMC (Blockeel et al., 2002), que combina árvores baseadas na noção de *predictive clustering trees*.

19.3 Avaliação de Classificadores Hierárquicos

Dada a particularidade de as previsões terem que seguir uma hierarquia predefinida, as medidas de avaliação de classificadores hierárquicos devem ser adaptadas. Por exemplo, é possível considerar previsões em níveis mais profundos mais úteis, dando pesos diferentes a classificações obtidas em níveis distintos da hierarquia. Outra possibilidade é permitir classificações parcialmente corretas, em que as previsões são corretas para apenas alguns níveis da hierarquia. Não existe ainda consenso em relação às medidas a serem adotadas, e muitos trabalhos se limitam a usar medidas de desempenho para classificadores planos, o que não é a melhor opção.

As medidas de avaliação hierárquicas podem ser genericamente divididas como baseadas (Costa et al., 2007):

- Em distância;
- Em profundidade;
- Em similaridade (semântica) entre as classes e a hierarquia;
- Nas relações de descendência e/ou ancestralidade das classes na hierarquia.

Em um dos primeiros trabalhos que propõe adaptações em medidas de desempenho para o cenário da classificação hierárquica, Sun e Lim (2001) modificaram as medidas de precisão, sensibilidade, acurácia e erro de maneira a considerar a distância na hierarquia entre a classe predita e classe real, a qual é dada pelo número de arestas que separam esses nós na hierarquia. Para a Figura 19.1(a), por exemplo, a distância entre 1.2 e 2.2 é 4, e a distância entre 2.1 e 2.2 é 2. Maiores distâncias implicam maior erro. De fato, se um novo objeto pertence à classe 2.2, é pior classificá-lo na classe 1.2 do que na classe 2.1, pois no último caso ao menos as classes 2.1 e 2.2 pertencem à mesma superclasse na hierarquia, a classe 2.

Entretanto, a distância não leva em consideração o fato de que as classificações em níveis mais profundos são mais difíceis de ser realizadas e são geralmente mais úteis. Isso pode ser corrigido pela associação de pesos a cada aresta, de acordo com sua posição na hierarquia, diminuindo o peso com a profundidade (Blockeel et al., 2002). Contudo, a definição dos valores de pesos a serem empregados geralmente não é trivial, sendo dependente da hierarquia. Para DAGs, devido aos múltiplos caminhos existentes para uma mesma previsão, o uso dessa métrica pode ser considerado pouco viável.

Sun e Lim (2001) também propuseram considerar a similaridade entre as classes para medir os erros de classificação, em uma abordagem semântica. Cada classe é inicialmente descrita por um vetor de características, e a similaridade entre as classes predita e real é calculada a partir desses vetores. Essa similaridade é então usada para definir novas medidas de precisão, revocação, acurácia e erro. Essas medidas foram aplicadas ao domínio de categorização hierárquica de textos, podendo não ser adequadas para outros domínios.

As medidas apontadas como mais interessantes no contexto de classificação hierárquica por Silla e Freitas (2010) são as que se baseiam nas relações de ancestralidade na hierarquia, propostas por Kiritchenko et al. (2004). Nessas medidas, é considerado o número de ancestrais comuns existentes entre a(s) classe(s) verdadeira(s) do exemplo e aquela(s) predita(s) pelo classificador. No cálculo da precisão, esse valor é dividido pelo número de ancestrais da(s) classe(s) predita(s), como ilustrado na Equação 19.1 para um objeto

em particular, em que c_p representa a(s) classe(s) predita(s), c_v corresponde à(s) classe(s) verdadeira(s) do objeto, $Ancestral(z)$ fornece o conjunto de antecessores de um determinado nó z na hierarquia e $|Z|$ é o operador que fornece a cardinalidade do conjunto Z . É importante destacar que o conjunto $Ancestral(Z)$ inclui a classe Z . A raiz da hierarquia não é considerada um ancestral.

$$P_H = \frac{|Ancestral(c_p) \cap Ancestral(c_v)|}{|Ancestral(c_p)|} \quad (19.1)$$

Na medida de revocação, o número de ancestrais em comum é dividido pelo número de ancestrais da(s) classe(s) verdadeira(s) do objeto, conforme a Equação 19.2.

$$R_H = \frac{|Ancestral(c_p) \cap Ancestral(c_v)|}{|Ancestral(c_v)|} \quad (19.2)$$

Para obtenção da precisão e revocação em um conjunto contendo n dados, basta somar as medidas anteriores calculadas para cada objeto individual.

Essas medidas são consideradas mais genéricas e podem ser aplicadas em diferentes tipos de problemas de classificação hierárquica, a saber: árvore ou DAG, multirrotulo ou não, com profundidade de rotulação completa. Deficiências podem ocorrer somente no caso de a profundidade de rotulação ser parcial (predição não mandatória em nós folha), pois o desconhecimento da classe real mais específica é penalizado, mas ele pode ser resultado do fato de o domínio ainda ser desconhecido e previsões obtidas em níveis mais profundos podem estar trazendo informações valiosas.

19.4 Considerações Finais

Neste capítulo foram apresentados os conceitos fundamentais de classificação hierárquica, comum em domínios como a categorização de textos e a Bioinformática. Nesse tipo de problema, as classes seguem uma relação hierárquica, que deve ser levada em consideração na etapa induativa e preditiva. Os tipos de problemas hierárquicos e de abordagens algorítmicas para tratá-los foram apresentados, sendo distinguidos de acordo com a hierarquia tratada, os caminhos e a profundidade das classificações e, no caso dos algoritmos, também na abordagem seguida na solução do problema.

Por fim, foi discutida a questão da necessidade de considerar as características dos problemas hierárquicos para avaliar um classificador obtido nesse domínio. Embora muitos trabalhos se limitem a usar medidas de desempenho para classificadores planos, é indicado considerar as particularidades introduzidas por uma hierarquia de classes.

Capítulo 20

Computação Natural

A Computação natural é uma área de pesquisa associada a AM, a Estatística e a Otimização que tem apresentado um forte crescimento nos últimos anos. A Computação Natural se inspira em processos que ocorrem na natureza para o desenvolvimento de novos algoritmos que possam ser utilizados em problemas reais. Essa inspiração tem origem nos mais diversos processos, o que levou ao desenvolvimento de várias técnicas, como, por exemplo, RNAs, computação evolutiva, computação quântica, computação molecular, autômatos celulares e geometria fractal.

Um dos principais argumentos em favor da computação natural é que quando fenômenos naturais complexos são computacionalmente analisados e modelados, é possível compreender melhor a natureza e utilizar esse conhecimento para criar novos algoritmos computacionais. Pesquisas nessa área também têm investigado a utilização de materiais naturais, como moléculas de DNA, para realizar tarefas de computação. Esses novos materiais podem complementar os materiais atualmente utilizados na construção de computadores digitais (Castro, 2007). Uma subárea da computação natural, conhecida como computação bioinspirada, se inspira, particularmente, em processos biológicos.

Este capítulo descreve técnicas de computação bioinspirada utilizadas em AM. Esas técnicas compõem uma classe de meta-heurísticas (Blum e Roli, 2003; Dorigo et al., 2006). Uma meta-heurística é um processo de geração iterativo que guia uma heurística subordinada pela combinação inteligente de diferentes conceitos para exploração e exploração do espaço de busca. Nesse processo, estratégias são utilizadas para encontrar de forma eficiente soluções próximas de ótimas (Osman e Laporte, 1996). De acordo com Maniezzo et al. (2004), as meta-heurísticas utilizam algumas heurísticas básicas para escapar do problema de obtenção de mínimos locais: iniciam a busca a partir de uma solução inicial (ou conjunto de soluções iniciais) e adicionam elementos até obterem uma boa solução ou iniciam com uma solução completa e iterativamente modificam alguns de seus elementos até que seja atingido um critério de parada.

As seções seguintes apresentam algumas das principais técnicas de computação bioinspirada propostas na literatura. Na Seção 20.1 são introduzidos os conceitos básicos de inteligência de enxames, quando são apresentadas duas de suas variações, otimização por colônia de formigas e otimização por enxame de partículas. Computação evolutiva, uma das técnicas mais populares da computação bioinspirada, é explicada na Seção 20.2. Na última seção, Seção 20.3, são feitas as considerações finais para este capítulo.

20.1 Inteligência de Enxames

Na natureza, a inteligência de enxames, ou inteligência coletiva, diz respeito a agentes (indivíduos) que apresentam um nível superior de inteligência dentro do comportamento social. Os indivíduos devem ser capazes de interagir entre si e com o ambiente. A incorporação de vida social em uma meta-heurística permite incorporar no processo de busca aspectos como maior facilidade de encontrar alimento, melhor divisão de trabalho, melhor aproveitamento das capacidades de cada indivíduo e como evitar predadores e facilitar a operação de caça (Castro, 2006).

Os algoritmos baseados em inteligência de enxames manipulam indivíduos simples que atuam de forma auto-organizada, isto é, sem nenhuma forma de controle central sobre os membros do enxame. Segundo Millonas (1994), os sistemas baseados em inteligência coletiva seguem cinco princípios:

1. **Proximidade:** indivíduos de uma população devem interagir entre si.
2. **Qualidade:** indivíduos devem ser capazes de avaliar a interação entre si e com o ambiente.
3. **Diversidade:** a capacidade de um sistema reagir a ações inesperadas.
4. **Estabilidade:** os indivíduos não podem modificar seu comportamento em resposta a qualquer modificação no ambiente.
5. **Adaptabilidade:** os indivíduos devem ser capazes de se adaptar às mudanças do ambiente e da população.

As técnicas otimização por colônia de formigas e otimização por enxame de partículas são variações da inteligência de enxames. A primeira é baseada no comportamento de formigas na busca por alimentos, e a segunda, na organização existente entre bandos de animais, como pássaros e peixes, e no comportamento social do ser humano.

20.1.1 Otimização por Colônia de Formigas

Colônias de formigas inspiraram o desenvolvimento de várias meta-heurísticas. Dentre essas, a mais estudada e de maior sucesso é uma técnica de otimização de propósito geral conhecida como ACO (do inglês, *Ant Colony Optimization*) (Dorigo et al., 2006). ACO é inspirada no comportamento das formigas na busca por alimento. O principal aspecto desse comportamento é a comunicação que ocorre entre formigas de uma colônia por meio do depósito de feromônio nas trilhas percorridas.

Inicialmente, durante a busca por alimentos, as formigas exploram de maneira aleatória uma dada região. Durante a movimentação, essas formigas depositam feromônio pelo solo ao longo do caminho percorrido (Blum, 2005). O feromônio é uma substância química cujo odor é sentido pelas formigas. Ao escolher um caminho dentre mais de uma opção, estudos sugerem que as formigas escolham o caminho marcado com uma maior concentração de feromônio. Como a probabilidade de as formigas que alcançaram o alimento pelo menor caminho retornarem antes das que escolheram o caminho mais longo é maior, o menor caminho ficará com uma maior concentração de feromônio e, provavelmente, será o caminho seguido pelas próximas formigas.

A Figura 20.1 ilustra essa busca pelo menor caminho, mostrando o aumento do número de formigas (círculos cinza) que utilizam o caminho mais curto entre o ninho (N) e a fonte de alimento (A) ao longo três instantes de tempo. Com passar do tempo, o menor caminho possuirá a maior quantidade de feromônio depositado, atraindo um número maior de formigas.

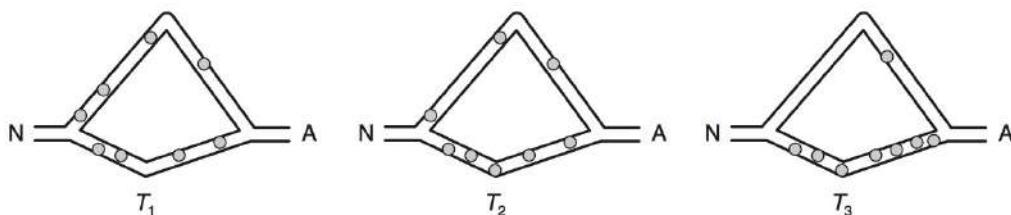


Figura 20.1 Formigas em busca de alimento.

ACO explora um mecanismo semelhante para resolver problemas de otimização e foi formalizado como meta-heurística para problemas de otimização combinatorial por Dorigo e Di-Caro (1999). Em um problema de otimização por maximização, ele gera como solução o indivíduo que maximiza o valor de uma função objetivo. Uma função objetivo simples seria o menor caminho entre dois pontos. Assim, ACO pode ser facilmente utilizada no problema do caixeiro-viajante.¹ Mas a função objetivo pode representar qualquer problema de otimização.

O funcionamento da ACO pode ser resumido como um conjunto de agentes computacionais concorrentes e assíncronos (como uma colônia de formigas) que se movimentam através de estados do problema, que correspondem às soluções parciais no espaço de busca. A movimentação dos agentes é baseada em dois parâmetros: trilha e atratividade. Durante seu movimento, cada formiga constrói incrementalmente uma solução para o problema. Durante a fase de construção ou finalização de uma solução, a formiga avalia a solução encontrada e modifica o valor de feromônio associado com as trilhas usadas. O total de feromônio nas trilhas será utilizado pelas demais formigas na busca pela melhor solução (Maniezzo et al., 2004).

Um algoritmo ACO inclui dois mecanismos adicionais: evaporação da trilha e, opcionalmente, ações. A evaporação da trilha decrementa a quantidade de feromônio em todas as trilhas com o passar do tempo, com o intuito de evitar o acúmulo ilimitado de feromônio. Ações podem ser utilizadas para permitir ações centralizadas, o que não acontece com as colônias naturais de formigas (Maniezzo et al., 2004).

Em ACO para problemas combinatoriais, formigas constroem soluções incrementalmente. Inicialmente cada formiga começa com um conjunto vazio de soluções. A cada passo de construção, uma componente da solução é adicionada ao conjunto de soluções. A definição de componente da solução é dependente da aplicação. Para o problema do caixeiro-viajante, por exemplo, uma componente da solução é uma cidade que é adicionada ao percurso. Para escolher qual componente da solução deve ser adicionada ao conjunto, é feita uma escolha probabilística, considerando-se normalmente o total de feromônio associado a uma determinada componente e uma possível informação heurística sobre o problema (Socha, 2004).

¹No problema do caixeiro-viajante, dados um grupo de cidades e a distância entre elas, busca-se pelo caminho que visite todas as cidades percorrendo a distância mínima.

ACO foi desenvolvida inicialmente para resolver problemas de otimização combinatorial, logo a versão original não cobre problemas de otimização com variáveis contínuas. Uma extensão para a ACO, desenvolvida por Socha (2004), permite a otimização de problemas com variáveis contínuas e mistas (discretas e contínuas). Posteriormente, Socha e Dorigo (2008) desenvolveram o ACO_R para problemas de domínios contínuos. A ideia principal do ACO_R é substituir a distribuição de probabilidade discreta utilizada no ACO convencional por uma contínua, isto é, por uma função de densidade de probabilidade.

No Algoritmo 20.1, é apresentado um pseudocódigo com os principais passos do algoritmo ACO.

Algoritmo 20.1 Algoritmo ACO

Entrada: Uma função objetivo capaz de avaliar cada solução e um grafo com os possíveis caminhos

Saída: Um conjunto de soluções otimizadas

1 Gerar aleatoriamente uma população inicial de formigas

2 Inicializar valores de feromônio nas arestas

3 repita

4 **para cada** *formiga* **faça**

5 Mover a formiga por uma caminho completo até a construção de uma solução para o problema

6 Avaliar sua aptidão para resolver o problema

7 Aumentar quantidade de feromônio nas arestas percorridas

8 Evaporar parte do feromônio de todas as arestas

9 **fim**

10 **até** *Critério de parada for satisfeito*;

20.1.2 Otimização por Enxame de Partículas

PSO (do inglês, *particle swarm optimization*) é uma técnica de otimização global desenvolvida por Kennedy e Eberhart (1995) e inicialmente introduzida para otimização de funções contínuas não lineares. Ela é fortemente baseada no conceito de que o compartilhamento de informações entre indivíduos oferece uma vantagem evolutiva. Seu desenvolvimento foi inspirado no comportamento social de pássaros e peixes e no comportamento social dos seres humanos.

Na técnica PSO, as soluções são chamadas de partículas. As partículas se movimentam por um espaço de busca e são capazes de armazenar informações passadas e compartilhar informações com outras partículas. Esses dois tipos de informações correspondem à aprendizagem individual (cognitiva) e à transmissão cultural (social). Dessa forma, as partículas utilizam as melhores soluções no seu processo de “evolução”. Kennedy e Eberhart (2001) usaram três princípios para explicar de forma breve o processo de adaptação cultural:

1. **Avaliar:** cada partícula deve avaliar a solução encontrada por ela no espaço de busca.

2. **Comparar:** cada partícula deve comparar a solução obtida por ela com as soluções obtidas pelas demais partículas.
3. **Imitar:** as partículas devem imitar o funcionamento da partícula que mais se aproximou da solução desejada.

O compartilhamento de informações é realizado por partículas topologicamente vizinhas. A vizinhança considerada pelo PSO não é definida pelos valores de atributos de cada partícula. A vizinhança pode ser global (ou estrela), quando cada partícula é vizinha de todas as outras, e local (ou anel), em que os vizinhos de cada partícula são seus k vizinhos mais próximos (Castro, 2006). Na vizinhança global, cada partícula compartilha informação com todas as outras. As Figuras 20.2(a) e 20.2(b) apresentam exemplos de topologias com cinco partículas para vizinhança global e local, respectivamente.

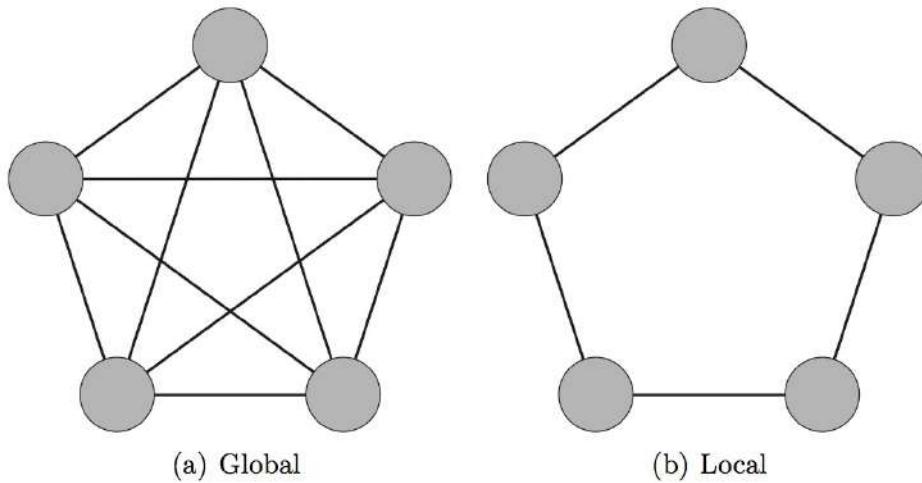


Figura 20.2 Gráfico ilustrativo para vizinhança.

O algoritmo PSO pode ser aplicado a problemas tanto binários quanto contínuos. Cada partícula é representada pela sua posição atual, sua velocidade e a melhor posição encontrada por ela. Cada partícula é tratada como um ponto em um espaço d -dimensional. A posição da partícula l é dada por $P_l = (p_{l1}, p_{l2}, \dots, p_{ld})$; sua velocidade é dada por $V_l = (v_{l1}, v_{l2}, \dots, v_{ld})$, e a melhor posição encontrada por essa partícula por $M_l = (m_{l1}, m_{l2}, \dots, m_{ld})$ (Castro, 2006). Para o caso da vizinhança global, a melhor posição encontrada entre todas as partículas é representada pelo símbolo M_g .

Uma partícula irá se mover em uma determinada direção em função da sua posição atual, da sua velocidade, da melhor posição encontrada por ela e da melhor posição encontrada pelos seus vizinhos. As Equações 20.1 e 20.2 determinam a forma com que a velocidade e a posição das partículas são atualizadas, respectivamente. Nessas equações, t representa a iteração, w , o peso da inércia, cujo papel é balancear a busca global e a local (introduzido por Shi e Eberhart (1998)), r_1 e r_2 são dois valores aleatórios independentes distribuídos uniformemente no intervalo $[0, 1]$ (utilizados para dar um comportamento estocástico), e φ_1 e φ_2 são taxas de aprendizado. Para evitar a explosão de velocidade das partículas, a velocidade máxima é limitada. A posição da partícula pode também ser restrita ao intervalo do espaço de busca utilizado. No Algoritmo 20.2 são descritos os principais passos do algoritmo PSO.

$$v_{ld}(t+1) = w \cdot v_{ld}(t) + \varphi_1 \cdot r_1 \cdot (m_{ld} - p_{ld}(t)) + \varphi_2 \cdot r_2 \cdot (m_{gd} - p_{ld}(t)) \quad (20.1)$$

$$p_{ld}(t+1) = p_{ld}(t) + v_{ld}(t) \quad (20.2)$$

Algoritmo 20.2 Algoritmo PSO

Entrada: Uma função objetivo capaz de avaliar cada solução

Saída: Um conjunto de soluções otimizadas

```

1 Inicializar as partículas
2 repita
3   para cada partícula faça
4     Avaliar sua aptidão para resolver o problema
5   fim
6   Escolher partícula com maior aptidão
7   para cada partícula faça
8     Atualizar sua velocidade
9     Atualizar sua posição
10  fim
11 até Critério de parada for satisfeito;
```

Ao final da execução do algoritmo, a posição da partícula com melhor avaliação será a melhor solução encontrada para resolver o problema de otimização. Cada componente do vetor que representa a posição será uma parte dessa solução.

20.2 Computação Evolutiva

A computação evolutiva (CE) compreende um conjunto de técnicas computacionais para resolução de problemas baseados na Genética e na Teoria da Evolução. Suas técnicas podem ser divididas nas seguintes áreas:

- Algoritmos genéticos (AGs);
- Estratégias evolutivas (EEs);
- Programação evolutiva (PE).

Como aspectos de cada área são continuamente assimilados por outros, é difícil definir fronteiras entre elas. Tanto EEs quanto PE foram originalmente propostas para otimização de funções contínuas. AGs em geral são utilizados em problemas de otimização combinatória. Para exemplificar o uso dessas técnicas, serão detalhados nesta seção os principais aspectos dos AGs, que concentram o maior número de aplicações utilizando CE.

Segundo a Teoria da Evolução, os organismos que melhor se adaptam a seu ambiente têm maiores chances de ter suas características reproduzidas em uma nova geração. A Genética explica como os filhos herdam características dos pais.

A utilização desses conceitos em um algoritmo computacional para a solução de um problema real ocorre, de forma simplificada, da seguinte maneira: é gerada uma população inicial de possíveis soluções para o problema. Busca-se então, em um processo iterativo, gerar uma boa solução por meio da evolução das melhores soluções da população atual, de acordo com uma função de aptidão, que mede a qualidade de cada solução. Para a definição de cada nova população a partir das soluções escolhidas, três operadores genéticos são geralmente aplicados: elitismo (cópias simples das melhores soluções), cruzamento (combinação de partes de pares de soluções) e mutação (altera a composição de algumas soluções, permitindo a criação de soluções que ainda não foram observadas).

Em AGs, uma solução é também chamada de indivíduo ou cromossomo. Um aspecto importante em relação aos AGs é a forma de representação de uma solução para o problema tratado.

Frequentemente, os indivíduos são representados por vetores binários. Nesses vetores, cada elemento, denominado gene, representa a presença (valor 1) ou ausência (valor 0) de alguma característica (Carvalho et al., 2003). Com essa representação, indivíduos podem ser representados por sequências de valores binários. Valores inteiros e valores reais também têm sido utilizados. O tipo dos valores depende da natureza do problema tratado.

Quando AGs são utilizados, o processo de busca por uma boa solução ocorre em várias gerações. A cada geração, mecanismos de seleção tendem a selecionar os indivíduos mais aptos, e operadores de reprodução geram uma nova população de indivíduos. Para a avaliação de cada indivíduo de uma população é utilizada uma função de aptidão, que mede a qualidade da solução representada por um indivíduo para o problema tratado. Embora diferentes implementações de AGs variem em alguns aspectos, seu funcionamento básico pode ser sumarizado pelo Algoritmo 20.3.

Algoritmo 20.3 Algoritmo genético básico

Entrada: Uma função objetivo capaz de avaliar cada solução

Saída: Um conjunto de soluções otimizadas

- 1 Gerar população inicial de indivíduos
 - 2 repita
 - 3 para cada *indivíduo da população* faça
 - 4 Avaliar sua aptidão para resolver o problema
 - 5 fim
 - 6 Selecionar indivíduos que participarão da próxima geração
 - 7 Aplicar operadores genéticos aos indivíduos selecionados
 - 8 até *Critério de parada for satisfeito*;
-

No Algoritmo 20.3, a população inicial geralmente ocorre pela atribuição de valores aleatórios aos genes dos indivíduos, que podem, dessa forma, ser representados por sequências de valores binários.

No processo de seleção, são selecionados de forma probabilística os indivíduos mais aptos da população atual. Existem vários métodos para a seleção dos indivíduos. Os mais conhecidos são seleção por roleta e seleção por torneio. O uso de probabilidade faz com que o indivíduo com maior valor de aptidão tenha maiores chances de participar da fase de reprodução. Como resultado, indivíduos cada vez mais aptos são gerados, e indivíduos menos aptos tendem a desaparecer.

Na seleção por roleta, cada indivíduo da população é representado em roleta por uma fatia proporcional ao seu índice de aptidão. Para selecionar n_i indivíduos, a roleta é girada n_i vezes, selecionando a cada rodada o indivíduo cuja fatia é apontada pela agulha da roleta. Na seleção por torneio, a seleção de n_s indivíduos para a fase de reprodução se dá pela realização de n_s torneio. A cada torneio, n_t indivíduos são aleatoriamente escolhidos da população atual, e o indivíduo que vencer o torneio é selecionado para a fase de reprodução.

Na fase de reprodução, são aplicados operadores genéticos sobre os indivíduos selecionados, criando novos indivíduos. Os principais operadores genéticos são os operadores de cruzamento (ou recombinação) e de mutação.

O operador de cruzamento é probabilisticamente aplicado a pares de indivíduos selecionados na fase de reprodução, de acordo com uma taxa de cruzamento. O operador de cruzamento tradicional produz dois novos indivíduos (filhos) a partir de dois indivíduos selecionados (pais). No cruzamento de um ponto, o mais simples, um ponto de corte divide cada pai em duas partes. Cada filho recebe uma parte de um dos pais. Dessa forma, ele permite que características dos dois pais passem para as próximas gerações. A Figura 20.3(a) ilustra o funcionamento do operador de cruzamento de um ponto.

O operador de mutação permite introdução e manutenção de diversidade genética na população. Para isso, ele altera aleatoriamente um ou mais componentes de uma estrutura escolhida. Sua aplicação é definida de forma probabilística utilizando uma taxa de mutação. O uso de um operador de mutação faz com que qualquer ponto do espaço de busca possa ser atingido. Em geral, o operador de mutação é aplicado a cada indivíduo após a aplicação do operador de cruzamento. A Figura 20.3(b) mostra como funciona o operador de mutação.

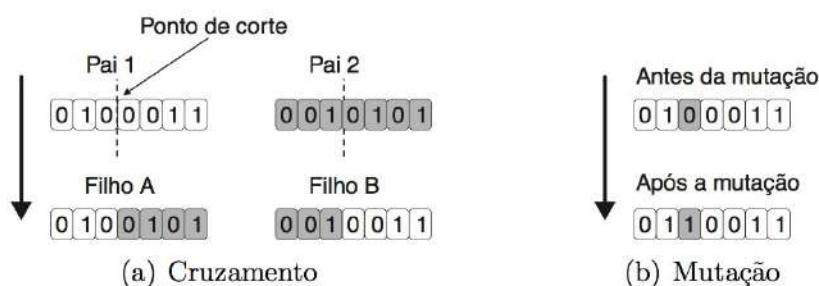


Figura 20.3 Operadores genéticos.

Outro operador muito utilizado é o operador de elitismo. Quando ele é utilizado, o indivíduo de maior desempenho é automaticamente selecionado para a próxima geração. Isso evita modificações desse indivíduo pelos operadores genéticos. Ele é utilizado para que os melhores indivíduos não desapareçam da população. O operador de elitismo também pode selecionar mais de um indivíduo por geração.

A sequência de operações seleção-reprodução se repete até que seja atingido um critério de parada. Diferentes critérios de parada têm sido utilizados, como um número máximo de repetições do ciclo (gerações), a obtenção de pelo menos uma solução satisfatória, ou a falta de melhoria, por um dado número de gerações, do valor de aptidão do melhor indivíduo (Carvalho et al., 2003).

Algumas das vantagens dos AGs residem no fato de estes realizarem buscas simultâneas em várias regiões do espaço de soluções (Michalewicz, 1996). Isso permite que se encontrem diversas soluções, e os torna métodos de busca global. Por esse motivo, os AGs também são definidos como técnicas de otimização global. Além disso, os AGs são capazes de realizar buscas em espaços de soluções (hipóteses) com regiões complexas, em que o impacto de cada parte na solução do problema pode ser de difícil modelagem por meio de técnicas convencionais (Mitchell, 1997).

Por outro lado, uma das principais desvantagens dos AGs é seu custo computacional. Dependendo da complexidade da aplicação investigada, seu uso pode ser computacionalmente caro. Contudo, os AGs são facilmente paralelizáveis, podendo tirar proveito do crescente uso de tecnologias distribuídas.

Existem duas variações de AGs que obtiveram grande destaque nas últimas décadas: Programação Genética (PG) (Koza, 1992), que é utilizada para evoluir programas, e Sistemas Classificadores (LCSs, do inglês *Learning Classifier Systems*) (Sigaud e Wilson, 2007), que é empregada para evoluir regras de classificação.

Por alguns classificados como uma subárea da CE, a PG permite evoluir programas. Para isso, as soluções são em geral representadas por grafos em forma de árvores em que os nós internos representam comandos, chamadas de funções e operações, e os nós externos ou nós folha representam variáveis, constantes ou comandos de entrada ou saída de dados. Na PG, a população inicial é formado por um conjunto de programas de computador, usualmente gerados de forma aleatória. A avaliação de cada indivíduo ocorre pela execução do programa que ele representa. Para que os programas possam ser executados, os indivíduos devem ser programas sintaticamente válidos.

Os LCSs utilizam conceitos de AGs para evoluir uma população de regras ou componentes de regras para lidar com um problema de classificação. Nesses sistemas, a aptidão ou qualidade de uma regra pode ser estimada por diferentes critérios, como sua acurácia preditiva e comprehensibilidade.

20.3 Considerações Finais

Este capítulo se iniciou com a apresentação de duas das principais áreas da computação bioinspirada, um subgrupo da computação natural: inteligência de enxames e computação evolutiva. Posteriormente, os conceitos envolvidos na inteligência de enxames foram brevemente descritos, e foram abordadas as meta-heurísticas bioinspiradas otimização por colônia de formigas e otimização por enxame de partículas, pertencentes a esse paradigma.

Na seção seguinte foram apresentados os principais conceitos da CE, um dos tópicos mais populares da computação bioinspirada. Após mostrar as diferentes variações de CE, foram apresentados os principais passos seguidos pelos AGs para a solução de problemas. Foram apresentados ainda os operadores de seleção e de reprodução mais utilizados nos trabalhos com AGs. Existem várias outras técnicas de computação bioinspiradas, como os sistemas imunes artificiais e a computação baseada em moléculas.

Parte V

Aplicações

Aplicações

Conforme comentado no capítulo de introdução deste livro, técnicas de AM têm sido utilizadas com reconhecido sucesso para a solução de vários problemas reais. Alguns desses problemas foram brevemente comentados, em alguns casos com exemplos de conjuntos de dados.

A primeira etapa no uso de AM em problemas reais é compreender o problema, quais são suas principais características e o que se espera de uma boa solução. Esse entendimento vai definir as possíveis técnicas a serem utilizadas e os procedimentos a serem adotados. Por exemplo, o problema é descritivo ou preditivo? Quantos e quais são os atributos de entrada? Existem valores ausentes? Os dados disponíveis apresentam ruído? Qual o tamanho do conjunto de dados? Novos dados são continuamente gerados?

Os dados em aplicações reais podem apresentar diferentes formatos, como tabelas atributo-valor, séries temporais, *itemsets*, transações, grafos ou redes sociais, textos, páginas web, imagens (vídeos) e áudios (músicas). Em geral, como visto no Capítulo 2, dados nesses formatos precisam transformados para o formato atributo-valor.

É de fundamental importância observar vários aspectos ao utilizar técnicas de AM em aplicações reais. Vários desses aspectos já foram comentados em capítulos anteriores que discutem procedimentos para planejamento de experimentos e análise de resultados. No desenvolvimento de uma ferramenta computacional que utiliza técnicas de AM, o desempenho da técnica no problema abordado deve ser experimentalmente validado.

O planejamento dos experimentos a serem realizados é essencial para a obtenção de resultados, e consequentemente conhecimentos, válidos. Todo o esforço despendido na realização de experimentos pode ser perdido se um planejamento adequado não for realizado. Adicionalmente, os resultados obtidos nos experimentos realizados precisam ser avaliados nas métricas apropriadas e devem ser analisados por meio de medidas estatísticas, como, por exemplo, testes de hipóteses. Existem várias métricas para análise de desempenho, como erro quadrático médio, área sob a curva ROC, índice silhueta. A escolha da métrica a ser utilizada depende da natureza do problema a ser atacado e dos dados disponíveis. Testes de hipóteses permitem afirmar que uma certa técnica, ou variação de uma técnica, apresenta um desempenho melhor que outra com uma dada certeza. Um grande número de testes pode ser utilizado, cada um mais adequado para uma dada situação.

Em alguns problemas, operações de pré-processamento e transformação podem ser necessárias. Decidir que operações devem ser realizadas, qual a sequência com que elas serão realizadas e que técnicas utilizar em cada uma dessas operações pode exercer uma forte influência na qualidade da solução desenvolvida.

A forma como as possíveis soluções ou hipóteses induzidas por técnicas de AM, assim como ocorre com o processo de indução, também pode variar de acordo com a aplicação. Uma técnica de AM pode ser utilizada em um problema de classificação em que a acurácia

preditiva não é a única medida de desempenho relevante. Por exemplo, o desenvolvimento de ferramentas baseadas em AM para serem utilizadas em dispositivos móveis em geral precisa limitar a quantidade de memória utilizada. Outro exemplo é o uso de técnicas de AM em aplicações que precisam lidar com fluxos contínuos de dados. Nessas aplicações, o algoritmo de treinamento utilizado pode precisar ser utilizado um grande número de vezes, favorecendo algoritmos de menor custo computacional.

Neste capítulo, alguns dos principais problemas em que técnicas de AM foram utilizadas são organizados em grandes grupos de aplicações. Vários dos problemas descritos podem estar em diferentes grupos. Os grupos utilizados foram escolhidos para aumentar a homogeneidade das aplicações em cada grupo. Assim, os seguintes grupos foram definidos:

- Agropecuária;
- Bioinformática;
- Ecologia e meio ambiente;
- Energia;
- Finanças;
- Mineração de dados (incluindo textos e *web*);
- Robótica;
- Saúde;
- Telecomunicações.

Deve ser observado que muitos problemas estão relacionados a mais de uma área. Por exemplo, um problema na área de agropecuária, como melhoramento genético de soja, pode também estar relacionado a bioinformática e ao meio ambiente. A seguir, será apresentada uma curta descrição de cada um desses grupos, juntamente com alguns exemplos de problemas pertencentes a cada um deles.

Capítulo 21

Agropecuária

Embora o Brasil possua uma grande fatia do mercado mundial em agronomia e pecuária, essa atividade econômica poderia ser mais bem explorada. A qualidade, a quantidade e a lucratividade do que é produzido no país e a minimização do impacto ambiental associado podem se beneficiar da incorporação de novas tecnologias computacionais que permitam uma análise inteligente de dados.

Dentre os problemas que podem ser tratados por técnicas de AM, podem ser citados: definição de logísticas de armazenamento e transporte, redução de perdas e desperdícios, manejo racional do cultivo e do solo, combinação de diferentes culturas; barateamento e expansão da produção agrícola, sem prejuízos ao meio ambiente, recuperação de áreas desmatadas e/ou degradadas; previsão e controle de pragas e epidemias, monitoramento em tempo real de animais e de safras, agricultura de alta precisão, previsão do preço de *commodities* e melhoramento genético. Em vários desses problemas, dados são continuamente gerados, sendo, por isso, candidatos naturais à aplicação de técnicas de análise de fluxos contínuos de dados.

Outra área em que o uso de algoritmos de AM pode trazer grandes benefícios é a de agricultura de precisão. A agricultura de precisão busca reduzir os custos de produção e a contaminação do solo com o auxílio da tecnologia da informação.

Tarefas de classificação, regressão, agrupamento de dados e mineração de padrões frequentes têm sido utilizadas em vários dos problemas mencionados. Algumas das aplicações reportadas na literatura são descritas brevemente a seguir.

Uma das primeiras aplicações de AM na pecuária foi a estimativa de gordura intramuscular em carne bovina por RNAs (Mccauley et al., 1994). Nesse trabalho, foram utilizadas imagens de carnes obtidas por ultrassom. De cada imagem, foram extraídas 14 medidas relacionadas à textura da carne, baseadas em descritores de Haralick (Haralick et al., 1973). Desses medidas, uma foi eliminada, e cada uma das demais foi representada por dois valores: média e variação. Foram utilizadas imagens de 159 bois de diferentes cruzamentos. Foram realizados experimentos para previsão tanto do valor da taxa de gordura quanto de classificação das carnes em três intervalos de taxa de gordura. A RNA utilizada nos experimentos foi a rede ALN (do inglês *Adaptive Logic Networks*) (Armstrong et al., 1991).

Outro trabalho que também utilizou AM para avaliar qualidade de carne, mas agora de carne de carneiro, foi publicado em Cortez et al. (2006). Nesse artigo, são apresentados resultados comparativos entre SVMs, RNAs e regressão múltipla em duas tarefas de regressão para avaliar a maciez de carnes de carneiro. O conjunto de dados utilizados é formado por 81 animais. Em uma primeira fase, 12 medidas, como sexo do animal, origem do animal, pH da carne, temperatura da carcaça, índice de cor vermelha da carne, foram

extraídas de cada animal para serem utilizadas como atributos de entrada. Uma técnica de seleção de atributos baseada em análise de sensibilidade foi utilizada para reduzir o número de atributos de entrada. Dois valores foram coletados para atributo de saída, o sabor da carne definido por um painel de 12 especialistas e a força de cisalhamento Warner-Bratzler, que é uma medida mecânica mais utilizada para avaliar a maciez de uma carne. Cada um desses rótulos foi utilizado em um problema de regressão. Os dados apresentaram valores ausentes para os rótulos de alguns animais. Esses animais foram descartados para gerar dois outros conjuntos de dados.

Na área de agricultura, uma aplicação frequente de algoritmos de AM é na avaliação da qualidade de vinhos. Em Policastro et al. (2007), um sistema de RBC com adaptação automática de casos é utilizado para classificação de vinhos. Foi utilizada uma base de casos contendo 289 desses casos. A parte de descrição de cada caso é formada por 9 medidas obtidas por sensores de paladar em uma língua eletrônica. A solução de cada caso é formada por 3 valores referentes a pH, índice de absorção e escore de sabor. Uma combinação de classificadores utilizando RNAs do tipo MLP, SVMs e o algoritmo de indução de árvores de regressão M5 foi utilizada para a adaptação dos casos.

Em vez de uma língua, um nariz eletrônico é utilizado em Masulli et al. (2002) para classificação de cafés. Nesse trabalho, o nariz é utilizado para distinguir entre duas classes: um único tipo de café e uma mistura de diferentes tipos de café. O conjunto de dados possui 187 exemplos de cafés com combinação de grãos de diferentes tipos (originados de 7 misturas) e 225 exemplos de cafés de um único tipo (para 6 tipos diferentes e o café expresso italiano). O nariz eletrônico possui 6 sensores, gerando assim 6 atributos de entrada para cada exemplo. Técnicas de processamento de sinais foram aplicadas aos sinais obtidos dos sensores, para melhorar a qualidade dos dados. Nos experimentos realizados, um algoritmo de *Boosting* cujos classificadores fracos são RNAs do tipo MLP foi comparado a RNAs do tipo MLP utilizadas individualmente.

Capítulo 22

Bioinformática

A Bioinformática é uma área de pesquisa que vem despertando crescente interesse na comunidade científica. Esse campo interdisciplinar consiste na aplicação de técnicas matemáticas, estatísticas e computacionais para a resolução de problemas provenientes da Biologia (Setubal e Meidanis, 1997). Entre as diversas áreas da Biologia, aquela em que a aplicação de técnicas computacionais tem se mostrado mais frutífera é a Biologia Molecular. A Biologia Molecular retrata o estudo das células e moléculas, blocos básicos utilizados na construção de todas as formas de vida (Casley, 1992). Em particular, estuda-se o genoma dos organismos, definido como o conjunto de informações genéticas dos mesmos.

Os estudos em genética possibilitam ganhos em várias áreas, entre elas Medicina, Farmaçia e Agricultura. Pode-se, por exemplo, obter uma melhor compreensão da origem de diversas doenças, o que permite a formulação de tratamentos médicos mais eficazes, assim como o desenvolvimento de novos medicamentos. Como existem alimentos atualmente provenientes de organismos geneticamente modificados, as investigações em Bioinformática também permitem um maior controle e melhoria no desenvolvimento desses novos organismos.

Atualmente, as pesquisas em Bioinformática estão se concentrando na análise dos grandes volumes de dados gerados em laboratórios, visando tratar problemas como, por exemplo, a identificação de genes¹ relacionados a determinadas doenças, como câncer (Wang et al., 2005; Song e Rajasekaran, 2010). Com o objetivo de obter procedimentos automáticos para tal, recursos e técnicas computacionais vêm sendo largamente empregados na análise desses dados.

O uso de técnicas de AM mostra-se uma alternativa promissora, já que elas são capazes de lidar com grandes quantidades de dados, mesmo na presença de ruídos (muito comuns nas bases de dados de Biologia Molecular) e na ausência de teorias gerais (Baldi e Brunak, 1998).

Entre os problemas de Bioinformática em que técnicas de AM vêm sendo aplicadas com sucesso estão:

- **Análise da forma de proteínas:** as proteínas são moléculas presentes no meio celular que possuem diversas funções biológicas, entre elas estruturais e regulatórias. As funções e propriedades de uma proteína são determinadas por sua estrutura tridimensional. Uma área de pesquisa que tem despertado grande interesse é a análise dos dados genéticos sequenciados com o fim de determinar a estrutura das proteínas

¹O gene é a estrutura fundamental determinante da hereditariedade, codificando as proteínas, componentes essenciais de todo ser vivo.

codificadas. Entre os algoritmos de AM empregados para esse fim, destacam-se as redes neurais artificiais (Bohr, 1999) e os algoritmos genéticos (Unger, 2004), entre outros (Cheng et al., 2008).

- **Localização de proteínas no meio celular:** nesse caso, o objetivo está em, a partir da sequência de aminoácidos de uma proteína (ou atributos extraídos delas), prever sua localização na célula (membrana, núcleo etc.). O local em que uma proteína atua está fortemente relacionado à sua função. Esse problema pode ser modelado como uma classificação multiclass ou multirrotulo, pois diversas proteínas atuam em locais distintos no meio celular. Entre as técnicas de AM mais utilizadas na literatura correlata ao problema em questão encontram-se as máquinas de vetores de suporte (Hua e Sun, 2001; Garg et al., 2005; Lorena e Carvalho, 2007).
- **Predição da função de proteínas:** o interesse nesse caso está na previsão da função de proteínas a partir de suas sequências de aminoácidos (ou de atributos extraídos delas). Muitas vezes os tipos de funções se relacionam de maneira hierárquica, configurando um problema de classificação hierárquico. Adicionalmente, uma mesma proteína pode ter mais de uma função associada a ela, o que pode tornar o problema de classificação, além de hierárquico, multirrotulo. Diversas técnicas de AM supervisionado vêm sendo empregadas na solução desse problema, entre as quais máquinas de vetores de suporte, redes neurais artificiais, *naive Bayes*, *k*-vizinhos mais próximos (Freitas e de Carvalho, 2007; Costa et al., 2008; Cerri et al., 2009; Cerri e Carvalho, 2010).
- **Alinhamento de sequências:** o objetivo nesse caso é alinhar sequências relacionadas, dada uma função de custo. Com isso permite-se explorar o grau de similaridade entre cadeias de DNA², mRNA³ ou proteínas. Um grau de similaridade elevado indica uma alta probabilidade de as funções executadas pelas moléculas comparadas serem semelhantes. Pode-se então inferir, por exemplo, funções de novas proteínas até então desconhecidas, ou identificar possíveis mutações genéticas. O alinhamento pode ser utilizado também para medir a distância evolutiva entre duas ou mais espécies (Shamir et al., 2001). Algoritmos genéticos têm sido largamente utilizados no alinhamento de sequências (Notredame e Higgins, 1996; Gondro e Kinghorn, 2007).
- **Identificação de genes em sequências de DNA:** compreende o reconhecimento e a localização de cada gene em sequências de DNA. São normalmente utilizadas duas abordagens na realização dessa tarefa (Craven e Shavlik, 1994). A primeira envolve determinar sinais que podem ser identificados nas sequências e que indicam situações especialmente adequadas à localização de genes. A segunda abordagem, denominada busca por conteúdo, procura por padrões nas sequências que determinem regiões do DNA que são codificadas em proteínas. Em Bandyopadhyay et al. (2008) são discutidos trabalhos que usam técnicas como árvores de decisão, raciocínio baseado em casos, redes neurais artificiais e algoritmos genéticos no problema de identificação de genes.

²Os genes dos organismos encontram-se codificados em moléculas de DNA (ácido desoxirribonucleico).

³O mRNA (ácido ribonucleico mensageiro) é uma molécula intermediária formada durante a síntese de proteínas.

- **Análise de expressão gênica:** a expressão gênica é o processo pelo qual as sequências dos genes são interpretadas na produção de proteínas. O objetivo na análise desses dados é verificar padrões no nível de expressão de diferentes genes (a “quantidade” de proteínas que eles produzem). Com isso pode-se, por exemplo, comparar mudanças na expressão de alguns genes perante alguma doença, possibilitando assim identificar alvos para futuros medicamentos e/ou terapias gênicas. Técnicas de AM supervisionado e não supervisionado diversas vêm sendo aplicadas na análise de dados de expressão gênica (Souto et al., 2003; de Souza et al., 2010).
- **Previsão de interação entre proteínas:** a maioria das proteínas desempenha seu papel nos organismos por meio de interações com outras moléculas, frequentemente outras proteínas (Shoemaker e Panchenko, 2007a,b). As interações entre proteínas, conhecidas como interações proteína-proteína, permitem que cientistas possam prever a sequência de reações e processos que ocorrem nas células. Isso ajuda a entender a função das proteínas e o desenvolvimento de novos medicamentos. O conhecimento das interações em um grupo de proteínas permite ainda a construção de redes de interações. Essas redes ajudam a entender os princípios gerais seguidos por sistemas biológicos. Técnicas de AM, como *naive Bayes*, máquinas de vetores de suporte e modelos múltiplos preditivos, têm sido utilizadas com sucesso para a previsão de interação entre proteínas (Qi et al., 2005; Yu et al., 2010).

A lista anterior apresenta alguns exemplos de aplicações de AM em Bioinformática que vêm sendo exploradas, porém ela não é exaustiva. Ainda há muitos resultados a serem obtidos e métodos a serem investigados, o que faz o uso de AM em Bioinformática um campo de pesquisa promissor.

Capítulo 23

Ecologia e Meio Ambiente

É crescente a preocupação com a manutenção da qualidade do meio ambiente no planeta. Problemas associados a desastres ambientais, como desmatamento de florestas, queimadas, extinção de espécies, inundações, poluição de rios, lagos e represas, aparecem com frequência no noticiário. Sistemas computacionais baseados em AM fazem parte das soluções mais recentes para lidar de forma preventiva com esses problemas.

Esses sistemas já vêm sendo utilizados como ferramentas auxiliares na geração de modelos que podem ser utilizados em variados tipos de análises, tais como na indicação de áreas para conservação ambiental (Egbert et al., 1999; Ortega-Huerta e Peterson, 2004), na avaliação do risco de proliferação de espécies invasoras (Higgins et al., 1999; Peterson e Vieglais, 2001; Peterson et al., 2003; Peterson, 2003; Thuiller et al., 2005; Williams et al., 2008), no estudo do impacto de mudanças ambientais na biodiversidade (Huntley et al., 1995; Pearson et al., 2002; Peterson et al., 2002a; Berry, 2002; Hannah et al., 2005, 2007; Sinclair et al., 2010), na indicação de rotas de disseminação de doenças (Peterson et al., 2002b, 2006, 2007), no suporte à conservação e seleção de reservas (Araújo et al., 2000; Ferrier, 2002; Loiselle, 2003; Ortega-Huerta e Peterson, 2004; Leathwick, 2005), no teste de teorias ecológicas (Peterson et al., 1999; Anderson et al., 2002; Hugall, 2002; Graham et al., 2006), na reintrodução de espécies em perigo de extinção (Pearce e Lindenmayer, 1998; Martínez-Meyer et al., 2006), entre outros.

O Brasil possui uma vasta biodiversidade a ser explorada por meio do uso de ferramentas de modelagem, como meio para o monitoramento, a preservação ambiental e o desenvolvimento de estratégias que auxiliem o uso sustentável dos recursos naturais.

Esta seção vai apresentar alguns trabalhos reais que ilustram como algoritmos de AM podem ser utilizados em alguns dos problemas mencionados. Os trabalhos discutidos cobrem três temas, a saber:

- Modelagem da distribuição potencial de espécies;
- Monitoramento da qualidade da água;
- Prevenção de incêndios florestais.

Incêndios em florestas, acidentais ou provocados por ação humana, levam a prejuízos ecológicos e financeiros, muitas vezes provocando a perda de vidas tanto de pessoas que moram nas proximidades quanto daquelas que trabalham para controlar e apagar o fogo. Quanto mais rápida for a detecção de incêndios florestais, maior a chance de controle, o que leva a uma redução dos danos causados.

Em Cortez e Moraes (2007), os autores mostram como algoritmos de AM podem ser utilizados para a prevenção de incêndios florestais. Nesse trabalho, cinco algoritmos de AM tiveram sua acurácia preditiva comparada: algoritmo de indução de árvores de decisão, florestas aleatórias, máquinas de vetores de suporte, modelo de regressão múltipla e redes neurais. Os dados originais tinham atributos relacionados a coordenadas da área de incêndio, mês do ano, dia da semana, atributos relacionados a condições atmosféricas, temperatura, umidade relativa do ar, velocidade do vento e precipitação atmosférica. Técnicas de transformação converteram atributos qualitativos em quantitativos, e técnicas de seleção foram utilizadas para selecionar os atributos mais relevantes. Os melhores resultados foram obtidos por máquinas de vetores de suporte com quatro atributos, dados meteorológicos relacionados a temperatura, umidade relativa do ar, precipitação pluviométrica e velocidade do vento.

O impacto do despejo de água sem tratamento por indústrias em rios, lagos, reservatórios e mares é um problema conhecido. Mais recentemente, têm sido denunciados os efeitos causados por mudanças no nível de nutrientes e composição química, com graves danos à fauna e à flora, causados pela contaminação por adubos utilizados na agricultura e lançamento de esgoto sem o tratamento adequado. Dentre os efeitos, alguns deles indiretos, podem ser citados problemas de saúde em seres humanos. Em Policastro et al. (2004), os autores utilizam um sistema híbrido de raciocínio baseado em casos para o monitoramento da qualidade da água em rios. No sistema híbrido proposto, três algoritmos de AM, algoritmo de indução de árvore de decisão, máquina de vetores de suporte e redes neurais artificiais, foram utilizados em comitês para adaptar os casos recuperados da base de casos. Nos experimentos realizados, utilizou-se como atributos de entrada a quantidade de substâncias químicas, como nitrogênio, fosfato, pH, oxigênio e cloreto, junto com distribuição de populações de algas.

A modelagem da distribuição potencial de animais e vegetais tem sido uma ferramenta valiosa na obtenção de informações biogeográficas a respeito de diferentes espécies, as quais podem ser aplicadas em vários campos, entre eles estudos ecológicos (Pearson, 2007). Por meio do uso desses modelos, é possível obter uma melhor compreensão da influência e da relação de condições ambientais com a distribuição das diferentes espécies animais e vegetais.

Os modelos para predizer a distribuição potencial de espécies são construídos a partir da combinação de dois tipos de dados: (a) sobre a ocorrência/localização da espécie; e (b) ambientais. Os dados de ocorrência são coordenadas geográficas, normalmente pares de latitude e longitude em algum sistema de referência em que a espécie foi observada ou coletada. Os dados ambientais representam variáveis que reconhecidamente influenciam a distribuição da espécie. O primeiro passo no processo de modelagem consiste então em encontrar a correspondência entre as condições ambientais e cada ponto de ocorrência. Para o caso dos algoritmos de AM, obtém-se nesse passo um conjunto de dados em um formato atributo-valor, em que os atributos representam as condições ambientais e as classes constituem a presença ou a ausência da espécie. Tem-se assim um problema de classificação, consistindo em predizer a presença ou ausência da espécie em estudo.

Em geral, os dados de ocorrência são desbalanceados em termos da proporção de exemplos de presença e ausência (Elith et al., 2006). Isso ocorre porque é mais fácil e usual registrar a presença de espécies em campo, resultando em poucos dados de ausência. Esse fator representa um problema para o uso usual das técnicas de AM, em que o cenário ideal é de mesma quantidade de dados em ambas as classes. Uma estratégia

frequentemente empregada para lidar com essa limitação é gerar pontos de “pseudoausência” (Stockwell e Peters, 1999) ou gerar modelos a partir somente dos dados de presença (*one-class classification*) (Elith et al., 2006).

A partir do conjunto de dados montado, o algoritmo de AM, empregando o paradigma de aprendizado supervisionado, deve identificar os padrões de condições ambientais adequadas à espécie e que garantam a sua sobrevivência. O uso desse modelo em novas regiões permite extrapolar suas previsões a novos ambientes. Normalmente, é requerido que a técnica retorne uma probabilidade de ocorrência da espécie de acordo com as características ambientais fornecidas como entrada. O campo de AM abre diversas possibilidades de interação com a modelagem de distribuição de espécies e vem sendo cada vez mais utilizado nesse processo (Lorena et al., 2010).

Capítulo 24

Energia

Outro campo em que técnicas de AM são frequentemente utilizadas é o de energia. Com o esgotamento de fontes de energia não renováveis e a preocupação com os danos ambientais relacionados a algumas fontes, algoritmos de AM têm possibilitado uma utilização mais racional dos recursos disponíveis, tanto do ponto de vista financeiro como ambiental, e um planejamento que permita um melhor uso dos recursos futuros. Algoritmos de AM têm sido utilizados em diversas aplicações, relacionadas tanto a aspectos operacionais, como distribuição, exploração, geração de recursos energéticos, como a aspectos relacionados a planejamento e comercialização de energia. Nessas aplicações, os algoritmos têm sido utilizados principalmente em ferramentas de otimização e de suporte à tomada de decisão. Dentre as principais aplicações, podem ser listadas:

- previsão de carga, de preço;
- planejamento reativo, de expansão de sistemas de distribuição, de redistribuição de alimentadores, de agendamento de geradores, de minimização de perdas;
- controle da operação;
- proteção de sistemas de energia;
- composição de fontes energéticas;
- previsão de rompimento de dutos.

Nas áreas de exploração e geração de energia, podem ser facilmente encontrados na literatura trabalhos relacionados a diferentes fontes de energia tanto não renováveis, como petróleo, gás, nuclear, carvão, como renováveis, como hidrelétrica, solar, eólica e bioengenharia. A maioria dos trabalhos diz respeito a aplicações no sistema elétrico de potência. Esse sistema é formado pelos sistemas de geração, de transmissão de distribuição e de subestações. A transmissão de energia ocorre por linhas de transmissão. Sistemas de transmissão de energia levam energia de estação de geração para sistemas de distribuição, que por sua vez enviam a energia para as subestações, que alimentam, por exemplo, bairros de uma cidade. Técnicas de AM têm sido utilizadas em problemas associados a esses sistemas.

Uma das principais aplicações na área de transmissão de energia é a detecção e localização de falhas em linhas de transmissão de energia (Oleskovicz et al., 1998). A maioria das falhas que ocorrem no sistema elétrico acontece nessas linhas. As causas dessas falhas podem ser várias, como, por exemplo, problemas de isolamento, queimadas próximas

às linhas e contatos entre os cabos. Técnicas de regressão podem ser utilizadas para a previsão e a localização de falhas ou faltas em redes de distribuição. Por exemplo, uma rede neural artificial pode ser treinada para prever tanto quando uma falta está prestes a ocorrer quanto em que posição da linha de transmissão ocorreu uma falta. Para isso, a entrada da rede neural pode ser um vetor com os valores de tensão e de corrente medidos antes e após a falta.

Um sistema de distribuição pode ser modelado por um grafo em que subestações fornecem energia para diferentes setores do sistema, ilustrado pela Figura 24.1 (Delbem et al., 2003). Nesse grafo, as arestas representam linhas para a distribuição de energia, e os nós representam chaves. Um nó preto representa uma chave fechada, permitindo a passagem de energia entre os setores anterior e posterior à chave. Um nó branco representa uma chave aberta.

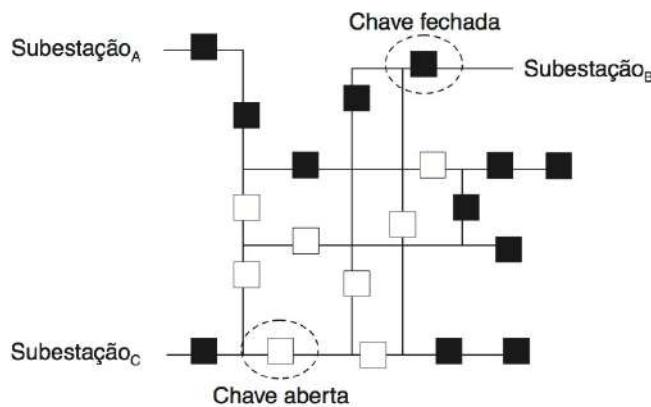


Figura 24.1 Sistema de distribuição de energia.

Falhas em um sistema de distribuição de energia podem fazer com que setores deixem de receber energia, ficando desenergizados. Uma vez que a falha tenha ocorrido, planos de restabelecimento de energia elétrica precisam ser elaborados rapidamente, de forma a realimentar os setores desenergizados. Técnicas de AM podem também ser utilizadas para restabelecimento de energia em sistemas de distribuição. Para isso, a falta deve ser inicialmente identificada e isolada. Em seguida, a técnica de AM gera um plano para o restabelecimento de energia. O uso de AM permite que o restabelecimento seja realizado rapidamente. Uma ferramenta baseada em algoritmos genéticos foi desenvolvida para definir que chaves devem ser abertas e fechadas para gerar um plano ótimo (Delbem et al., 2003, 2005). Um plano ótimo pode minimizar o número de regiões desenergizadas, reduzir sobrecarga de equipamentos, evitar que as chaves alteradas estejam distantes entre si e equilibrar a carga entre as subestações. Em uma aplicação semelhante, AM pode ser utilizada para a redefinição das fontes de fornecimento de energia para que seja realizada a manutenção de partes do sistema de distribuição.

A área de planejamento também oferece várias oportunidades para a utilização de técnicas de AM. Assim como AM pode ser utilizada na composição de carteiras ou fundos de investimento, definindo o quanto utilizar de um dado ativo, elas podem ser utilizadas para definir quanto de cada fonte de energia será utilizado para alimentar uma demanda futura. Fatores ambientais, políticos e econômicos fazem com que diferentes fontes de geração de energia tenham custos diferentes ao longo do tempo. Por exemplo, os recursos hidrelétricos de geração de energia têm sua disponibilidade afetada pelo volume

de água armazenado nos reservatórios de usinas hidrelétricas. Em períodos de estiagem, geralmente ocorre uma baixa nesses volumes.

Em Leite et al. (2002), algoritmos genéticos foram utilizados em uma ferramenta para o planejamento de sistemas hidrotérmicos formados por usinas hidrelétricas e usinas térmicas. No sistema utilizado, que reflete o sistema brasileiro, as usinas hidrelétricas estão situadas em uma mesma bacia hidrográfica, com os reservatórios acoplados. Isso forma um sistema em cascata em que o volume de um reservatório é influenciado pelo volume do reservatório anterior. Essa dependência não ocorre em um sistema de geração térmica, em que a capacidade de geração de uma usina não depende da geração da outra, mas depende da cotação do combustível utilizado. Na ferramenta apresentada, o algoritmo genético otimiza os volumes dos reservatórios de forma a reduzir o uso de energia térmica, que tem um custo financeiro maior. O AG busca a solução ótima, que procura equilibrar o uso atual da água para geração hidrelétrica e seu uso futuro decorrente de um maior armazenamento.

Outra aplicação associada ao planejamento é a previsão do preço da energia proveniente de diferentes fontes e da energia necessária para um dado período futuro. Essas aplicações têm sido abordadas tanto como séries temporais quanto como fluxos contínuos de dados (Rodrigues e Gama, 2009).

Com relação à exploração de gás e de petróleo, algoritmos de AM podem ser utilizados para a detecção de vazamento, problema que tem causado vários desastres ambientais, estimativa de propriedades de reservas, previsão de viscosidade, previsão de rompimento de dutos de escoamento e previsão do valor do barril de petróleo ou metro cúbico de gás natural.

Capítulo 25

Finanças

A demanda por sistemas financeiros mais eficientes e confiáveis tem levado a um crescente aprimoramento dos sistemas de análise de dados utilizados pelos agentes financeiros. Essa sofisticação tem levado ao desenvolvimento de sistemas computacionais que sejam capazes de analisar grandes quantidades de dados rapidamente e de gerar relatórios capazes de subsidiar a tomada de decisões. A maioria das principais instituições financeiras, tanto internacionais quanto nacionais, está utilizando algoritmos de AM para a análise de seus dados. Em várias aplicações na área de finanças, o uso de AM tem possibilitado ganhos financeiros expressivos. Entre os principais problemas em que AM tem sido utilizada com sucesso, podem ser citados:

- análise de risco de crédito;
- previsão de falências;
- previsão de cotações de moedas e de ações;
- detecção de fraudes;
- segmentação de mercados;
- composição de carteiras de investimento.

Algoritmos baseados em diferentes paradigmas de AM têm sido utilizados nesses problemas. A seguir, alguns desses problemas serão descritos em mais detalhes, junto com as principais abordagens de AM utilizadas para lidar com eles.

Análise de risco de crédito é possivelmente o problema financeiro em que AM tem sido mais popular. Esse problema pode estar associado tanto a pessoa jurídica, por exemplo, empresas, como a pessoa física. A análise de risco de crédito para pessoa física é geralmente utilizada quando uma pessoa solicita um cartão de crédito, cheque especial, financiamento ou crediário. Nesse caso, a instituição que faz a análise de crédito geralmente utiliza um conjunto de dados de aplicações passadas, que possuem, para cada aplicação, o perfil do cliente dessa aplicação e sua classificação. A classificação pode ser um escore dado por um analista de crédito, que pode informar se deve ou não ser dado o crédito, como uma análise do histórico financeiro do cliente. O histórico financeiro pode conter dados como quantas vezes o cliente se tornou inadimplente. Observe que o primeiro caso, o rotulamento dos dados, é subjetivo. Diferentes analistas de crédito podem dar escores diferentes à aplicação.

Quando AM é utilizada nesses problemas, um algoritmo de AM para classificação de dados utiliza esse conjunto de dados para induzir um modelo ou hipótese, que será depois utilizado para os novos clientes. Embora esse seja geralmente um problema de classificação binária, ele pode apresentar mais de duas classes, que podem inclusive formar um *ranking* de classes, para diferentes perfis de clientes. Uma dificuldade geralmente encontrada é que os conjuntos de dados em geral têm muito mais dados de clientes que foram aprovados (ou têm um bom histórico financeiro) que o contrário. Para lidar com essa limitação, técnicas de pré-processamento para dados desbalanceados podem ser utilizadas. Observe que, nessas aplicações, os erros podem ter custos diferentes. Pode ser menos custoso rejeitar o crédito a um cliente que não ficaria inadimplente que o contrário. Algoritmos de agrupamento de dados também podem ser aplicados a esses dados para encontrar diferentes perfis de clientes, que seriam os grupos encontrados pelo algoritmo. Diferentes produtos poderiam ser oferecidos a diferentes grupos de clientes.

Na detecção de fraudes, o objetivo é detectar transações que fogem do normal. Fraudes ocorrem não apenas em transações financeiras, mas também no uso de energia, na compra de suprimentos, na utilização de recursos sociais, no acesso a redes de computadores, para citar alguns casos. Algoritmos de AM são geralmente utilizados para prever ou classificar quando uma dada transação é uma fraude, um problema de classificação binária. Assim como os dados de análise de crédito, os conjuntos de dados de detecção de fraudes apresentam (felizmente) muito mais dados de transações corretas do que dados que representam fraudes. Outra dificuldade é que novos tipos de fraude estão constantemente sendo criados, o que torna necessária a adaptação contínua dos modelos. A detecção de fraudes pode ser vista como um problema de fluxos contínuos de dados.

Enquanto dados de análise de crédito para experimentos com algoritmos de AM estão disponíveis em vários repositórios públicos, o mesmo não ocorre com dados de crédito. A razão é simples: quem tem esses dados não quer liberá-los para que suas vulnerabilidades passadas (e talvez presentes) não sejam conhecidas.

A previsão de cotações de valores de moedas ou ações é geralmente realizada por algoritmos que podem gerar um valor real como valor de saída. Os dados utilizados nesses casos são em geral séries temporais, que são conjuntos de dados gerados em intervalos predefinidos. A cada intervalo, o mesmo número de valores é gerado. Por exemplo, uma série temporal de cotações de moedas pode ser formada pelo valor da moeda de N países, tomado diariamente, nos últimos M anos.

A composição de carteiras ou fundos de investimentos procura definir quanto de um dado recurso total deve ser alocado a cada ativo de um conjunto de possíveis ativos, que podem incluir ações negociadas na bolsa de valores, títulos da dívida pública, títulos cambiais, derivativos ou *commodities* negociados em bolsas de mercadorias e futuros, entre outros. Na composição de carteiras de investimentos, técnicas baseadas em busca, que incluem as técnicas bioinspiradas, podem ser utilizadas para definir o peso ou grau de participação de cada ativo na carteira. Por exemplo, se um algoritmo genético for utilizado, cada gene pode armazenar o peso de um dado ativo. Durante o processo evolutivo, o AG buscaria o conjunto de pesos que provavelmente dará o maior retorno financeiro em um dado período.

Capítulo 26

Mineração de Dados e Textos

A mineração de dados (DM) consiste em extrair ou “minerar” conhecimento a partir de grandes quantidades de dados. Em parte da literatura relacionada, a MD é também tratada como sinônimo para outro termo, a descoberta de conhecimento em bases de dados (KDD, do inglês *Knowledge Discovery in Databases*). Outros autores consideram a mineração de dados uma etapa no processo de KDD, o qual compreende as seguintes etapas: a seleção, limpeza e integração dos dados, a transformação dos dados, a mineração dos dados e a avaliação e apresentação dos resultados (Fayyad et al., 1996).

É importante destacar que a mineração de dados busca extrair conhecimento inovador, ou seja, algo anteriormente não conhecido e que tenha valor para o domínio em que é aplicada. As técnicas de AM estão entre as mais empregadas no processo de MD, e diversas aplicações de sucesso de AM em MD são continuamente reportadas. Um dos usos mais clássicos de MD, no caso utilizando regras de associação, é a análise de bases de dados de estabelecimentos comerciais, tais como supermercados. Os modelos gerados a partir desses dados podem ser empregados no auxílio à tomada de decisões gerenciais, como a definição de que produtos podem ser comercializados em conjunto para aumentar as vendas. Muitas das aplicações das seções anteriores podem ser caracterizadas como de MD, tais como a análise de crédito, a geração de modelos para suporte ao diagnóstico médico, entre outras.

É importante observar que a MD lida com dados de observações já realizadas, não com dados gerados propositadamente para serem no futuro analisados por técnicas de MD. Os dados utilizados são armazenados para um propósito diferente de sua análise por técnicas de MD. Como exemplo pode ser citado um banco de dados com cadastros de pacientes de um hospital. Esses dados não são coletados com o propósito de serem posteriormente analisados por técnicas de MD, mas sim para que profissionais da área médica possam acessar com facilidade os dados de um paciente.

Existem várias ferramentas computacionais, tanto comerciais quanto gratuitas, que contêm várias técnicas de MD, incluindo técnicas para pré-processamento de dados, algoritmos de AM e técnicas para pós-processamento. Algumas dessas ferramentas possuem interface gráfica, que pode ser utilizada para selecionar e conectar módulos. Outras são utilizadas por meio de um ambiente de linhas de comando. Outras ainda integram essas duas alternativas.

Quando os dados que estão sendo analisados se apresentam na forma de textos, é usual se referir à sua análise como mineração de textos (MT). As técnicas convencionais de AM são usualmente empregadas sobre conjuntos de dados no formato atributo-valor, que é considerada uma forma estruturada de representar dados. Como dados em textos são apresentados de uma forma não estruturada, eles precisam ser convertidos para o formato

atributo-valor antes de serem utilizados. Assim, os textos precisam ser pré-processados para que sejam convertidos em formas estruturadas de dados. De posse do conjunto de dados no formato atributo-valor, algoritmos convencionais de AM podem ser utilizados para extrair informações relevantes de textos.

Pela forma como os dados são representados, a área de MT possui forte ligação com as áreas recuperação de informação (RI) e processamento de língua natural (PLN). A RI procura identificar, em um grande conjunto de textos, aqueles que são relevantes para as necessidades do usuário (Shatkay e Feldman, 2003). As técnicas de RI podem ser classificadas:

- De acordo com a forma de consulta: baseada em documento ou baseada em consulta (Yandell e Majoros, 2002);
- De acordo com a forma de análise: estatística ou semântica (Greengrass, 2001).

RI baseada em consulta faz uma busca por documentos que atendam a um conjunto de condições fornecido pelo usuário. RI baseada em documento compara documentos para, por exemplo, a obtenção de documentos semelhantes a um dado documento. Com relação à forma de análise, em geral é utilizada a RI estatística, que gera uma análise estatística da presença de palavras em textos. A RI semântica, por outro lado, produz uma análise sintática e semântica de textos, dando importância à função de cada palavra em uma frase, e não apenas à sua presença.

As técnicas que seguem a abordagem estatística normalmente extraem palavras isoladas ou palavras compostas do texto analisado. Essas palavras são identificadas pela frequência de co-ocorrência ou pela presença em dicionários, ontologias etc. Geralmente, para eliminar variações de uma mesma palavra, cada palavra é associada à sua raiz (*stem*). Em seguida, cada raiz e a frequência com que ela ocorre no texto podem ser representadas em uma tabela atributo-valor. Raízes para palavras que ocorrem com frequência elevada em todos os textos, como artigos e termos comuns, denominadas *stop-lists*, são eliminadas. Tanto palavras que ocorrem muito raramente quanto as que ocorrem com muita frequência são em geral eliminadas, pois podem ser pouco representativas ou discriminativas.

Os métodos baseados na abordagem semântica utilizam conhecimento sintático e semântico das palavras presentes nos textos. Eles permitem aprimorar os resultados obtidos com a abordagem estatística. Com isso, é possível melhorar a qualidade do conjunto de atributos e interpretar melhor os resultados do processo de mineração.

A aplicação da MD a dados provenientes da *web*, denominada mineração *web* (*web mining*), também é cada vez mais frequente. As análises desses dados buscam descobrir fontes de informações relevantes, mapear e analisar o padrão de acesso à informação contida na *web*, assim como a forma de armazenamento de informações na *web*. Muitas vezes, as informações presentes encontram-se no formato textual, formando assim uma interface com a MT.

Como exemplos de análises usualmente efetuadas, podem ser mencionadas a extração de conhecimento do conteúdo das páginas e de suas descrições, a obtenção de conhecimento a partir da organização da *web* e da referência cruzada de ligações entre diferentes páginas e a geração de padrões interessantes a partir do uso dos registros de acesso da *web* (Ebecken et al., 2003).

Capítulo 27

Robótica

O termo robótica se refere ao estudo e uso de robôs. Uma caracterização completa desse termo, portanto, envolve uma definição formal do que vem a ser um robô. Existem diversas tentativas nesse sentido. A definição mais usual é a de que um robô é uma máquina que procura reproduzir alguma capacidade física de uma pessoa. Historicamente, a palavra robô foi introduzida pelo dramaturgo tcheco Karel Capek em 1921 em sua peça *R.U.R. (Robôs Universais de Rossum)*, a qual ilustrava a desumanização do homem em uma civilização tecnológica, e tem origem na palavra tcheca que designa trabalho forçado ou escravo. Já o termo robótica foi criado e usado primeiramente pelo cientista russo Isaac Asimov, mais conhecido pelos inúmeros trabalhos de ficção científica que produziu (entre eles, *Eu Robô*, de 1950).

Em geral, os robôs podem ser divididos em duas classes (Turban e Frenzel, 1992). A primeira é constituída por robôs pré-programados na realização de tarefas específicas, tais como os robôs de indústrias automotivas. Esses tipos de robôs podem substituir os humanos em tarefas bem definidas e limitadas. Dessa forma, eles são capazes de realizar somente um conjunto de tarefas repetidamente. Não são capazes, por exemplo, de se adaptar a mudanças que venham a ocorrer em seu ambiente, sem que seja necessária uma reprogramação. No sentido de acomodar esse tipo de requisito, tem-se o desenvolvimento de robôs inteligentes, os quais se utilizam de técnicas de Inteligência Artificial na percepção do ambiente e na realização de suas tarefas. Os robôs inteligentes podem ser definidos, portanto, como mecanismos capazes de extrair informações de seu ambiente e usar conhecimento sobre ele para realizar suas tarefas de forma segura e útil (Arkin, 1998). Para extração de informações do ambiente, os robôs utilizam dispositivos mecânicos especiais. Esses dispositivos, denominados sensores, em geral incluem câmeras e sonares.

Na comunidade científica, os primeiros robôs autônomos (inteligentes) surgiram na década de 1940, sendo eles o *machina* de Grey Walter e a *criatura* de John Hopkins. Importantes e pioneiras contribuições também foram dadas por Nilsson em 1969, com o robô *Shakey*, e por Moravec em 1977, com o *Standford Cart*. Desde os primeiros trabalhos em robótica até a atualidade, houve grande proliferação de pesquisas na área, com avanços que incluem máquinas capazes de navegar em rodovias a grandes velocidades e humanoides com capacidade (e equilíbrio) de subir e descer escadas.

A definição de um robô inteligente deve levar em conta o ambiente em que ele deve operar e as tarefas que ele deve realizar nesse meio. Os ambientes podem ser os mais variados; entre eles podemos citar escritórios, casas, labirintos, indústrias, ruas. Também podem ser aquáticos, aéreos ou mesmo espaciais. Na maioria dos casos, os robôs são projetados de forma a interagir com um meio específico, pois a capacidade de lidar com uma mudança de meio aumenta em grande escala a complexidade do sistema. As tarefas

que um robô deve ser capaz de realizar também podem ser diversas, e estão intimamente ligadas ao meio em que atuam. Algumas tarefas gerais são: navegação, manipulação de objetos, percepção e mapeamento do ambiente, entre outras. Um robô pode estar programado de forma a executar uma tarefa específica ou uma combinação destas no alcance de seus objetivos.

Cada uma das tarefas mencionadas pode ser modelada com o uso de técnicas de AM, permitindo a obtenção de controladores automáticos. De forma geral, as técnicas mais empregadas em aplicações de robótica são aquelas adequadas a tarefas de percepção e controle, tais como redes neurais artificiais e algoritmos genéticos. Em Sobolewski (2009), por exemplo, redes neurais artificiais são usadas no controle de robôs em tarefas de resgate.

Capítulo 28

Saúde

Uma das áreas de aplicação em que técnicas de AM têm tido maior receptividade é a área de saúde. Nessa área, ferramentas computacionais baseadas em AM são frequentemente utilizadas para controle de epidemias, auxílio a exames clínicos, monitoramento do estado de pacientes, dosagem de medicamentos e auxílio para o diagnóstico médico.

As aplicações de AM para auxílio a exames clínicos têm sido realizadas para avaliar sinais gerados por equipamentos médicos, como eletrocardiogramas, eletroencefalogramas, imagens obtidas de tomógrafos computadorizados, ressonância magnética, raios X, mamografias, para citar algumas.

Diferentemente de outras aplicações, em que soluções do tipo caixa-preta são aceitáveis, em várias aplicações na área de saúde, além de acurácia preditiva, o usuário do algoritmo de AM deseja uma explicação de como o algoritmo constrói suas hipóteses a partir dos dados. Além disso, nessas aplicações os erros podem ter custos diferentes. É usualmente pior diagnosticar o paciente como saudável quando ele porta a doença do que diagnosticar uma pessoa saudável como doente, pois nesse último caso novas baterias de exames serão provavelmente realizadas e permitirão identificar que a doença está ausente, enquanto no primeiro o paciente pode ser dispensado do tratamento.

Diabetes é uma das doenças mais comuns nos dias de hoje. Assim como em outras doenças, a falta de um diagnóstico correto, de preferência feito nos estágios iniciais, pode ter graves consequências. Em Patil et al. (2010), os autores descrevem uma abordagem híbrida de AM para diagnóstico de pacientes com diabetes do tipo 2. Na abordagem proposta, foi combinado o algoritmo de agrupamento de dados *k*-médias com o algoritmo C4.5 de indução de árvores de decisão. O algoritmo *k*-médias foi empregado para validar as classes associadas aos exemplos, de forma a remover os objetos, no caso pacientes, estimados como incorretamente rotulados. O algoritmo C4.5 foi então aplicado aos novos dados.

Nos experimentos foi utilizado o conjunto de dados de diabetes dos índios PIMA, do repositório UCI (Frank e Asuncion, 2010). Vários dos conjuntos de dados disponíveis no repositório da UCI são relacionados a problemas médicos. O conjunto PIMA possui dados de pacientes indígenas do sexo feminino com atributos de entrada como número de gestações, pressão sanguínea, idade, massa corporal. Cada objeto desse conjunto tem uma de duas possíveis classes: a paciente mostra sinais de diabetes e a paciente não mostra esses sinais. Objetos com valores ausentes foram eliminados, e os atributos de entrada foram transformados para que apresentassem média igual a 0 e variância igual a 1.

Ambulatórios médicos e setores de urgência de hospitais raramente contam com computadores portáteis para que médicos possam entrar com informações referentes às respostas dadas às perguntas feitas a pacientes e sintomas detectados. Por isso, muitas vezes es-

sas informações são preenchidas manualmente em formulários específicos. A pressa no preenchimento de formulários pode levar a dificuldade de sua leitura e correta interpretação. Para minimizar esses problemas, em de Faveri Honorato et al. (2008) e Zalewski et al. (2008) é proposto e avaliado um sistema computacional baseado em AM para reconhecimento de escrita em formulários médicos.

Em Zalewski et al. (2008), os autores se dedicam ao reconhecimento de dígitos escritos à mão e identificação do itens do formulário selecionados quando o médico tinha mais de uma opção de escolha. Esses dois problemas foram tratados de forma independente. Com respeito aos dígitos manuscritos, os dados são inicialmente pré-processados para reduzir os problemas que podem dificultar o reconhecimento. Após o pré-processamento, são extraídas características representativas dos dados, que são então utilizadas por algoritmos de AM. O conjunto de dados utilizado consiste em 250 questões de múltipla escolha e 504 dígitos. Como algoritmos de AM foram utilizados o *k*-vizinhos mais próximos, máquinas de vetores de suporte e redes neurais artificiais.

Outro problema em que algoritmos de AM têm sido utilizados é a classificação de imagens oftalmológicas. Muitas pessoas apresentam problemas de vista provocados por erros refrativos, que se manifestam por meio de problemas oculares como miopia, hipermetropia e astigmatismo. Os dispositivos comumente empregados para diagnóstico desses erros precisam ser frequentemente calibrados e possuem uma manutenção custosa e suporte técnico especializado. Em Libralao et al. (2005), uma combinação de classificadores induzidos por diferentes algoritmos de AM é utilizada para estimar erros refrativos por meio da classificação de imagens oftalmológicas. O sistema desenvolvido utiliza imagens geradas pela técnica *Hartmann-Shack* (Thibos e Hong, 1999).

O trabalho foi realizado com um conjunto de dados formado por seis variações da imagem dos olhos de 100 pacientes. Antes do uso dos algoritmos de AM, as imagens foram primeiro pré-processadas por meio do uso de filtros e técnicas de eliminação de ruídos e passaram por um processo de extração de características que utilizou transformadas *wavelet*. Assim, cada imagem foi representada por um vetor de características. Em seguida, experimentos para a classificação das imagens oculares pré-processadas foram realizados utilizando diferentes combinações de quatro algoritmos de AM: máquinas de vetores de suporte, C4.5, redes neurais do tipo MLP e redes neurais do tipo RBF. Cada combinação é formada por classificadores induzidos por três desses algoritmos.

O diagnóstico de doenças na laringe, como câncer nas cordas vocais e surgimento de pólipos, nódulos e cistos, por exemplo, é geralmente realizado por fonoaudiólogos por meio de percepção visual e auditiva. Para isso, equipamentos, como tubos flexíveis, são geralmente introduzidos na boca do paciente, causando incômodo e desconforto. Além disso, as técnicas utilizadas têm limitações que dificultam um diagnóstico preciso.

Em Rosa et al. (1998), redes neurais são utilizadas para diagnosticar doenças na laringe de seres humanos. Foram realizados experimentos com 378 vozes obtidas de 119 pessoas, com e sem alguma doença na laringe. Cada paciente pronunciou por 5 segundos um grupo de fonemas da língua portuguesa. Técnicas de filtragem inversa e filtragem adaptativa foram utilizadas para extrair sete medidas acústicas de cada voz. Cada exemplo possuía, dessa forma, sete atributos de entrada. O atributo de saída para cada objeto era uma dentre seis classes, uma classe para voz normal e cinco para diferentes patologias.

Em Trambaiolli et al. (2009) empregaram-se máquinas de vetores de suporte no auxílio ao diagnóstico da doença de Alzheimer. Os dados eram provenientes de exames de eletroencefalograma (EEG) de pacientes com e sem a doença. Foram extraídos atributos

diversos desses exames relativos a coerências dos sinais de diferentes eletrodos. Utilizando um *kernel* do tipo RBF, foi possível obter uma taxa de acerto de 83% no diagnóstico, com 83,5% de sensibilidade e 82,7% de especificidade.

Parte VI

Tendências Futuras

A questão-chave na primeira definição de Aprendizado de Máquina, *Self constructing or self-modifying representations of what is being experienced for possible future use* (Mitchell, 1997), é a palavra *self*: sistemas que se modificam para melhor se acomodarem ao ambiente. O futuro da área de AM aponta para sistemas autônomos que podem incorporar o conhecimento, aprender a partir de dados não estacionários distribuídos em ambientes dinâmicos, com capacidade de transferir o conhecimento entre os problemas de aprendizado.

O objetivo do aprendizado automático é a construção de modelos computacionais que descrevem sistemas complexos a partir da observação do comportamento do sistema. Nas últimas duas décadas, a investigação e a prática do aprendizado automático centram-se no aprendizado a partir de conjuntos de dados relativamente pequenos, que podiam ser carregados na totalidade em memória. Regra geral, os algoritmos percorrem o conjunto de treinamento várias vezes, de forma a compensar o reduzido número de observações. A lógica por trás dessa prática é que os exemplos são independentes e gerados aleatoriamente por uma distribuição estacionária. A maioria dos algoritmos utiliza uma estratégia voraz, de subida da colina na pesquisa pelo espaço de modelos. Os modelos gerados são propensos a problemas de sobreajuste, máximos locais etc.

O desenvolvimento de novas tecnologias da informação e comunicações alterou drasticamente os processos de coleta, transformação e processamento de dados. O surgimento de novas tecnologias para redes de todos os tipos (nomeadamente redes sem fios) e os avanços na miniaturização e tecnologia de sensores possibilitam a coleta de informação espaço-temporal nos mais diversos domínios, com um nível de detalhe até antes impensável.

Ao longo do livro (com raras exceções como o CBR) foi assumida uma representação atributo-valor para as observações. Cada exemplo é descrito por um conjunto de variáveis (um registro na terminologia de base de dados) e um conjunto de exemplos é armazenado em uma matriz (a relação ou tabela na terminologia de banco de dados). Em algumas das mais desafiadoras aplicações de AM, os dados são descritos por sequências (por exemplo, dados de ADN), árvores (documentos XML) e gráficos (componentes químicos, análises de rede).

Os objetos que nos rodeiam, usualmente estáticos e inanimados, estão a tornar-se sistemas adaptativos e reativos com o potencial de serem cada vez mais úteis. Esses objetos, associados a redes de todos os tipos, oferecem novas e até o momento desconhecidas possibilidades para o desenvolvimento e a auto-organização de comunidades.

Um dos aspectos característicos desses contextos é a dinâmica da informação que circula entre os diferentes agentes. Esse aspecto implica algoritmos adaptativos que evoluem e se adaptam com o tempo. Hoje em dia, temos uma compreensão clara das linguagens

usadas para representar generalizações dos exemplos. A próxima geração de algoritmos de aprendizado deve considerar a gestão de custo/benefício, as limitações dos dispositivos físicos onde os algoritmos são executados e as limitações no conhecimento que aprendem.

Os algoritmos de aprendizado devem ser capazes de se adaptar continuamente a mudanças no ambiente em que atuam, incluindo o seu próprio regime de funcionamento. Por outro lado, devem ter em conta as restrições impostas pelo tempo de resposta esperado, poder computacional, comunicações e bateria limitados. Agentes com capacidade de adaptação, atuando em ambientes dinâmicos sob condições adversas, deverão ser capazes de se *autodiagnosticar* e de tomar ações preventivas para a eventualidade de falhas. O desenvolvimento de sistemas e mecanismos de autoconfiguração, auto otimização e autorreparação é um dos maiores desafios científicos. Todos esses aspectos requerem o monitoramento sobre a evolução do processo de aprendizado e a capacidade de raciocinar sobre ele.

Referências Bibliográficas

- Aamodt, A. e Plaza, E. (1994). Case based reasoning: foundational issues, methodological variations, and systems approaches. *AI Communications*, 7:39–59. Citado em: 66, 67, 68
- Aggarwal, C., Han, J., Wang, J. e Yu, P. (2003). A framework for clustering evolving data streams. In: *Proceedings of the 29th International Conference on Very Large Databases*, p. 81–92. Morgan Kaufmann. Citado em: 258, 261
- Agrawal, R., Gehrke, J., Gunopulos, D. e Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In: *Proceedings of 1998 ACM-SIGMOD International Conference on Management of Data*, p. 94–105. Citado em: 217
- Agrawal, R., Imielinski, T. e Swami, A. (1993). Mining association rules between sets of items in large databases. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, p. 207–216, Washington D.C.. Citado em: 181, 182
- Agrawal, R. e Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In: Bocca, J. B., Jarke, M. e Zaniolo, C. (Ed.) *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, p. 487–499, Santiago, Chile. Citado em: 182
- Aha, D. W., Kibler, D. e Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66. Citado em: 56, 64
- Ali, K. e Pazzani, M. (1996). Error reduction through learning multiple descriptions. *Machine Learning*, Vol. 24, No. 1. Citado em: 139, 140, 157
- Allwein, E. L., Shapire, R. E. e Singer, Y. (2000). Reducing multiclass to binary: a unifying approach for margin classifiers. In: *Proceedings of the 17th International Conference on Machine Learning*, p. 9–16. Citado em: 279, 282, 286
- Anderson, R. P., Laverde, M. e Peterson, A. T. (2002). Using niche-based gis modeling to test geographic predictions of competitive exclusion and competitive release in South American pocket mice. *Oikos*, 93:3–16. Citado em: 323
- Andrews, R., Cable, R., Diederich, J., Geva, S., Golea, M., Hayward, R., Ho-Stuart, C. e Tickle, A. B. (1996). An evaluation and comparison of techniques for extracting and refining rules from artificial neural networks. Relatório técnico, Queensland University of Technology. Citado em: 122
- Andrews, R., Diederich, J. e Tickle, A. B. (1995). A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks. *Knowledge-Based Systems*, 8(6):373–389. Citado em: 122
- Ankerst, M., Breunig, M. M., Kriegel, H.-P. e Sander, P. (1999). Optics: Ordering points to identify the clustering structure. In: *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'1999)*, p. 49–60. Citado em: 209
- Araújo, M. B., Williams, P. H. e Reginster, I. (2000). Selecting areas for species persistence using occurrence data. *Biological Conservation*, 96:331–345. Citado em: 323

- Arkin, R. C. (1998). *Behavior-based robotics*. MIT Press. Citado em: 333
- Armstrong, W. W., Dwelly, A., Liang, J., Liang, O., Lin, D., Reynolds, S., William, C. e Armstrong, W. (1991). Some results concerning adaptive logic networks. In: *Atree Release 2 Software Package*, p. 1173–1176. Citado em: 318
- Azuaje, F. (2002). A cluster validity framework for genome expression data. *Bioinformatics*, 18(2):319–320. Citado em: 243
- Babcock, B., Babu, S., Datar, M., Motwani, R. e Widom, J. (2002). Models and issues in data stream systems. In: Kolaitis, P. G. (Ed.) *Proceedings of the 21nd Symposium on Principles of Database Systems*, p. 1–16. ACM Press. Citado em: 260
- Baldi, P. e Brunak, S. (1998). *Bioinformatics - the machine learning approach*. The MIT Press. Citado em: 320
- Bandyopadhyay, S., Maulik, U. e Roy, D. (2008). Gene identification: classical and computational intelligence approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (applications and reviews)*, 38(1):55–68. Citado em: 321
- Baranauskas, J. A. e Monard, M. C. (2000). Reviewing some machine learning concepts and methods. Relatório Técnico 102, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Disponível em: ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_102.ps.zip. Citado em: 162
- Barbara, D. (2000). An introduction to cluster analysis for data mining. http://www-users.cs.umn.edu/~han/dmclass/cluster_survey_10_02_00.pdf [Acessado em 12/Nov./2003]. Citado em: 14, 192, 196, 210, 211, 213
- Barnard, G. A. (1963). New methods of quality control. *Journal Royal Statistical Society - Series A*, p. 126–255. Citado em: 137
- Barnett, V. e Lewis, T. (1994). *Outliers in statistical data*. John Wiley and Sons. Citado em: 40
- Basseville, M. e Nikiforov, I. (1987). *Detection of abrupt changes: theory and applications*. Prentice-Hall Inc. Citado em: 266
- Battiti, R. (1991). First and second-order methods for learning: between steepest descent and Newton's method. Relatório técnico, University of Trento. Citado em: 119
- Bauer, E. e Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–139. Citado em: 157
- Bay, S. D. (1998). Combining nearest neighbor classifiers through multiple feature subsets. In: Shavlik, J. (Ed.) *Proceedings of the 15th International Conference -ICML'98*. Morgan Kaufmann. Citado em: 149
- Ben-Hur, A., Horn, D., Siegelmann, H. e Vapnik, V. (2001). Support vector clustering. *Journal of Machine Learning Research*, 2:125–137. Citado em: 209
- Ben-Hur, A., Horn, D., Siegelmann, H. T. e Vapnik, V. N. (2000). A support vector clustering method. In: *Proceedings of the International Conference on Pattern Recognition (ICPR'00)*, vol. 2, p. 724–727. Citado em: 107
- Bensusan, H. (1998). God doesn't always shave with occam's razor - learning when and how to prune. In: *Proceedigs of the 10th European Conference on Machine Learning*, p. 119–124. Springer. Citado em: 273

- Bensusan, H. e Giraud-Carrier, C. (2000a). Casa batlo is in passeig de gracia or landmarking the expertise space. In: *Proceedings of the ECML'2000 Workshop on Meta-Learning: building automatic advice strategies for model selection and method combination*, p. 29–47. ECML'2000. Citado em: 273
- Bensusan, H. e Giraud-Carrier, C. (2000b). Discovering task neighbourhoods through landmark learning performances. In: Zighed, D., Komorowski, J. e Zytkow, J. (Ed.) *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, p. 325–331. Springer. Citado em: 274
- Bensusan, H., Giraud-Carrier, C. e Kennedy, C. (2000). A higher-order approach to meta-learning. In: *Proceedings of the ECML'2000 Workshop on Meta-Learning: building automatic advice strategies for model selection and method combination*, p. 109–117. ECML'2000. Citado em: 273
- Bentley, J. (1975). Multidimensional binary search tree used for associative searching. *Machine Learning*, 18(9):509–517. Citado em: 65
- Berkhin, P. (2002). Survey of clustering data mining techniques. Relatório técnico, Accrue Software, San Jose, CA. http://www.accrue.com/products/rp_cluster_review.pdf [Acessado em 05/Fev./2004]. Citado em: 213, 214
- Berry, P. M. (2002). Modelling potential impacts of climate change on the bioclimatic envelope of species in Britain and Ireland. *Global Ecology & Biogeography*, 11:453–462. Citado em: 323
- Beyer, K., Goldstein, J., Ramakrishnan, R. e Shaft, U. (1999). When is “nearest neighbor” meaningful? In: *International Conference on Database Theory*, p. 217–235. ACM. Citado em: 64
- Bezdek, J. C. e Pal, N. R. (1998). Some new indexes of cluster validity. *IEEE Trans. Syst., Man, Cybernetics - Part B: Cybernetics*, 28(3):301–315. Citado em: 243
- Bifet, A. e Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In: *Proceedings SIAM International Conference on Data Mining*, p. 443–448, Minneapolis, USA. SIAM. Citado em: 267
- Blake, C., Keogh, E. e Merz, C. (1999). UCI repository of Machine Learning Databases. Citado em: 105
- Blanco, R., Inza, I., Merino, I., Quiroga, M. e Larrañaga, P. (2005). Feature selection in Bayesian classifiers for the prognosis of survival of cirrhotic patients treated with tips. *Internat.Journal of Biomed. Informatics*, 38(5). Citado em: 82
- Blockeel, H., Bruynooghe, M., Dzeroski, S., Ramon, J. e Struyf, J. (2002). Hierarchical multi-classification. In: *Proceedings of the ACM SIGKDD 2002 Workshop on Multi-Relational Data Mining (MRDM 2002)*, p. 21–35. Citado em: 302, 303
- Blum, A. L. e Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271. Citado em: 50
- Blum, C. (2005). Ant colony optimization: introduction and recent trends. *Physics of Life Reviews*, 2:353–373. Citado em: 306
- Blum, C. e Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308. Citado em: 305
- Bohr, H. G. (1999). Neural networks for protein structure prediction. vol. 522 of *Lecture Notes in Physics*, p. 189–206. Citado em: 321
- Boser, B. E., Guyon, I. L. e Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In: *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, p. 144–152, Pittsburg, Pennsylvania, US. Citado em: 125

- Boser, R. C. e Ray-Chaudhuri, D. K. (1960). On a class of error-correcting binary group codes. *Information and Control*, 3:68–79. Citado em: 282
- Boutell, M. R., Luo, J., Shen, X. e Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771. Citado em: 288, 295
- Boutin, F. e Hascoët, M. (2004). Cluster validity indices for graph partitioning. In: *Eighth International Conference on Information Visualisation (IV'2004)*, p. 376–381, London, England. Citado em: 252
- Braga, A. P., Carvalho, A. C. P. L. F. e Ludermir, T. B. (2007). *Redes neurais artificiais: teoria e aplicações*. LTC. Citado em: 107, 110, 121
- Bratko, I. (1984). *Prolog, Programming for Artificial Intelligence*. Addison-Wesley Publishing Company. Citado em: 93
- Brazdil, P., Giraud-Carrier, C., Soares, C. e Vilalta, R. (2009). *Metalearning: applications to data mining*. Cognitive Technologies. Springer. Citado em: 258, 270, 271, 273, 274, 277
- Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24:123–140. Citado em: 144, 145
- Breiman, L. (1996b). Bias, variance, and arcing classifiers. Relatório Técnico 460, Statistics Department, University of California. Citado em: 96, 173
- Breiman, L. (1996c). Stacked regressions. *Machine Learning*, 24:49–64. Citado em: 151, 152
- Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics*, 26(3):801–849. Citado em: 157, 173
- Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32. Citado em: 149
- Breiman, L., Friedman, J., Olshen, R. e Stone, C. (1984). *Classification and Regression Trees*. Wadsworth International Group., USA. Citado em: 83, 86, 90, 91, 92, 93, 95, 96, 99, 104, 278
- Brodley, C. (1993). Addressing the selective superiority problem: Automatic algorithm / model class selection problem. In: Utgoff, P. (Ed.) *Machine Learning, Proceedings of the 10th International Conference*. Morgan Kaufmann. Citado em: 156
- Brodley, C. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, 20. Citado em: 156
- Buntine, W. (1990). *A Theory of Learning Classification Rules*. Tese de Doutorado, University of Sydney. Citado em: 105
- Buntine, W. e Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8:75–85. Citado em: 91
- Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):1–43. Citado em: 123, 124, 128, 129, 135
- Calinski, R. e Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3:1–27. Citado em: 245
- Callaghan, L., Mishra, N., Meyerson, A., Guha, S. e Motwani, R. (2002). Streaming-data algorithms for high-quality clustering. In: *Proceedings of IEEE International Conference on Data Engineering*, p. 685. Citado em: 261
- Campbell, C. (2000). An introduction to kernel methods. In: Howlett, R. J. e Jain, L. C. (Ed.) *Radial Basis Function Networks: Design and Applications*, p. 155–192, Berlin. Springer Verlag. Citado em: 126
- Campello, R. J. G. B. (2007). A fuzzy extension of the rand index and other related indexes for clustering and classification assessment. *Pattern Recogn. Lett.*, 28:833–841. Citado em: 253

- Carvalho, A. C. P. L. F., Braga, A. P. e Ludermir, T. B. (2003). Computação evolutiva. In: Rezende, S. O. (Ed.) *Sistemas Inteligentes: Fundamentos e Aplicações*, Capítulo 9, p. 225–248. Editora Manole Ltda. Citado em: 311, 313
- Casley, D. (1992). Primer on molecular biology. Relatório técnico, U. S. Department of Energy, Office of Health and Environmental Research. Citado em: 320
- Castillo, G. (2006). *Adaptive Learning Algorithms for Bayesian Network Classifiers*. Ph.D. thesis, University of Aveiro. Citado em: 81, 82
- Castillo, G. e Gama, J. (2005). Bias management of Bayesian networks classifiers. In: *Proceedings of the 8th International Conference of Discovery Science*, vol. 3735 of *LNAI*, p. 70–83. Springer Verlag. Citado em: 81, 82
- Castro, L. N. (2006). *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*. Chapman & Hall/CRC. Citado em: 306, 309
- Castro, L. N. (2007). Fundamentals of natural computing: an overview. *Physics of Life Reviews*, 4(1):1–36. Citado em: 305
- Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In: Kodratoff, Y. (Ed.) *European Working Session on Learning -EWSL91*. LNAI 482 Springer Verlag. Citado em: 96
- Cerri, R. e Carvalho, A. C. (2010). New top-down methods using SVMs for hierarchical multilabel classification problems. In: *2010 International Joint Conference on Neural Networks*, p. 3064–3071. IEEE. Citado em: 321
- Cerri, R., Silva, R. R. e Carvalho, A. C. (2009). Comparing methods for multilabel classification of proteins using machine learning techniques. In: *BSB '09: Proceedings of the 4th Brazilian Symposium on Bioinformatics*, p. 109–120, Berlin, Heidelberg. Springer-Verlag. Citado em: 321
- Cestnik, B., Kononenko, I. e Bratko, I. (1987). Assistant 86: a knowledge-elicitation tool for sophisticated users. In: Bratko, I. e Lavrac, N. (Ed.) *European Working Session on Learning -EWSL87*. Sigma Press, Wilmslow, England. Citado em: 83
- Chan, P. e Stolfo, S. (1995a). A comparative evaluation of voting and meta-learning on partitioned data. In: Prieditis, A. e Russel, S. (Ed.) *Machine Learning Proc. of 12th International Conference*. Morgan Kaufmann. Citado em: 155
- Chan, P. e Stolfo, S. (1995b). Learning arbiter and combiner trees from partitioned data for scaling machine learning. In: Fayyad, U. M. e Uthurusamy, R. (Ed.) *Proc. of the First Intern. Conference on Knowledge Discovery and Data Mining*. AAAI Press. Citado em: 155
- Chan, P. e Stolfo, S. (1997). On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information systems*, 8:5–28. Citado em: 155
- Chapelle, O., Vapnik, V., Bousquet, O. e Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159. Citado em: 135
- Chen, L. e Pung, H. K. (2008). Convergence analysis of convex incremental neural networks. *Annals of Mathematics and Artificial Intelligence*, 52:67–80. Citado em: 266
- Cheng, J. e Greiner, R. (1999). Comparing Bayesian network classifiers. In: *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, p. 101–108. Morgan Kaufmann Publishers Inc. Citado em: 80
- Cheng, J., Tegge, A. N. e Baldi, P. (2008). Machine learning methods for protein structure prediction. *IEEE Reviews in Biomedical Engineering*, 1:41–49. Citado em: 321

- Cheng, Y. e Church, G. (2000). Bioclustering of expression data. In: *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB'2000)*, p. 93–103. Citado em: 209
- Chi, Y., Wang, H., Yu, P. S. e Muntz, R. R. (2004). Moment: Maintaining closed frequent itemsets over a stream sliding window. In: *Proceedings of the IEEE International Conference on Data Mining*, p. 59–66, Brighton, UK. Citado em: 185
- Chiang, J.-H. e Hao, P.-Y. (2003). A new kernel-based fuzzy clustering approach: support vector clustering with cell growing. *IEEE Transactions on Fuzzy Systems*, 11(4):518–527. Citado em: 209
- Clare, A. e King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In: *5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2001), volume 2168 of Lecture Notes in Artificial Intelligence*, p. 42–53. Springer. Citado em: 288, 293
- Clare, A. e King, R. D. (2003). Predicting gene function in *saccharomyces cerevisiae*. *Bioinformatics*, 19(2):42–53. Citado em: 297, 302
- Clark, P. e Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283. Citado em: 98, 99, 100, 101
- Coelho, A. L., Fernandes, E. e Faceli, K. (2010). Inducing multi-objective clustering ensembles with genetic programming. *Neurocomputing*, In Press, Corrected Proof. Citado em: 219, 234
- Cohen, W. (1995). Fast effective rule induction. In: Prieditis, A. e Russel, S. (Ed.) *Machine Learning, Proceedings of the 12th International Conference*. Morgan Kaufmann. Citado em: 98
- Corne, D. W., Jerram, N. R., Knowles, J. D. e Oates, M. J. (2001). PESA-II: Region-based selection in evolutionary multiobjective optimization. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, p. 283–290, San Francisco, California, USA. Morgan Kaufmann. Citado em: 231
- Cortes, C. e Vapnik, V. N. (1995). Support vector networks. *Machine Learning*, 20(3):273–296. Citado em: 125
- Cortez, P. e Morais, A. (2007). A data mining approach to predict forest fires using meteorological data. In: Neves, J. et al. (Ed.) *New Trends in Artificial Intelligence, 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence*, p. 512–523, Guimarães, Portugal. APPIA. Citado em: 324
- Cortez, P., Portelinha, M., Rodrigues, S., Cadavez, V. e Teixeira, A. (2006). Lamb meat quality assessment by support vector machines. *Neural Processing Letters*, 24(1):41–51. Citado em: 318
- Costa, E. P., Lorena, A. C., Carvalho, A. C. P. L. F. e Freitas, A. A. (2007). A review of performance evaluation measures for hierarchical classifiers. *AAAI07 - II Workshop on Evaluation for Machine Learning - 22nd Conference on Artificial Intelligence*, p. 1–6. Citado em: 303
- Costa, E. P., Lorena, A. C., Carvalho, A. C. P. L. F. e Freitas, A. A. (2008). Top-down hierarchical ensembles of classifiers for predicting g-protein-coupled-receptor functions. In: *III Brazilian Symposium on Bioinformatics (BSB)*, vol. 5167 of *Lecture Notes in Bioinformatics*, p. 35–46. Citado em: 321
- Craven, M. e Shavlik, J. (1997). Using neural networks for data mining. *Future Generation Computer Systems*, 13:211–229. Citado em: 266
- Craven, M. e Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In: *ICML*, p. 37–45. Citado em: 122, 321
- Cristianini, N. e Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press. Citado em: 107, 122, 129, 135, 292
- Cybenko, G. (1989). Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals and Systems*, 2:303–314. Citado em: 115

- de Carvalho, A. e Freitas, A. (2009). A tutorial on multi-label classification techniques. vol. Foundations of Computational Intelligence Vol. 5 of *Studies in Computational Intelligence* 205, p. 177–195. Springer. Citado em: 289, 291, 292, 293, 295
- de Comite, F., Gilleron, R. e Tommasi, M. (2003). Learning multi-label alternating decision trees from texts and data. In: *International Conference on Machine Learning and Data Mining*, number 2734 In: LNAI, p. 35–49. SV. Citado em: 293
- de Faveri Honorato, D., Monard, M. C., Cherman, E. A., Lee, H. D. e Wu, F. C. (2008). Construção de uma representação atributo-valor para extração de conhecimento a partir de informações semi-estruturadas de laudos médicos. *CLEI Electronic Journal*, 11:1–12. Citado em: 336
- de Souto, M. C. P., Prudêncio, R. B. C., Soares, R. G. F., de Araujo, D. S. A., Costa, I. G., Ludermir, T. B. e Schliep, A. (2008). Ranking and selecting clustering algorithms using a meta-learning approach. In: *IJCNN*, p. 3729–3735. Citado em: 276
- de Souza, B. F. (2010). *Meta-aprendizagem aplicada à classificação de dados de expressão gênica*. Tese de Doutorado, Universidade de São Paulo. Citado em: 270, 272, 273, 276
- de Souza, B. F., Carvalho, A. C. e Soares, C. (2010). A comprehensive comparison of ml algorithms for gene expression data classification. In: *2010 International Joint Conference on Neural Networks*, p. 98–105. IEEE. Citado em: 276, 322
- Delbem, A. C. B., de Carvalho, A. C. P. L. F. e Bretas, N. G. (2003). Optimal energy restoration in radial distribution systems using a genetic approach and graph chain representation. *Electric Power Systems Research*, 67(3):197–205. Citado em: 327
- Delbem, A. C. B., de Carvalho, A. C. P. L. F. e Bretas, N. G. (2005). Main chain representation for evolutionary algorithms applied to distribution system reconfiguration. *IEEE Transaction on Power Systems*, 20(1):425–436. Citado em: 327
- Demsár, J. (2006). Statistical comparisons of classifiers over multiple datasets. *Journal of Machine Learning Research*, 7:1–30. Citado em: 169, 170, 171
- Devore, J. L. (2006). *Probabilidade e Estatística: para Engenharia e Ciências*. Thompson. Citado em: 161, 168
- Dietterich, T. (1997). Machine learning research: four current directions. *AI Magazine*, 18(4):97–136. Citado em: 137, 138, 146
- Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924. Citado em: 54, 169
- Dietterich, T. G. e Bariki, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286. Citado em: 280, 281, 282, 286
- Dom, B. E. (2002). An information-theoretic external cluster-validity measure. In: *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-2002)*, p. 137–145, San Francisco, CA. Morgan Kaufmann Publishers. Citado em: 252
- Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168. Citado em: 98, 156
- Domingos, P. (1997a). Bayesian model averaging in rule induction. In: Smyth, P. e Madigan, D. (Ed.) *Preliminary papers of the Sixth International Workshop on Artificial Intelligence and Statistics*. Citado em: 140
- Domingos, P. (1997b). *A Unified Approach to Concept Learning*. Tese de Doutorado, University of California, Irvine. Citado em: 156, 262

- Domingos, P. (1998). Knowledge discovery via multiple models. *Intelligent Data Analysis*, 24(2). Citado em: 156
- Domingos, P. e Hulten, G. (2000). Mining High-Speed Data Streams. In: Parsa, I., Ramakrishnan, R. e Stolfo, S. (Ed.) *Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining*, p. 71–80. ACM Press. Citado em: 258, 261
- Domingos, P. e Hulten, G. (2001). A general method for scaling up machine learning algorithms and its application to clustering. In: *Proceedings of the Eighteenth International Conference on Machine Learning*, p. 106–113. Citado em: 261
- Domingos, P. e Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–129. Citado em: 74, 76, 80
- Dopazo, J., Zanders, E., Dragoni, I., Amphlett, G. e Falciani, F. (2001). Methods and approaches in the analysis of gene expression data. *Journal of Immunological Methods*, 250(1-2):93 – 112. Citado em: 49
- Dorigo, M., Birattari, M. e Stützle, T. (2006). Ant colony optimization - artificial ants as a computational intelligence technique. *IEEE Comput. Intell. Mag.*, 1:28–39. Citado em: 305, 306
- Dorigo, M. e Di-Caro, G. (1999). The ant colony optimization metaheuristic. In: Corne, D., Dorigo, M. e Glover, F. (Ed.) *New Ideas in Optimization*, p. 11–32. McGraw-Hill, London, UK. Citado em: 307
- Dougherty, J., Kohavi, R. e Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In: Prieditis, A. e Russel, S. (Ed.) *Machine Learning Proc. of 12th International Conference*. Morgan Kaufmann. Citado em: 74
- Duan, K., Keerthi, S. S. e Poo, A. N. (2003). Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51:41–59. Citado em: 135
- Dubes, R. e Jain, A. (1976). Clustering techniques: The user's dilemma. *Pattern Recognition*, 8:247–260. Citado em: 206
- Duda, R. O., Hart, P. E. e Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience, 2. ed. Citado em: 56, 73, 209, 213
- Dudoit, S. e Fridlyand, J. (2002). A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology*, 3(7):research0036.1–0036.21. Citado em: 249
- Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64. Citado em: 171
- Ebecken, N. F. F., Lopes, M. C. S. e Costa, M. C. A. (2003). Mineração de textos. In: Rezende (2003), p. 337–370. Citado em: 332
- Egbert, S. L., Peterson, A. T., Sanchez-Cordero, C. e Price, K. P. (1999). Modeling conservation priorities in Veracruz, Mexico. In: *GIS in natural resource management: Balancing the technical-political equation*, p. 141–150. OnWord Press. Citado em: 323
- Eisner, R., Poulin, B., Szafron, D., Lu, P. e Greiner, R. (2005). Improving protein function prediction using the hierarchical structure of the gene ontology. In: *Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, p. 1–10. Citado em: 301
- Elisseeff, A. e Weston, J. (2001a). Kernel methods for multi-labelled classification and categorical regression problems. Relatório técnico, Biowulf Technologies. Citado em: 289
- Elisseeff, A. E. e Weston, J. (2001b). A kernel method for multi-labelled classification. In: *In Advances in Neural Information Processing Systems 14*, p. 681–687. MIT Press. Citado em: 288

- Elith, J., Crahanm, C. H., Anderson, R. P., Miroslav, M. D., Ferrier, S., Guisan, A., Hijmans, R. J., Hucttmann, F., Leathwick, J. R., Lehmann, A., Li, J., Lohmann, L. G., Loiselle, B. A., Manion, G., Moritz, G., Nakamura, M., Nakazawa, Y., Overton, J. M., Peterson, A. T., Phillips, S. J., Richardson, K., Scachetti-Pereira, R., Schapire, R. E., Soberón, J., Williams, S., Wisz, M. S. e Zimmermann, N. (2006). Novel methods improve prediction of species' distributions from occurrence data. *Ecography*, 29:129–151. Citado em: 324, 325
- Ertöz, L., Steinbach, M. e Kumar, V. (2002). A new shared nearest neighbor clustering algorithm and its applications. In: *Proceedings of the Workshop on Clustering High Dimensional Data and its Applications, 2nd SIAM International Conference on Data Mining (SDM'2002)*, p. 105–115. Citado em: 209
- Esposito, F., Malerba, D. e Semeraro, G. (1993). Decision tree pruning as a search in the state space. In: Brazdil, P. (Ed.) *Machine Learning: ECML-93*. LNAI 667, Springer Verlag. Citado em: 92, 94
- Esposito, F., Malerba, D. e Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *Transactions on Pattern Analysis and Machine Intelligence*, 19(5). Citado em: 91, 92, 94
- Ester, M., Kriegel, H.-P., Sander, J. e Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Proceedings of 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'1996)*, p. 226–231. Citado em: 214
- Estivill-Castro, V. (2002). Why so many clustering algorithms - a position paper. *SIGKDD Explorations*, 4(1):65–75. Citado em: 192, 194, 204, 205
- Faceli, K., Souto, M. C. P., de Araújo, D. S. A. e Carvalho, A. C. F. L. F. (2009). Multi-objective clustering ensemble for gene expression data analysis. *Neurocomputing*, 72(13-15):2763–2774. Citado em: 219, 234
- Fagni, T. e Sebastiani, F. (2007). On the selection of negative examples for hierarchical text categorization. In: *Proceedings of the 3rd Language Technology Conference*, p. 24–28. Citado em: 301
- Fahlman, S. E. (1988). An empirical study of learning speed in backpropagation networks. Relatório técnico, Carnegie Mellon University. Citado em: 119
- Falkman, G. (2002). Adaptation using interactive estimations. In: Craw, S. e Preece, A. (Ed.) *6th European Conference on Case-Based Reasoning, Aberdeen, Scotland, Uk*, p. 88–102. Springer Verlag. Citado em: 69
- Fawcett, T. (2005). An introduction to ROC analysis. *Pattern Recognition Letters*, p. 861–874. Citado em: 165
- Fayyad, U. e Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In: *13th International Joint Conference of Artificial Intelligence*, p. 1022–1029. Morgan Kaufman. Citado em: 96
- Fayyad, U., Piatetski-Shapiro, G. e Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, p. 27–34. Citado em: 331
- Fayyad, U. M. e Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102. Citado em: 90
- Feeelders, A. e Verkooijen, W. (1996). On the statistical comparison of inductive learning methods. In: Fisher, D. e Lenz, H.-J. (Ed.) *Learning from Data: Artificial Intelligence and Statistics V*, p. 272–279. Springer Verlag, New York, NY. Citado em: 171
- Fern, X. Z. e Brodley, C. E. (2004). Solving cluster ensemble problems by bipartite graph partitioning. In: *Proceedings of the Twenty First International Conference on Machine Learning (ICML'2004)*, p. 36, New York, NY, USA. ACM Press. Citado em: 219, 221, 222, 223, 224, 225, 229, 230

- Ferrer-Troyano, F., Aguilar-Ruiz, J. e Riquelme, J. (2005). Incremental rule learning and border examples selection from numerical data streams. *Journal of Universal Computer Science*, 11(8):1426–1439. Citado em: 261
- Ferrier, S. (2002). Extended statistical approaches to modelling spatial pattern in biodiversity: the northeast New South Wales experience. I. species-level modelling. *Biodiversity and Conservation*, 11:2275–2307. Citado em: 323
- Filho, I. G. C. (2003). Comparative analysis of clustering methods for gene expression data. Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco, Recife. Citado em: 241, 254
- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals Eugen.*, 7:179–188. Citado em: 18
- Fix, E. (1951). Discriminatory analysis: Nonparametric discrimination: Consistency properties. Relatório Técnico Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolph Field, Texas. Citado em: 31
- Frank, A. e Asuncion, A. (2010). UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml>. Citado em: 75, 335
- Frank, E. e Kramer, S. (2004). Ensembles of nested dichotomies for multi-class problems. In: *Proceedings of the 21st International Conference on Machine Learning*, p. 305–312. Citado em: 285
- Frank, E., Wang, Y., Inglis, S., Holmes, G. e Witten, I. H. (1998). Using model trees for classification. *Machine Learning*, 32(1):63–76. Citado em: 104
- Frank, E. e Witten, I. H. (1998). Generating accurate rule sets without global optimization. In: Shavlik, J. (Ed.) *Proceedings of the 15th International Conference -ICML'98*, p. 144–151. Morgan Kaufmann. Citado em: 99
- Fred, A. e Jain, A. (2002). Evidence accumulation clustering based on the k-means algorithm. In: *Proceedings of Structural and Syntactic Pattern Recognition (SSPR'2002)*, p. 442–451, Windsor, Canada. Citado em: 219, 221, 222, 224
- Fred, A. e Jain, A. K. (2003). Robust data clustering. In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, (CVPR'2003)*, vol. II, Madison - Wisconsin, USA. Citado em: 219, 221, 222, 223, 224
- Fred, A. L. N. (2001). Finding consistent clusters in data partitions. In: Kittler, J. e Roli, F. (Ed.) *Second International Workshop on Multiple Classifier Systems (MCS'2001)*, vol. 2096 of *Lecture Notes in Computer Science*, p. 309–318, Cambridge, UK. Citado em: 219, 221, 223, 224
- Freitas, A. A. e de Carvalho, A. C. P. L. F. (2007). A tutorial on hierarchical classification with applications in bioinformatics. *Intelligent Information Technologies: Concepts, Methodologies, Tools and Applications*, 1:114–145. Citado em: 297, 321
- Freund, Y. e Mason, L. (1999). The alternating decision tree learning algorithm. In: *In Machine Learning: Proceedings of the Sixteenth International Conference*, p. 124–133. Morgan Kaufmann. Citado em: 293
- Freund, Y. e Schapire, R. (1999). A short introduction to boosting. *Japanese Society for Artificial Intelligence*, 14(5):771–780. Citado em: 293
- Freund, Y. e Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In: *European Conference on Computational Learning Theory*, p. 23–37. Citado em: 293

- Freund, Y. e Schapire, R. E. (1996). Experiments with a new boosting algorithm. In: Saitta, L. (Ed.) *Machine Learning, Proc. of the 13th International Conference*. Morgan Kaufmann. Citado em: 147, 157
- Friedman, J. (1999). Greedy Function Approximation: a gradient boosting machine. Relatório técnico, Statistics Department, Stanford University. Citado em: 95
- Friedman, J. H., Kohavi, R. e Yun, Y. (1996). Lazy decision trees. In: AAAI (Ed.) *Thirteenth National Conference on Artificial Intelligence*. MIT Press. Citado em: 96
- Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701. Citado em: 171
- Friedman, N., Geiger, D. e Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163. Citado em: 80
- Fritzke, B. (1994). Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460. Citado em: 216
- Frossyniotis, D., Pertselakis, M. e Stafylopatis, A. (2002). A multi-clustering fusion algorithm. In: Vlahavas, I. P. e Spyropoulos, C. D. (Ed.) *Methods and Applications of Artificial Intelligence, Proceedings of the 2nd Hellenic Conference on AI (SETN'2002)*, vol. 2308 of *Lecture Notes in Computer Science*, p. 225–236, Thessaloniki, Greece. Springer Verlag. Citado em: 221, 223, 224
- Fu, L. (1994). Rule generation from neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(8):1114–1124. Citado em: 122
- Fu, X., Ong, C., Keerthi, S., Hung, G. G. e Goh, L. (2004). Extracting the knowledge embedded in support vector machines. In: *Proceedings IEEE International Joint Conference on Neural Networks*, vol. 1, p. 296. Citado em: 135
- Furnkranz, J. (2002). Round Robin classification. *Journal of Machine Learning Research*, 2:721–747. Citado em: 278, 280
- Furnkranz, J. (2003). Round Robin ensembles. *Intelligent Data Analysis*, 7(5):385–404. Citado em: 280
- Gama, J. (2000). A linear-Bayes classifier. In: Monard, C. e Sichman, J. (Ed.) *Advances on Artificial Intelligence - SBIA2000*, p. 269–279. LNAI 1952 Springer Verlag. Citado em: 78
- Gama, J. e Brazdil, P. (2000). Cascade generalization. *Machine Learning*, 41:315–343. Citado em: 152
- Gama, J., Medas, P., Castillo, G. e Rodrigues, P. (2004). Learning with drift detection. In: Bazzan, A. L. C. e Labidi, S. (Ed.) *Advances in Artificial Intelligence - SBIA 2004*, vol. 3171 of *Lecture Notes in Computer Science*, p. 286–295. Springer Verlag. Citado em: 267
- Gama, J., Rocha, R. e Medas, P. (2003). Accurate decision trees for mining high-speed data streams. In: *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 523–528. ACM Press. Citado em: 258, 261
- Gama, J. M. P. (1999). *Combining Classification Algorithms*. Tese de Doutorado, Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto. <http://www.ncc.up.pt/~jgama/tese.ps.gz> [Acessado em 19/Jul./2002]. Citado em: 220
- García, S. e Herrera, F. (2008). An extension on “statistical comparisons of classifiers over multiple data sets” for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694. Citado em: 169, 171
- Garg, A., Bhasin, M. e Raghava, G. P. S. (2005). Support vector machine-based method for subcellular localization of human proteins using amino acid compositions, their order, and similarity search. *The Journal of Biological Chemistry*, 280(15):14427–14432. Citado em: 321

- Geman, S., Bienenstock, E. e Doursat, R. (1992). Neural networks and the bias/variance dilemma. In: *Neural Computation*, vol. 4, p. 1–58. Citado em: 172
- Getz, G., Gal, H., Kela, I., Notterman, D. A. e Domany, E. (2003). Coupled two-way clustering analysis of breast cancer and colon cancer gene expression data. *Bioinformatics*, 19:1079–1089. Citado em: 209
- Geurts, P. (2000). *Contributions to Decision Tree Induction:bias/variance tradeoff and time series classification*. Tese de Doutorado, University of Liege. Citado em: 150
- Geurts, P. (2001). Dual perturb and combine algorithm. In: *Proc. of the Eighth International Workshop on Artificial Intelligence and Statistics*, p. 196–201. Springer Verlag. Citado em: 150
- Ghosh, J., Strehl, A. e Merugu, S. (2002). A consensus framework for integrating distributed clusterings under limited knowledge sharing. In: *Proceedings of NSF Workshop on Next Generation Data Mining*, p. 99–108. Citado em: 219, 221
- Giannella, C., Han, J., Pei, J., Yan, X. e Yu, P. (2003). Mining frequent patterns in data streams at multiple time granularities. In: Kargupta, H., Joshi, A., Sivakumar, K. e Yesha, Y. (Ed.) *Next Generation Data Mining*. AAAI/MIT. Citado em: 261
- Godbole, S. e Sarawagi, S. (2004). Discriminative methods for multi-labeled classification. In: *Advances in Knowledge Discovery and Data Mining*, p. 22–30. Citado em: 295
- Gonçalves, T. e Quaresma, P. (2003). A preliminary approach to the multilabel classification problem of portuguese juridical documents. In: *EPIA*, p. 435–444. Citado em: 288
- Gondro, C. e Kinghorn, B. P. (2007). A simple genetic algorithm for multiple sequence alignment. *Genet. Mol. Res.*, 6(4):964–982. Citado em: 321
- Gordon, A. (1999). *Classification*. Chapman & Hall/CRC. Citado em: 198, 199, 237, 239, 252
- Gordon, A. D. (1996). *From Data to Knowledge: Theoretical and Practical Aspects of Classification, Data Analysis and Knowledge Organization*, Capítulo Null models in cluster validation, p. 32–44. Springer-Verlag. Citado em: 241
- Graham, C. H., Moritz, C. e Williams, S. E. (2006). Habitat history improves prediction of biodiversity in rainforest fauna. *PNAS (Proceedings of the National Academy of Sciences of the United States of America)*, 103:632–636. Citado em: 323
- Granger, C. W. J. e Newbold, P. (1976). The use of r² to determine the appropriate transformation of regression variables. *J. Econometrics*, 4:205–210. Citado em: 137
- Greengrass, E. (2001). *Information Retrieval: a survey*. Relatório Técnico TR-R52-008-001, United States Department of Defense. Citado em: 332
- Guha, S., Rastogi, R. e Shim, K. (1998). CURE: an efficient clustering algorithm for large databases. In: *Proceedings of ACM SIGMOD International Conference on Management of Data*, p. 73–84. Citado em: 209
- Guha, S., Rastogi, R. e Shim, K. (2000). ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366. Citado em: 209
- Gutin, G., Punnen, A., Barvinok, A. e Edward Kh. Gimadi, A. I. S. (2002). *The Traveling Salesman Problem and Its Variations (Combinatorial Optimization)*. Springer. Citado em: 276
- Guyon, I. e Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182. Citado em: 49
- Hadjitodorov, S. T., Kuncheva, L. I. e Todorova, L. P. (2006). Moderate diversity for better cluster ensembles. *Information Fusion*, 7(3):264–275. Citado em: 221

- Hagan, M. e Menhaj, M. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993. Citado em: 119
- Halkidi, M., Batistakis, Y. e Vazirgiannis, M. (2001). On clustering validation techniques. *Intelligent Information Systems Journal*, 17(2-3):107–145. Citado em: 205, 207, 208, 213, 237, 239, 249
- Halkidi, M., Batistakis, Y. e Vazirgiannis, M. (2002a). Cluster validity methods: Part I. *SIGMOD Record*, 31(2):40–45. Citado em: 251
- Halkidi, M., Batistakis, Y. e Vazirgiannis, M. (2002b). Cluster validity methods: Part II. *SIGMOD Record*, 31(3):19–27. Citado em: 242, 243
- Halkidi, M. e Vazirgiannis, M. (2001). A data set oriented approach for clustering algorithm selection. In: *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, p. 165–179. Citado em: 245
- Hamming, R. (1950). Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29:147–160. Citado em: 281
- Han, J. e Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann. Citado em: 39, 178
- Han, J., Pei, J. e Yin, Y. (2000). Mining frequent patterns without candidate generation. In: *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, p. 1–12, New York, NY, USA. ACM Press. Citado em: 261
- Han, J., Pei, J., Yin, Y. e Mao, R. (2004). Mining frequent patterns without candidate generation. *Data Mining and Knowledge Discovery*, 8:53–87. Citado em: 185
- Handl, J. e Knowles, J. (2004). Multiobjective clustering with automatic determination of the number of clusters. Technical Report TR-COMPSYSBIO-2004-02, UMIST, Manchester. Citado em: 219, 220, 231, 232, 233
- Handl, J. e Knowles, J. (2005a). Exploiting the trade-off - the benefits of multiple objectives in data clustering. In: Coello, C. A. et al. (Ed.) *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO'2005)*, vol. 3410 of *Lecture Notes in Computer Science*, p. 547–560, Guanajuato, Mexico. Springer-Verlag. Citado em: 219, 231, 233
- Handl, J. e Knowles, J. (2005b). Improvements to the scalability of multiobjective clustering. In: *IEEE Congress on Evolutionary Computation*, p. 438–445. IEEE Computer Society Press. Citado em: 219, 231, 232, 233
- Handl, J. e Knowles, J. (2007). An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, 11(1):56–76. Citado em: 195, 219, 231, 244
- Handl, J., Knowles, J. e Kell, D. (2005). Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201–3212. Citado em: 192, 194, 237, 242
- Hannah, L., Midgley, G. F., Andelman, S., Araújo, M. B., Hughes, G. O., Martinez-Meyer, E., Pearson, R. G. e Williams, P. H. (2007). Protected area needs in a changing climate. *Frontiers in Ecology and the Environment*, 5:131–138. Citado em: 323
- Hannah, L., Midgley, G. F., Hughes, G. e Bomhard, B. (2005). The view from the cape: Extinction risk, protected areas, and climate change. *BioScience*, 55:231–242. Citado em: 323
- Hansen, L. e Salamon, P. (1990). Neural networks ensembles. *Transactions on Pattern Analysis and Machine Intelligence*, 12(10). Citado em: 137
- Haralick, R. M., Shanmugam, K. e Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621. Citado em: 318

- Hartigan, J. (1975). *Clustering Algorithms*. Wiley. Citado em: 245
- Hartigan, J. A. (1985). Statistical theory in clustering. *Journal of Classification*, 2(1):63–76. Citado em: 204
- Hartuv, E. e Shamir, R. (2000). A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(200):175–181. Citado em: 216
- Hastie, T. e Tibshirani, R. (1998). Classification by pairwise coupling. *The Annals of Statistics*, 26(2):451–471. Citado em: 280, 287
- Hastie, T., Tibshirani, R. e Friedman, J. (2001). *The Elements of Statistical Learning*. Springer New York. Citado em: 80
- Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, Upper Saddle River, NJ, USA, 2. ed. Citado em: 107, 109, 110, 121, 130, 136
- Hayward, R., Tickle, A. B. e Diederich, J. (1995). Extracting rules for grammar recognition from cascade-2 networks. In: *Learning for Natural Language Processing*, p. 48–60. Citado em: 122
- He, Q. (1999). A review of clustering algorithms as applied in IR. Relatório Técnico UIUCIS-1999/6+IRG, Information Retrieval Group, University of Illinois. Citado em: 198, 204
- Hearst, M. A., Schölkopf, B., Dumais, S., Osuna, E. e Platt, J. (1998). Trends and controversies - support vector machines. *IEEE Intelligent Systems*, 13(4):18–28. Citado em: 130
- Heath, D., Kasif, S. e Salzberg, S. (1996). Committees of decision trees. In: *Cognitive Technology: in Search of a Humane Interface*, p. 305–317. Elsevier Science. Citado em: 146
- Hebb, D. O. (1949). *The Organization of Behavior*. Wiley. Citado em: 108
- Herbrich, R. (2001). *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press. Citado em: 132
- Herrero, J., Valencia, A. e Dopazo, J. (2001). A hierarchical unsupervised growing neural network for clustering gene expression patterns. *Bioinformatics*, 17(2):126–136. Citado em: 216
- Higgins, S. I., Richardson, D. M., Cowling, R. M. e Trinder-Smith, T. H. (1999). Predicting the landscape-scale distribution of alien plants and their threat to plant diversity. *Conservation Biology*, 13:303–313. Citado em: 323
- Hinkley, D. (1970). Inference about the change point from cumulative sum-tests. *Biometrika*, 58:509–523. Citado em: 267
- Hinneburg, A. e Keim, D. A. (1998). An efficient approach to clustering in large multimedia databases with noise. In: *Proceedings of 4rd Int. Conf. on Knowledge Discovery and Data Mining*, p. 58–65. AAAI Press. Citado em: 214, 215
- Hinneburg, A. e Keim, D. A. (1999). Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In: *Proceedings of the 25th International Conference on Very Large Databases*, p. 506–517. Citado em: 217
- Hoeting, J. A., Madigan, D., Raftery, A. E. e Volinsky, C. T. (1999). Bayesian model averaging: A tutorial. *Statistical Science*, 14(1):382–401. Citado em: 137
- Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor. Citado em: 121
- Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91. Citado em: 98

- Hsu, C.-W., Chang, C.-C. e Lin, C.-J. (2003). *A Practical Guide to Support Vector Classification*. Relatório técnico, Department of Computer Science, National Taiwan University. Citado em: 132
- Hsu, C.-W. e Lin, C.-J. (2002). A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425. Citado em: 135, 278
- Hua, S. e Sun, Z. (2001). Support vector machine approach for protein subcellular localization prediction. *Bioinformatics*, 17(8):721–728. Citado em: 321
- Hubert, L. J. e Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2:193–218. Citado em: 246, 253
- Hugall, A. (2002). Reconciling paleodistribution models and comparative phylogeographic in the wet tropics rainforest lan snail *Gnarosophia bellendenkerensis*. *Brazier 1875*, 99:6112–6117. Citado em: 323
- Hulten, G., Spencer, L. e Domingos, P. (2001). Mining time-changing data streams. In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 97–106, San Francisco, California. ACM Press. Citado em: 264
- Huntley, B., Berry, P. M., Cramer, W. e McDonald, A. P. (1995). Modelling present and potential future ranges of some european higher plants using climate response surfaces. *J. Biogeography*, 22:967–1001. Citado em: 323
- Ihaka, R. e Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314. Citado em: 83
- Jain, A. e Dubes, R. (1988). *Algorithms for Clustering Data*. Prentice Hall. Citado em: 14, 191, 194, 197, 201, 202, 204, 206, 207, 208, 213, 237, 238, 239, 241, 245, 249, 251, 252
- Jain, A., Murty, M. e Flynn, P. (1999). Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323. Citado em: 196, 197, 199, 208, 213
- Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651 – 666. Award winning papers from the 19th International Conference on Pattern Recognition (ICPR), 19th International Conference in Pattern Recognition (ICPR). Citado em: 213, 218
- Jain, A. K., Dubes, R. C. e Chen, C.-C. (1987). Bootstrap techniques for error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):628–633. Citado em: 163
- Jarmulak, J., Craw, S. e Rowe, R. (2001). Using case-base data to learn adaptation knowledge for design. In: Nebel, B. (Ed.) *17th International Joint Conference on Artificial Intelligence*, Seattle, EUA, p. 1011–1020. Morgan Kaufmann. Citado em: 69
- Jiang, D., Tang, C. e Zhang, A. (2004). Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386. Citado em: 192, 205, 245
- Jin, R. e Agrawal, G. (2003). Efficient decision tree construction on streaming data. In: P.Domingos e Faloutsos, C. (Ed.) *Proceedings of the Ninth International Conference on Knowledge Discovery and Data Mining*. ACM Press. Citado em: 261
- Joachims, T. (2002). *Learning to Classify Text using Support Vector Machines*. Kluwer/Springer. Citado em: 122
- John, G. (1997). *Enhancements to the Data Mining Process*. Tese de Doutorado, Stanford University. Citado em: 78
- John, G., Kohavi, R. e Pfleger, K. (1994). Irrelevant features and the subset selection problem. In: Cohen, W. e Hirsh, H. (Ed.) *Machine Learning, Proceedings of the 11th International Conference*. Morgan Kaufmann. Citado em: 78

- Kalousis, A. (2002). *Algorithm Selection via Meta-Learning*. Tese de doutorado, Centre Universitaire d'Informatique, Université de Genève, Genebra, Suíça. Citado em: 271, 273, 274
- Kanda, J., de Carvalho, A. C. P. L. F., Hruschka, E. e Soares, C. (2010). Using meta-learning to classify traveling salesman problems. In: *XI Simpósio Brasileiro de Redes Neurais*, p. 73–78. Citado em: 276
- Karalic, A. e Pirnat, V. (1991). Significance level based multiple tree classification. In: *Informatica*, vol. 5. Citado em: 288
- Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242. Citado em: 121
- Karypis, G., Han, E.-H. S. e Kumar, V. (1999). Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75. Citado em: 209
- Karypis, G. e Kumar, V. (1999). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392. Citado em: 226, 228, 229
- Kaufman, L. e Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons. Citado em: 213
- Keerthi, S. S. e Lin, C.-J. (2003). Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation*, 15(7):1667–1689. Citado em: 132
- Kellam, P., Liu, X., Martin, N. J., Orengo, C., Swift, S. e Tucker, A. (2001). Comparing, contrasting and combining clusters in viral gene expression data. In: *Proceedings of 6th Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, p. 56–62. Citado em: 221, 222, 224
- Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*, vol. 4, p. 1942–1948, Perth, Australia. Citado em: 308
- Kennedy, J. e Eberhart, R. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers. Citado em: 308
- Kifer, D., Ben-David, S. e Gehrke, J. (2004). Detecting change in data streams. In: *VLDB 04: Proceedings of the 30th International Conference on Very Large Data Bases*, p. 180–191. Morgan Kaufmann Publishers Inc. Citado em: 267
- Kijsirikul, B. e Ussivakul, N. (2002). Multiclass support vector machines using adaptive directed acyclic graph. In: *Proceedings of International Joint Conference on Neural Networks (IJCNN 2002)*, p. 980–985. Citado em: 283, 284
- Kritchenco, S., Matwin, S. e Famili, A. F. (2004). Hierarchical text categorization as a tool of associating genes with gene ontology codes. In: *Proceedings of the 2nd European Workshop on Data Mining and Text Mining for Bioinformatics*, p. 26–30. Citado em: 303
- Kritchenco, S., Matwin, S., Nock, R. e Famili, A. F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In: *Proceedings of the 19th Canadian Conference on Artificial Intelligence*, vol. 4013 of *Lecture Notes in Artificial Intelligence*, p. 395–406. Citado em: 302
- Kittler, J. (1998). Combining classifiers: A theoretical framework. *Pattern Analysis and Applications*, 1(1). Citado em: 140, 141
- Klautau, A., Jevtić, N. e Orlitsky, A. (2003). On nearest-neighbor error-correcting output codes with application to all-pairs multiclass support vector machines. *Journal of Machine Learning Research*, 4:1–15. Citado em: 287
- Kleinberg, J. (2002). An impossibility theorem for clustering. *Advances in Neural Information Processing Systems*, 15:446–453. Citado em: 194

- Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300. Citado em: 266, 269
- Knerr, S., Personnaz, L. e Dreyfus, G. (1992). Handwritten digit recognition by neural networks with single-layer training. *IEEE Transactions on Neural Networks*, 3(6):962–968. Citado em: 278
- Koepf, C., Taylor, C. C. e Keller, J. (2000). Meta-analysis: From data characterisation for meta-learning to meta-regression. In: Brazdil, P. e Jorge, A. (Ed.) *Proceedings of the PKDD-00 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, Lyon, France. Citado em: 274
- Kohavi, R. (1996). Scaling up the accuracy of naive-Bayes classifiers: a decision tree hybrid. In: Simoudis, E., Han, J. W. e Fayyad, U. (Ed.) *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining*, p. 202–207. AAAI Press, USA. Citado em: 77, 156
- Kohavi, R. e Kunz, C. (1997). Option decision trees with majority votes. In: Fisher, D. (Ed.) *Machine Learning Proc. of 14th International Conference*. Morgan Kaufmann. Citado em: 96, 106
- Kohavi, R., Sommerfield, D. e Dougherty, J. (1997). Data mining using MLC++, a machine learning library in C++. *International Journal of Artificial Intelligence Tools*, 6 Nr. 4. Citado em: 95
- Kohavi, R. e Wolpert, D. (1996). Bias plus variance decomposition for zero-one loss functions. In: Saitta, L. (Ed.) *Proceedings of the 13th International Conference on Machine Learning*, p. 275–283. Morgan Kaufmann. Citado em: 173
- Kohonen, T. (2001). *Self-Organizing Maps*. Springer, Berlin. Citado em: 216
- Kong, E. B. e Dietterich, T. (1995). Error-correcting output coding correct bias and variance. In: Prieditis, A. e Russel, S. (Ed.) *Proceedings of the 12th International Conference on Machine Learning*. Morgan Kaufmann. Citado em: 173
- Kononenko, I. (1991). Semi-naive Bayesian classifier. In: Kodratoff, Y. (Ed.) *European Working Session on Learning -EWSL91*. LNAI 482 Springer Verlag. Citado em: 76, 78
- Kontkanen, P., Myllymäki, P., Silander, T. e Tirri, H. (1999). On supervised selection of Bayesian networks. In: *Proceedings of the 15th International Conference on Uncertainty in Artificial Intelligence*, p. 334–342. Morgan Kaufmann Publishers, Inc. Citado em: 80
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, 1 ed. Citado em: 313
- Krogh, A. e Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In: *Advances in Neural Information Processing Systems*, vol. 7, p. 231–238. Citado em: 220
- Krzanowski, W. e Lai, Y. (1985). A criterion for determining the number of groups in a dataset using sum of squares clustering. *Biometrics*, 44:23–34. Citado em: 245
- Kumar, S., Gosh, J. e Crawford, M. M. (2002). Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis and Applications*, 5:210–220. Citado em: 285
- Kuncheva, L. I. (2004). *Combining Pattern Classifiers*. John Wiley & Sons. Citado em: 220, 221
- Kuncheva, L. I., Hadjitodorov, S. T. e Todorova, L. P. (2006). Experimental comparison of cluster ensemble methods. In: *Proceedings of FUSION 2006*, p. 105–115. Citado em: 219, 221
- Lange, T., Braun, M., Roth, V. e Buhmann, J. (2003). Stability-based model selection. In: *Advances in Neural Information Processing Systems*, vol. 15, p. 617–624. Citado em: 248
- Langley, P. (1993). Induction of recursive bayesian classifiers. In: Brazdil, P. (Ed.) *Machine Learning: ECML-93*. LNAI 667, Springer Verlag. Citado em: 77

- Lauser, B. e Hotho, A. (2003). Automatic multi-label subject indexing in a multilingual environment. In: *Proc. of the 7th European Conference in Research and Advanced Technology for Digital Libraries, ECDL 2003*, vol. 2769, p. 140–151. Springer. Citado em: 288
- Law, M., Topchy, A. e Jain, A. K. (2004). Multiobjective data clustering. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, p. 424–430. Citado em: 193, 194, 219, 220, 221, 224
- Law, M. H. e Jain, A. K. (2003). Cluster Validity by Bootstrapping Partitions. Relatório Técnico MSU-CSE-03-5, Department of Computer Science and Engineering, Michigan State University. Citado em: 239, 245, 247, 248
- Lazzeroni, L. e Owen, A. (2002). Plaid models for gene expression data. *Statistica Sinica*, 12(1):61–86. Citado em: 209
- Leathwick, J. R. (2005). Using multivariate adaptive regression splines to predict the distributions of New Zealand's freshwater diadromous fish. *Freshwater biology*, 50:2034–2052. Citado em: 323
- LeBlanc, M. e Tibshirani, R. (1993). Combining estimates in regression and classification. Relatório Técnico 9318, Department of Statistics, University of Toronto. Citado em: 220
- LeCun, Y. A., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Müller, E., Säckinger, E., Simard, P. Y. e Vapnik, V. N. (1995). Comparison of learning algorithms for handwritten digit recognition. In: Foulgerman-Soulié, F. e Gallinari, P. (Ed.) *Proceedings of the International Conference on Artificial Neural Networks (ICANN '95)*, vol. 2, p. 53–60, Nanterre, France. Citado em: 136
- Leite, P. T., de Carvalho, A. C. P. L. F. e Carneiro, A. A. F. M. (2002). Energetic operation planning using genetic algorithms. *IEEE Transaction on Power Systems*, 17(1):173–179. Citado em: 328
- Libralao, G. L., Almeida, O. C. P. e de Carvalho, A. C. P. L. F. (2005). Classification of ophthalmologic images using an ensemble of classifiers. In: *IEA/AIE*, p. 380–389. Citado em: 336
- Lin, H.-T. e Lin, C.-J. (2003). A study on sigmoid kernels for SVM and the training of non-psd kernels by smo-type methods. Relatório técnico, Department of Computer Science, National Taiwan University. Citado em: 133
- Loh, W. e Shih, Y. (1997). Split selection methods for classification trees. *Statistica Sinica*, 7:815–840. Citado em: 278
- Loiselle, B. A. (2003). Avoiding pitfalls of using species distribution models in conservation planning. *Conservation Biology*, 17:1591–1600. Citado em: 323
- Lorena, A. C. e Carvalho, A. C. P. L. F. (2007). Design of directed acyclic graph multiclass structures. *Neural Network World*, 17:657–674. Citado em: 284, 321
- Lorena, A. C. e Carvalho, A. C. P. L. F. (2008). Hierarchical decomposition of multiclass problems. *Neural Network World*, 5:407–425. Citado em: 285
- Lorena, A. C. e Carvalho, A. C. P. L. F. (2010). Building binary-tree-based multiclass classifiers using separability measures. *Neurocomputing*, 73:2837–2845. Citado em: 285
- Lorena, A. C., Jacintho, L. F. O., Siqueira, M. F., de Giovanni, R., Lohmann, L. G., de Carvalho, A. A. C. P. L. F. e Yamamoto, M. (2010). Comparing machine learning classifiers in potential distribution modelling. *Expert Systems with Applications*, 38:5268–5275. Citado em: 325
- Luo, F., Khan, L., Bastani, F., Yen, I.-L. e Zhou, J. (2004). A dynamical growing self-organizing tree (DGSOT) for hierarchical clustering gene expression profiles. *Bioinformatics*, 20(16):2605–2617. Citado em: 216

- Luo, F., Tang, K. e Khan, L. (2003). Hierarchical clustering of gene expression data. In: *3rd IEEE Symposium on BioInformatics and BioEngineering (BIBE'2003)*, p. 328–335. Citado em: 216
- Luo, X. e Zincir-Heywood, N. A. (2005). Evaluation of two systems on multi-class multi-label document classification. In: *International Symposium on Methodologies for Intelligent Systems*, p. 161–169. Citado em: 288
- Main, J., Dillom, T. e Shiu, S. (2001). A tutorial on case based reasoning. In: Pal, S., Dillon, T. e Yeung, D. (Ed.) *Soft Computing in Case Based Reasoning*. Springer Verlag. Citado em: 69
- Malek, M. (2001). Hybrid approaches for integrating neural networks and case-based reasoning: From loosely coupled to tightly coupled models. In: Pal, S., Dillon, T. e Yeung, D. (Ed.) *Soft Computing in Case Based Reasoning*. Springer Verlag. Citado em: 69
- Malek, M. e Amy, B. (1994). Integration of Case-based Reasoning and Neural Networks Approaches for Classification. Relatório Técnico 131 IMAG - 28 LIFIA, laboratoire Leibniz - IMAG. Disponível em <http://www-leibniz.imag.fr/RESEAUX/pub/malek.cbrnn.e.ps.gz>. Citado em: 67
- Manevitz, L. M., Yousef, M., Cristianini, N., Shawe-taylor, J. e Williamson, B. (2001). One-class SVMs for document classification. *Journal of Machine Learning Research*, 2:139–154. Citado em: 34
- Maniezzo, V., Gambardella, L. M. e Luigi, F. (2004). Ant colony optimization. In: Onwubolu, G. C. e Babu, B. V. (Ed.) *New Optimization Techniques in Engineering*, p. 101–117. Springer-Verlag, Berlin, Heidelberg. Citado em: 305, 307
- Markou, M. e Singh, S. (2003). Novelty detection: A review - part 1: Statistical approaches. *Signal Processing*, 83(12):2481–2497. Citado em: 261
- Martin, J. (1997). An exact probability metric for decision tree splitting and stopping. *Machine Learning*, 28:257–291. Citado em: 86
- Martínez-Meyer, E., Peterson, A. T., Servín, J. I. e Kiff, L. F. (2006). Ecological niche modelling and prioritizing areas for species reintroductions. *Oryx*, 40(4):411–418. Citado em: 323
- Martínez-Muñoz, G. e Suárez, A. (2006). Pruning in ordered bagging ensembles. In: Cohen, W. e Moore, A. (Ed.) *Machine Learning, Proceedings of the 23th Int. Conference*. OmniPress. Citado em: 146
- Masulli, F., Pardo, M., Sberveglieri, G. e Valentini, G. (2002). Boosting and classification of electronic nose data. In: *MCS '02: Proceedings of the Third International Workshop on Multiple Classifier Systems*, p. 262–271, London, UK. Springer-Verlag. Citado em: 319
- Matsubara, E. T. (2008). *Relações entre Ranking, Análise ROC e Calibração em Aprendizado de Máquina*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos-SP. Citado em: 166
- Mattison, R. (1998). *AnswerTree Algorithm User's Guide*. SPSS Inc. USA. Citado em: 83
- Mayoraz, E. e Alpaydim, E. (1998). Support vector machines for multi-class classification. Research Report IDIAP-RR-98-06, Dalle Molle Institute for Perceptual Artificial Intelligence. Citado em: 287
- Mayoraz, E. e Moreira, M. (1996). On the decomposition of polychotomies into dichotomies. Research Report 96-08, IDIAP, Dalle Molle Institute for Perceptive Artificial Intelligence, Martigny, Valais, Switzerland. Citado em: 280
- McCauley, J. D., Thane, B. R. e Whittaker, A. D. (1994). Fat estimation in beef ultrasound images using texture and adaptive logic networks. *Transactions of ASAE*, 37:997–102. Citado em: 318
- McCulloch, W. S. e Pitts, W. (1943). A logical calculus of the ideas in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133. Citado em: 108, 110

- McIntyre, R. M. e Blashfield, R. K. (1980). A nearest-centroid technique for evaluating the minimum-variance clustering procedure. *Multivariate Behavioral Research*, 15:225–238. Citado em: 239, 245
- Meila, M. (2007). Comparing clusterings - an information based distance. *Journal of Multivariate Analysis*, 98:873–895. Citado em: 252
- Mercer, J. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, A* 209:415–446. Citado em: 132
- Merz, C. J. (1998). *Classification and Regression by Combining Models*. Tese de Doutorado, University of California, Irvine. <http://www.ics.uci.edu/~cmerz/thesis.ps> [Acessado em 19/Jul./2002]. Citado em: 143, 220
- Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 3rd ed. Citado em: 313
- Michalski, R. S., Mozetic, I., Hong, J. e Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In: *Proceedings of the Fifth National Conference on Artificial Intelligence*, p. 1041–1045. Morgan Kaufmann. Citado em: 100, 102
- Michie, D., Spiegelhalter, D. J. e Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA. Citado em: 137, 156, 162, 272
- Milligan, G. e Cooper, M. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179. Citado em: 245
- Milligan, G. e Cooper, M. (1986). A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, 21:441–458. Citado em: 253, 254
- Milligan, G., Soon, T. e Sokol, L. (1983). The effect of cluster size, dimensionality, and the number of clusters on recovery of true cluster structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(1):40–47. Citado em: 254
- Milligan, G. W. (1996). *Clustering and Classification*, Capítulo Clustering validation: results and implications for applied analyses. World Scientific Publ. Citado em: 245
- Millonas, M. M. (1994). Swarms, phase transitions and collective intelligence. In: Langton, C. (Ed.) *Artificial Life III*. Addison-Wesley. Citado em: 306
- Mingers, J. (1989a). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243. Citado em: 94
- Mingers, J. (1989b). An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342. Citado em: 90, 91
- Minsky, M. e Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Massachusetts. Citado em: 108
- Mirkin, B. (2011). *Core Concepts in Data Analysis: Summarization, Correlation and Visualization*. Springer. Citado em: 178
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, 1. ed. Citado em: 3, 5, 13, 56, 63, 70, 73, 146, 179, 313, 341
- Müller, K. R., Mika, S., Rätsch, G., Tsuda, K. e Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201. Citado em: 123, 124, 126, 131
- Monard, M. C. e Baranauskas, J. A. (2003). Conceitos de aprendizado de máquina. In: Rezende (2003), p. 89–114. Citado em: 5, 159, 160, 162, 164

- Monti, S., Tamayo, P., Mesirov, J. e Golub, T. (2003). Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52(1-2):91–118. Citado em: 221, 222, 224
- Moreira, M. (2000). *The Use of Boolean Concepts in General Classification Contexts*. Tese de Doutorado, Ecole Polytechnique Federale de Lausanne. Citado em: 280
- Moreira, M. e Mayoraz, E. (1997). Improved pairwise coupling classification with correcting classifiers. Research report IDIAP-RR 97-09, IDIAP, Dalle Molle Institute of Perceptual Artificial Intelligence, Martigny, Switzerland. Citado em: 278
- Morey, L. e Agresti, A. (1984). The measurement of classification agreement: An adjustment to the Rand statistic for chance agreement. *Educational and Psychological Measurement*, 44:33–37. Citado em: 253
- Morey, L. C., Blashfield, R. K. e Skinner, H. A. (1983). A comparison of cluster analysis techniques within a sequential validation framework. *Multivariate Behavioral Research*, 18:309–329. Citado em: 239, 245
- Nadeau, C. e Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, 52(3):239–281. Citado em: 169
- Nadler, M. e Smith, E. P. (1993). *Pattern Recognition Engineering*. Wiley. Citado em: 202
- Nagesh, H., Goil, S. e Choudhary, A. (2001a). Adaptive grids for clustering massive data sets. In: *SIAM Conference on Data Mining*. Citado em: 217
- Nagesh, H., Goil, S. e Choudhary, A. (2001b). *Data Mining for Scientific and Engineering Applications*, Capítulo Parallel algorithms for clustering high-dimensional large-scale datasets, p. 335–356. Kluwer Academic Publishers. Citado em: 217
- Nemenyi, P. B. (1963). *Distribution-free Multiple Comparisons*. Tese de Doutorado, Princeton University. Citado em: 171
- Ng, A. Y., Jordan, M. I. e Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In: *Advances in Neural Information Processing Systems*, vol. 14, p. 849–856. MIT Press. Citado em: 229
- Ng, R. T. e Han, J. (1994). Efficient and effective clustering methods for spatial data mining. In: *Proc. of 20th Int. Conference on Very Large Databases*, p. 144–155, Santiago, Chile. Citado em: 213
- Noble, W. S. (2004). Support vector machine applications in computational biology. In: Schölkopf, B., Tsuda, K. e Vert, J.-P. (Ed.) *Kernel Methods in Computational Biology*, p. 71–92. MIT Press. Citado em: 122
- Notredame, C. e Higgins, D. G. (1996). Saga: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–24. Citado em: 321
- Nuts, R. e Rousseeuw, P. (1996). Computing depth countours of bivariate point clouds. *Journal of Computational Statistics and Data Analysis*, 23:153–168. Citado em: 40
- Oleskovicz, M., Coury, D. V. e de Carvalho, A. C. P. L. F. (1998). Artificial neural network applied to power system protection. In: *V Simpósio Brasileiro de Redes Neurais*, p. 247–252. Citado em: 326
- Ortega, J. (1995). Exploiting multiple existing models and learning algorithms. In: *AAAI 96 - Workshop in Induction of Multiple Learning Models*. Citado em: 141
- Ortega-Huerta, M. A. e Peterson, A. T. (2004). Modelling spatial patterns of biodiversity for conservation prioritization in northeastern Mexico. *Diversity and Distributions*, 10:39–54. Citado em: 323
- Osman, I. e Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63:511–623. 10.1007/BF02125421. Citado em: 305

- Pagallo, G. e Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99. Citado em: 96
- Pakhira, M. K., Bandyopadhyay, S. e Maulik, U. (2004). Validity index for crisp and fuzzy clusters. *Pattern Recognition*, 37(3):487–501. Citado em: 243, 245
- Pal, N. R. e Bezdek, J. C. (1995). On cluster validity for fuzzy c-means model. *IEEE Transactions on Fuzzy Systems*, 3(3):370–379. Citado em: 245
- Park, Y.-J. e Song, M.-S. (1998). A genetic algorithm for clustering problems. In: *Proceedings of the Third Annual Conference on Genetic Programming*, p. 568–575, San Francisco, CA, USA. Morgan Kaufmann Publisher. Citado em: 232
- Passerini, A., Pontil, M. e Frasconi, P. (2004). New results on error correcting output codes of kernel machines. *IEEE Transactions on Neural Networks*, 15:45–54. Citado em: 286, 287
- Patil, B. M., Joshi, R. C. e Toshniwal, D. (2010). Hybrid prediction model for type-2 diabetic patients. *Expert Syst Appl.*, 37(12):8102–8108. Citado em: 335
- Pau, L. e Gotzsche, T. (1992). Explanation facility for neural networks. *Journal of Intelligent and Robotic Systems*, 5:193–206. Citado em: 122
- Pavlidis, P. e Grundy, W. N. (1999). Combining microarray expression data and phylogenetic profiles to learn functional categories using support vector machines. p. 44–59. Routledge. Citado em: 292
- Pazzani, M. (1996). Constructive induction of Cartesian product attributes. In: *Proc. of the Conference ISIS96: Information, Statistics and Induction in Science*, p. 66–77. World Scientific. Citado em: 78
- Pearce, J. e Lindenmayer, D. B. (1998). Bioclimatic analysis to enhance reintroduction biology of the endangered helmeted honeyeater (*Lichenostomus melanops cassidix*) in Southeastern Australia. *Restoration Ecology*, 6:238–243. Citado em: 323
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc. Citado em: 78
- Pearlmutter, B. (1992). Gradient descent: second order momentum and saturation error. In: Moody, J. E., Hanson, S. e Lippmann, R. (Ed.) *Advances in Neural Information Processing Systems 2*, p. 887–894. Morgan Kaufmann. Citado em: 119
- Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572. Citado em: 47
- Pearson, R. G. (2007). Species' distribution modeling conservation educators and practitioners. *Synthesis*, American Museum of Natural History. Citado em: 324
- Pearson, R. G., Dawson, T. P., Berry, P. M. e Harrison, P. A. (2002). Species: A spatial evaluation of climate impact on the envelope of species. *Ecological Modelling*, 154:289–300. Citado em: 323
- Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. e Hsu, M. (2001). Prefixspan: Mining sequential patterns by prefix-projected growth. In: *Proceedings of the 17th International Conference on Data Engineering*, p. 215–224, Heidelberg, Germany. Citado em: 185
- Peng, Y., Flach, P. A., Soares, C. e Brazdil, P. (2002). Improved dataset characterisation for meta-learning. In: *DS '02: Proceedings of the 5th International Conference on Discovery Science*, p. 141–152, London, UK. Springer-Verlag. Citado em: 273
- Pestana, D. e Velosa, S. (2002). *Introdução à probabilidade e à estatística*. Fundação Calouste Gulbenkian. Citado em: 70

- Peterson, A. T. (2003). Predicting the geography of species' invasions via ecological niche modeling. *Quarterly Review of Biology*, 78:419–433. Citado em: 323
- Peterson, A. T., Benz, B. W. e Papes, M. (2007). Highly pathogenic h5n1 avian influenza: Entry pathways into North America via bird migration. *PLoS ONE*, 2:e261. Citado em: 323
- Peterson, A. T., Lash, R. R., Carroll, C. R. e Johnson, K. M. (2006). Geographic potential for outbreaks of Marburg hemorrhagic fever. *American Journal of Tropical Medicine & Hygiene*, 75:9–15. Citado em: 323
- Peterson, A. T., Ortega-Huerta, M. A., Bartley, J., Sanchez-Cordero, V., Buddemeier, R. H. e Stockwell, D. R. B. (2002a). Future projections for Mexican faunas under global climate change scenarios. *Nature*, 416:626–629. Citado em: 323
- Peterson, A. T., Papes, M. e Kluza, D. A. (2003). Predicting the potential invasive distributions of four alien plant species in North America. *Weed Science*, 51(6):863–868. Citado em: 323
- Peterson, A. T., Sanchez-Cordero, V., Beard, C. B. e Ramsey, J. M. (2002b). Ecologic niche modeling and potential reservoirs for Chagas disease, Mexico. *Emerging Infectious Diseases*, 8:662–667. Citado em: 323
- Peterson, A. T., Soberón, J. e Sanchez-Cordero, V. (1999). Conservatism of ecological niches in evolutionary time. *Science*, 285:1265–1267. Citado em: 323
- Peterson, A. T. e Vieglais, D. A. (2001). Predicting species invasions using ecological niche modeling: New approaches from bioinformatics attack a pressing problem. *BioScience*, 51(5):363–371. Citado em: 323
- Pfahringer, B., Bensusan, H. e Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In: *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'2000*, p. 743–750. Morgan Kaufmann. Citado em: 273
- Phetkaew, T., Kijsirikul, B. e Rivepiboon, W. (2003). Reordering adaptive directed acyclic graphs: an improved algorithm for multiclass support vector machines. In: *Proceedings of the International Conference on Neural Networks*, p. 1605–1610. Citado em: 284
- Pimenta, E., Gama, J. e Carvalho, A. C. P. L. F. (2007). Pursuing the best ecoc dimension for multiclass problems. In: *Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2007)*, p. 622–627. AAAI Press. Citado em: 282
- Platt, J. (2000). Probabilities for sv machines. In: *Advances in Large Margin Classifiers*, p. 61–74. Citado em: 136
- Platt, J. C., Cristianini, N. e Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In: Solla, S. A., Leen, T. K. e Müller, K.-R. (Ed.) *Advances in Neural Information Processing Systems*, vol. 12, p. 547–553. The MIT Press. Citado em: 283
- Plaza, E. e Arcos, J. (2002). Constructive adaptation. In: Craw, S. e Preece, A. (Ed.) *6th European Conference on Case-Based Reasoning, Aberdeen, Scotland, UK*, p. 306–320. Citado em: 69
- Policastro, C. A., de Carvalho, A. C. P. L. F. e Delbem, A. C. B. (2004). A hybrid case based reasoning approach for monitoring water quality. In: *IEA/AIE*, p. 492–501. Citado em: 324
- Policastro, C., d. C. A., Delbem, A. C. B., Mattoso, L. H. C., Minatti, E., Ferreira, E. J., Borato, C. E. e Zanus, M. C. (2007). A hybrid case based reasoning approach for wine classification. In: *ISDA '07: Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*, p. 395–400, Washington, DC, USA. IEEE Computer Society. Citado em: 319
- Pontil, M. e Verri, A. (1998). Support vector machines for 3-D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646. Citado em: 284

- Prati, R. (2006). *Novas Abordagens em Aprendizado de Máquina para a Geração de Regras, Classes Desbalanceadas e Ordenação de Casos*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos-SP. Citado em: 166, 167
- Prati, R. C. e Flach, P. A. (2005). ROCCER: an algorithm for rule learning based on ROC analysis. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, p. 823–828. Citado em: 165
- Prentzas, J. e Hatzilygeroudis, I. (2002). Integrating hybrid rule-based with case-based reasoning. In: Craw, S. e Preece, A. (Ed.) *6th European Conference on Case-Based Reasoning*, p. 336–349. Springer Verlag. Citado em: 69
- Provost, F. e Domingos, P. (2001). *Well-trained PETs: Improving Probability Estimation Trees*. Relatório técnico, CeDER Working paper IS-00-04, Stern School of Business, New York University. Citado em: 167
- Prudêncio, R. B. C. e Ludermir, T. B. (2006). A machine learning approach to define weights for linear combination of forecasts. In: *ICANN* (1), p. 274–283. Citado em: 275, 276
- Prudêncio, R. B. C. e Ludermir, T. B. (2004). Meta-learning approaches to selecting time series models. *Neurocomputing*, 61:121–137. Citado em: 270, 275, 276
- Pyle, D. (1999). *Data Preparation for Data Mining*. Morgan Kaufmann. Citado em: 16
- Qi, Y., Klein-Seetharaman, J., Bar-Joseph, Z., Qi, Y. e Bar-Joseph, Z. (2005). Random forest similarity for protein-protein interaction prediction. *Pacific Symposium on Biocomputing*, 2005:531–542. Citado em: 322
- Qian, Y. e Suen, C. (2000). Clustering combination method. In: *Proceedings of the International Conference on Pattern Recognition (ICPR'2000)*, vol. II, p. 736–739. Citado em: 221
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Mateo, CA, USA. Citado em: 56, 83, 92, 93, 95, 98, 99, 293, 302
- Quinlan, R. (1979). Discovering rules by induction from large collections of examples. In: Michie, D. (Ed.) *Expert Systems in the Microelectronic Age*, p. 168–201. Edinburgh University Press. Citado em: 83
- Quinlan, R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106. Citado em: 95
- Quinlan, R. (1988). Simplifying decision trees. In: Gaines, B. e Boose, J. (Ed.) *Knowledge Acquisition for Knowledge Based Systems*. Academic Press. Citado em: 92, 93
- Quinlan, R. (1995). Mdl and categorical theories (continued). In: Prieditis, A. e Russel, S. (Ed.) *Machine Learning Proc. of 12th International Conference*. Morgan Kaufmann. Citado em: 99
- Quinlan, R. (1996). Bagging, boosting and C4.5. In: *Proc. 13th American Association for Artificial Intelligence*. AAAI Press. Citado em: 157
- Quinlan, R. (1998). Data mining tools See5 and C5.0. Relatório técnico, RuleQuest Research. Citado em: 105
- Quinlan, R. J. (1992). Learning with continuous classes. In: *5th Australian Joint Conference on Artificial Intelligence*, p. 343–348. Citado em: 104, 105
- Rezende, S. O. (2003). *Sistemas Inteligentes - Fundamentos e Aplicações*. Editora Manole. Citado em: 350, 362
- Riedmiller, M. e Braun, H. (1994). *RPROP – Description and Implementation Details*. Relatório técnico, University of Karlsruhe. Citado em: 119

- Rifkin, R. e Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:1533–7928. Citado em: 278, 280, 282
- Rivest, R. L. (1987). Learning Decision Lists. *Machine Learning*, 2:229–246. Citado em: 85, 98
- Rodrigues, P., Gama, J. e Pedroso, J. (2006). Odac: Hierarchical clustering of time series data streams. In: Ghosh, J., Lambert, D., Skillicorn, D. e Srivastava, J. (Ed.) *Proceedings of the Sixth SIAM International Conference on Data Mining*, p. 499–503, Bethesda, Maryland, USA. Society for Industrial and Applied Mathematics. Citado em: 261, 264
- Rodrigues, P. P. e Gama, J. (2009). A system for analysis and prediction of electricity-load streams. *Intelligent Data Analysis*, 13(3):477–496. Citado em: 328
- Rosa, M. d. O., Pereira, J. C. e Carvalho, A. C. P. L. F. (1998). Evaluation of neural classifiers using statistic methods for identification of laryngeal pathologies. In: *SBRN '98: Proceedings of the Vth Brazilian Symposium on Neural Networks*, p. 220, Washington, DC, USA. IEEE Computer Society. Citado em: 336
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Reviews*, 65:386–408. Citado em: 108, 113
- Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65. Citado em: 243, 244
- Rousseeuw, P. J., Ruts, I. e Tukey, J. W. (1999). The bagplot: a bivariate boxplot. *The American Statistician*, 53(4):382–387. Citado em: 26
- Rumelhart, D. E., Hinton, G. E. e Willians, R. J. (1986). Learning internal representations by error propagation. In: Rumelhart, D. E. e McClelland, J. L. (Ed.) *Parallel Distributed Processing: Foundations*, vol. 1, p. 318–362. MIT Press, Cambridge, MA. Citado em: 117
- Rumelhart, D. E. e McClelland, J. L. (1986). *Parallel Distributed Processing*, vol. 1: Foundations. The MIT Press. Citado em: 119
- Sahami, M. (1996). Learning limited dependence Bayesian classifiers. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, p. 335–338. AAAI Press. Citado em: 80, 81, 82
- Salzberg, S. L. (1997). A method for identifying splice sites and translational start sites in eukaryotic mRNA. *Computer Applications in Biosciences*, 13(4):365–376. Citado em: 169, 171
- Saridis, G. (1983). Parameter estimation: Principles and problems. *Automatic Control, IEEE Transactions on*, 28(5):634–635. Citado em: 293
- Savicky, P. e Fürnkranz, J. (2003). Combining pairwise classifiers with stacking. In: Berthold, M. R., Lenz, H.-J., Bradley, E., Kruse, R. e Borgelt, C. (Ed.) *Advances in Intelligent Data Analysis V, 5th International Symposium on Intelligent Data Analysis, IDA 2003*, p. 219–229. Citado em: 287
- Schaffer, C. (1993). Overfitting avoidance as bias. *Machine Learning*, 10:153–178. Citado em: 94
- Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5:197–227. Citado em: 146
- Schapire, R. E. e Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336. Citado em: 293
- Schapire, R. E. e Singer, Y. (2000). Boostexter: a boosting-based system for text categorization. In: *Machine Learning*, p. 135–168. Citado em: 293, 295
- Schölkopf, B., Guyon, I. e Weston, J. (2003). Statistical learning and kernel methods in bioinformatics. In: Frasconi, P. e Shamir, R. (Ed.) *Artificial Intelligence and Heuristic Methods in Bioinformatics*, p. 1–21. IOS Press. Citado em: 122

- Seewald, A. e Fürnkranz, J. (2001). An evaluation of grading classifiers. In: Hoffmann, F., Hand, D., Adams, N. e Guimaraes, G. (Ed.) *Advances in Intelligent Data Analysis - IDA01*, p. 115–124. LNCS 2189 Springer Verlag, Berlin. Citado em: 142
- Seidman, C. (2001). *Data Mining with Microsoft SQL Server 2000 Technical Reference*. Microsoft Press. Citado em: 83
- Setubal, J. e Meidanis, J. (1997). *Introduction to Computational Molecular Biology*. PWS Publishing Company. Citado em: 320
- Shamir, R., Kimmel, G., Farkash, A., Klinger, N. e Peles, R. (2001). Algorithms for molecular biology - lecture notes. Disponível em: <http://www.math.tau.ac.il/~rshamir/algmb/01/algmb01.html>. Citado em: 321
- Shamir, R. e Sharan, R. (2002). *Current Topics in Computational Biology*, Capítulo Algorithmic approaches to clustering gene expression data, p. 269–299. MIT Press. Citado em: 216
- Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423. Citado em: 281
- Sharan, R. e Shamir, R. (2000). CLICK: A clustering algorithm with applications to gene expression analysis. In: *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB'2000)*, p. 307–316. Citado em: 216
- Sharkey, A. J. C. (1999). *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems (Perspectives in Neural Computing)*. Springer Verlag. Citado em: 220
- Shatkay, H. e Feldman, R. (2003). Mining the biomedical literature in the genomic era: An overview. *Journal of Computational Biology (JCB)*, 10(6):821–856. Citado em: 332
- Shawe-Taylor, J., Barlett, P. L., Williamson, R. C. e Anthony, M. (1998). Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940. Citado em: 125
- Sheikholeslami, G., Chatterjee, S. e Zhang, A. (1998). WaveCluster: A multi-resolution clustering approach for very large spatial databases. In: *Proceedings of the 24th International Conference on Very Large Data Bases*, p. 428–439, New York City. ACM Press. Citado em: 214, 261
- Shen, X., Boutell, M., Luo, J. e Brown, C. (2004). Multi-label machine learning and its application to semantic scene classification. In: *International Symposium on Electronic Imaging*, San Jose, CA. Citado em: 288, 295
- Shi, Y. e Eberhart, R. C. (1998). A modified particle swarm optimizer. In: *Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence*, p. 69–73. Citado em: 309
- Shoemaker, B. A. e Panchenko, A. R. (2007a). Deciphering protein-protein interactions. part i. experimental techniques and databases. *PLoS Comput Biol*, 3(3):e42. Citado em: 322
- Shoemaker, B. A. e Panchenko, A. R. (2007b). Deciphering protein-protein interactions. part ii. computational methods to predict protein and domain interaction partners. *PLoS Comput Biol*, 3(4):e43. Citado em: 322
- Sigaud, O. e Wilson, S. W. (2007). Learning classifier systems: a survey. *Soft Computing*, 11:1065–1078. Citado em: 313
- Silla, C. N. e Freitas, A. A. (2009). A global-model naive Bayes approach to the hierarchical prediction of protein functions. In: *Proceedings of the 9th IEEE International Conference on Data Mining*, p. 992–997. Citado em: 302

- Silla, C. N. e Freitas, A. A. (2010). A survey of hierarchical classification across different application domains. *Journal Data Mining and Knowledge Discovery*. Citado em: 297, 299, 301, 302, 303
- Sinclair, S. J., White, M. D. e Newell, G. R. (2010). How useful are species distribution models for managing biodiversity under future climates? *Ecology and Society*, 15(1). Citado em: 323
- Skalak, D. (1997). *Prototype Selection For Composite Nearest Neighbor Classifiers*. Tese de Doutorado, University of Massachusetts Amherst. Citado em: 149, 151, 152
- Smith-Miles, K. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1). Citado em: 270
- Smola, A. e Schölkopf, B. (1998). *A Tutorial on Support Vector Regression*. Relatório Técnico NC2-TR-1998-030, NeuroCOLT2. Citado em: 128, 133
- Smola, A. J., Bartlett, P., Schölkopf, B. e Schuurmans, D. (1999). Introduction to large margin classifiers. In: Smola, A. J., Bartlett, P., Schölkopf, B. e Schuurmans, D. (Ed.) *Advances in Large Margin Classifiers*, p. 1–28. MIT Press. Citado em: 123, 124, 125
- Smola, A. J. e Schölkopf, B. (2002). *Learning with Kernels*. The MIT Press, Cambridge, MA. Citado em: 123, 124, 128
- Soares, C. (2004). *Learning Rankings of Learning Algorithms: recommendation of algorithms with meta-learning*. Tese de Doutorado, Universidade do Porto, Porto, Portugal. Citado em: 271, 272, 273, 274
- Soares, C., Petrank, J. e Brazdil, P. (2001). Sampling-based relative landmarks: Systematically test-driving algorithms before choosing. In: P., B. e A., J. (Ed.) *Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving, Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA 2001)*, vol. 2258 of *Lecture Notes in Computer Science*, p. 88–95, Porto, Portugal. Springer. Citado em: 274
- Sobolewski, R. (2009). Machine Learning for Automated Robot Navigation in Rough Terrain. Dissertação de Mestrado, Computing Laboratory, University of Oxford. Citado em: 334
- Socha, K. (2004). Aco for continuous and mixed-variable optimization. In: *ANTS Workshop*, p. 25–36. Citado em: 307, 308
- Socha, K. e Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155–1173. Citado em: 308
- Song, M. e Rajasekaran, S. (2010). A greedy algorithm for gene selection based on svm and correlation. *Int J Bioinform Res Appl.*, 6(3):296–307. Citado em: 320
- Sousa, E., Traina, A., Traina, J. C. e Faloutsos, C. (2007). Evaluating the intrinsic dimension of evolving data streams. *New Generation Computing*, 25(1):33–60. Citado em: 261
- Souto, M. C. P., Lorena, A. C., Delbem, A. C. B. e Carvalho, A. C. P. L. F. (2003). *Técnicas de Aprendizado de Máquina para Problemas de Biologia Molecular*, p. 103–152. Minicursos de Inteligência Artificial, Jornada de Atualização Científica em Inteligência Artificial, XXIII Congresso da Sociedade Brasileira de Computação. Citado em: 178, 322
- Sovat, R. (2002). *Uma Abordagem Híbrida Baseada em Casos e Redes Neurais. Uma aplicação: escolha e configuração de modelos de redes neurais*. Tese de Doutorado, Universidade de São Paulo. Citado em: 272
- Spackman, K. A. (1989). Signal detection theory: valuable tools for evaluating inductive learning. In: *Proceedings of the 6th International Workshop on Machine Learning*, p. 160–163. Citado em: 165

- Spinoza, E., Gama, J. e Carvalho, A. (2008). Cluster-based novel concept detection in data streams applied to intrusion detection in computer networks. In: *Proceedings of the 2008 ACM Symposium on Applied Computing*, p. 976–980. ACM Press. Citado em: 261
- Stockwell, D. R. B. e Peters, D. P. (1999). The GARP modelling system: Problems and solutions to automated spatial prediction. *International Journal of Geographic Information Systems*, 13:143–158. Citado em: 325
- Strehl, A. e Ghosh, J. (2002). Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research (JMLR)*, 3:583–617. Citado em: 219, 221, 222, 223, 224, 225, 226, 227, 228, 230, 233
- Su, C.-Y., Lo, A., Lin, C.-C., Chang, F. e Hsu, W.-L. (2005). A novel approach for prediction of multi-labeled protein subcellular localization for prokaryotic bacteria. In: *CSBW '05: Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference - Workshops*, p. 79–82, Washington, DC, USA. IEEE Computer Society. Citado em: 292
- Sun, A. e Lim, E.-P. (2001). Hierarchical text classification and evaluation. In: *Proceedings of the 1st IEEE International Conference on Data Mining*, p. 521–528. Citado em: 297, 303
- Takahashi, F. e Abe, S. (2003). Optimizing directed acyclic graph support vector machines. In: *Proceedings of Artificial Neural Networks in Pattern Recognition*, p. 166–170. Citado em: 284
- Tapia, E., Bulacio, P. e Angelone, L. (2010). Recursive ecoc classification. *Pattern Recognition Letters*, 31(3):210–215. Citado em: 282
- Thibos, L. e Hong, X. (1999). Clinical applications of the shack-hartmann aberrometer. *Optom Vis Sci*, 76(12):817–25. Citado em: 336
- Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., Jong, K. D., Dzeroski, S., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R., Mitchell, T., Pachowicz, P., Roger, B., Vafaie, H., de Velde, W. V., Wenzel, W., Wnek, J. e Zhan, J. (1991). The MONK's problems: A Performance Comparison of Different Learning Algorithms. Relatório Técnico CMU-CS-91-197, Carnegie Mellon University. Citado em: 153
- Thuiller, W., Richardson, D. M., Pysek, P., Whittaker, R. J., Brotons, L. e Lavorel, S. (2005). Niche-based modeling as a tool for predicting the global risk of alien plant invasions. *Global Change Biology*, 11:2234–2250. Citado em: 323
- Tibshirani, R. e Hastie, T. (2007). Margin trees for high-dimensional classification. *Journal of Machine Learning Research*, 8:637–652. Citado em: 285
- Tibshirani, R., Walther, G., Botstein, D. e Brown, P. (2001a). *Cluster Validation by Prediction Strength*. Relatório técnico, Department of Statistics, Stanford University. <http://www-stat.stanford.edu/~tibs/ftp/predstr.ps> [Acessado em 08/Jul./2011]. Citado em: 239, 245
- Tibshirani, R., Walther, G. e Hastie, T. (2001b). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423. Citado em: 249
- Tickle, A., Orlowski, M. e Diederich, J. (1995). DEDEC: decision detection by rule extraction from neural networks. Relatório técnico, Queensland University of Technology. Citado em: 122
- Ting, K. e Witten, I. (1997). Stacked generalization: when does it work? In: *Proc. International Joint Conference on Artificial Intelligence*. Morgan Kaufmann. Citado em: 151, 155, 157
- Topchy, A., Jain, A. e Punch, W. (2003). Combining multiple weak clusterings. In: *Proceedings of the IEEE International Conference on Data Mining (ICDM'2003)*, p. 331–338, Melbourne, Florida, USA. Citado em: 219, 220, 221, 222, 223, 224

- Topchy, A., Jain, A. e Punch, W. (2004). A mixture model for clustering ensembles. In: *Proceedings of the SIAM International Conference on Data Mining (SDM'2004)*, p. 331–338, Lake Buena Vista, Florida, USA. Citado em: 219, 221, 222, 223, 224
- Topchy, A., Jain, A. K. e Punch, W. (2005). Clustering ensembles: models of consensus and weak partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1866–1881. Citado em: 225
- Toussaint, G. T. (1974). Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, 20(4):472–479. Citado em: 160
- Towell, G. e Shavlik, J. W. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101. Citado em: 122
- Trambaiolli, L. R., Lorena, A. C., Fraga, F. J. e Anghinah, R. (2009). Uso de aprendizado de máquina no auxílio ao diagnóstico de Alzheimer. *REIC - Revista Eletrônica de Iniciação Científica*, IV(III):14 p. Citado em: 336
- Tsoumakas, G. e Katakis, I. (2007). Multi label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13. Citado em: 288, 294, 295
- Tsoumakas, G. e Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In: *Proceedings of the 18th European Conference on Machine Learning (ECML 2007)*. Citado em: 290
- Tumer, K. e Ghosh, J. (1995). Classifier combining: analytical results and implications. In: *AAAI 96 - Workshop in Induction of Multiple Learning Models*. Citado em: 138
- Tumer, K. e Ghosh, J. (1996a). Error correlation and error reduction in ensemble classifiers. *Connection Science, Special issue on combining artificial neural networks: ensemble approaches*, 8, N.3-4:385–404. Citado em: 138
- Tumer, K. e Ghosh, J. (1996b). *Theoretical Foundations of Linear and Order Statistics Combiners for Neural Pattern Classifiers*. Relatório Técnico TR-95-02-98, University of Texas at Austin, The Computer and Vision Research Center. Citado em: 138
- Turban, E. e Frenzel, L. E. (1992). *Expert Systems and Applied Artificial Intelligence*. Prentice Hall College Div. Citado em: 333
- Unger, R. (2004). The genetic algorithm approach to protein structure prediction. *Structure and Bonding*, 110:153–175. Citado em: 321
- Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York. Citado em: 107, 122, 124, 133
- Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley and Sons. Citado em: 124
- Vapnik, V. N. e Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):283–305. Citado em: 122
- Vendramin, L., Campello, R. J. G. B. e Hruschka, E. R. (2010). Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining*, 3:209–235. Citado em: 245
- Vilalta, R. e Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95. Citado em: 270
- Vilalta, R., Giraud-Carrier, C. e Brazdil, P. (2005). *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers*, Capítulo Meta-Learning: Concepts and techniques, p. 1–17. Kluwer Academic Publishers. Citado em: 272, 273

- Vural, V. e Dy, J. G. (2004). A hierarchical method for multi-class support vector machines. In: *Proceedings of the 21st International Conference on Machine Learning*, p. 831–838. Citado em: 285
- Wald, A. (1947). *Sequential Analysis*. John Wiley and Sons, Inc. Citado em: 269
- Wang, W., Yang, J. e Muntz, R. (1997). STING: A statistical information grid approach to spatial data mining. In: *Proceedings of the 23rd International Conference on Very Large Databases*, p. 186–195, Athens, Greece. Citado em: 217
- Wang, Y., Tetko, I. V., Hall, M. A., Frank, E., Facius, A., Mayer, K. F. e Mewes, H. W. (2005). Gene selection from microarray data for cancer classification - a machine learning approach. *Comput Biol Chem.*, 29(1):37–46. Citado em: 320
- Wang, Y. e Witten, I. H. (1997). Induction of model trees for predicting continuous classes. In: *9th European Conference on Machine Learning*. Citado em: 104
- Webb, G., Boughton, J. e Wang, Z. (2005). Not so naïve Bayes: Aggregating one-dependence estimators. *Machine Learning*, 58(1):5–24. Citado em: 82
- Weingessel, A., Dimitriadou, E. e Hornik, K. (2003). An ensemble method for clustering. In: *Distributed Statistical Computing (DSC'2003)*, Wien, Austria. <http://www.ci.tuwien.ac.at/Conferences/DSC-2003/> [Acessado em: 19/Jan/2004]. Citado em: 221, 223, 224
- Weiss, S. e Indurkhy, N. (1998). *Predictive Data Mining, a Practical Guide*. Morgan Kaufmann Publishers. Citado em: 98
- Widrow, B. e Hoff, M. (1960). Adaptative switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention*. Citado em: 115
- Wilcoxon, F. (1943). Individual comparisons by ranking methods. *Biometrics*, 1:80–83. Citado em: 169
- Williams, N. S. G., Hahs, A. K. e Morgan, J. W. (2008). A dispersal-constrained habitat suitability model for predicting invasion of alpine vegetation. *Ecological Applications*, 18:347–359. Citado em: 323
- Wilson, D. R. e Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34. Citado em: 202
- Wilson, R. J. e Watkins, J. J. (1990). *Graphs: An Introductory Approach - A First Course in Discrete Mathematics*. John Wiley & Sons. Citado em: 232
- Windeatt, T. e Ghaderi, R. (2003). Coding and decoding strategies for multi-class learning problems. *Information Fusion*, 4(1):11–21. Citado em: 286
- Wiratunga, N., Craw, S. e Rowe, R. (2002). Learning to adapt for case-based design. In: Craw, S. e Preece, A. (Ed.) *6th European Conference on Case-Based Reasoning*, p. 421–435. Springer Verlag. Citado em: 69
- Witten, I., Frank, E. e Hall, M. (2011). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman Publishers Inc., San Francisco, CA, USA, 3 ed. Citado em: 10
- Wnek, J. e Michalski, R. S. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14:139–168. Citado em: 100, 101, 102
- Wolpert, D. (1992). Stacked generalization. *Neural Networks*, 5:241–260. Citado em: 151, 154, 157
- Wolpert, D. e Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82. Citado em: 137

- Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390. Citado em: 270
- Wu, T.-F., Lin, C.-J. e Weng, R. C. (2004). Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005. Citado em: 287
- Wu, W., Guo, Q., de Aguiar, P. F. e Massart, D. L. (1998). The star plot: an alternative display method for multivariate data in the analysis of food and drugs. *J Pharm. Biomed. Anal.*, 17(6-7):1001–13. Citado em: 26
- Xie, X. L. e Beni, G. (1991). A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847. Citado em: 245
- Xu, R. e Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678. Citado em: 203
- Yandell, M. D. e Majoros, W. H. (2002). Genomics and natural language processing. *J Nature Review Genetics*, 3(8):601–610. Citado em: 332
- Yang, M.-S. e Wu, K.-L. (2001). A new validity index for fuzzy clustering. In: *IEEE International Fuzzy Systems Conference*, p. 89–92. Citado em: 245
- Yang, Y., Webb, G. I. e Wu, X. (2005). Discretization methods. In: Maimon, O. e Rokach, L. (Ed.) *The Data Mining and Knowledge Discovery Handbook*, p. 113–130. Springer US. Citado em: 14, 44
- Yeung, K. (2001). *Cluster Analysis of Gene Expression Data*. Tese de Doutorado, University of Washington. Citado em: 244
- Yeung, K. Y., Haynor, D. R. e Ruzzo, W. L. (2001). Validating clustering for gene expression data. *Bioinformatics*, 17(4):309–318. Citado em: 239, 245, 246, 247
- Yu, C. Y., Chou, L. C. e Chang, D. (2010). Predicting protein-protein interactions in unbalanced data using the primary structure of proteins. *BMC Bioinformatics*, 11(1):167. Citado em: 322
- Zadrozny, B. (2001). Reducing multiclass to binary by coupling probability estimates. In: *Proceedings of the 14th Annual Conference Neural Information Processing Systems, Advances in Neural Information Processing Systems*, vol. 14, p. 1041–1048. Citado em: 287
- Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390. Citado em: 185
- Zalewski, W., Lee, H., Caetano, A., Lorena, A., Maletzke, A., Fagundes, J., Saddy, C., Coy, R. e Wu, F. (2008). Evaluation of models for the recognition of handwritten digits in medical forms. In: Bazzan, A., Craven, M. e Martins, N. (Ed.) *Advances in Bioinformatics and Computational Biology*, vol. 5167 of *Lecture Notes in Computer Science*, p. 178–181. Springer Berlin / Heidelberg. Citado em: 336
- Zeng, Y., Tang, J., Garcia-Frias, J. e Gao, G. (2002). An adaptive meta-clustering approach: Combining the information from different clustering results. In: *IEEE Computer Society Bioinformatics Conference (CSB'2002)*, p. 276–287, Stanford, California. Citado em: 203, 204, 221
- Zhang, M.-L. e Zhou, Z.-H. (2005). A k-Nearest Neighbor Based Algorithm for Multi-label Classification. In: *IEEE International Conference on Granular Computing*, vol. 2, p. 718–721. Citado em: 288, 293
- Zhang, T., Ramakrishnan, R. e Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, p. 103–114, Montreal, Canada. Citado em: 209, 261
- Zheng, Z. (1998). Naive Bayesian classifier committees. In: Nedellec, C. e Rouveiro, C. (Ed.) *Machine Learning ECML-98*. Springer Verlag. Citado em: 149

Índice Remissivo

- Acurácia, 159, 164
Adaptação de casos, 68
Adaptive Directed Acyclic Graph, 284
Agrupamento, 191, 202
 critérios de agrupamento, 192
 critérios de validação, 206
Agrupamento multiobjetivo, 220, 231, 234
Algoritmo da cobertura, 99
Algoritmos genéticos, 311
Amostragem, 160
Amostragem aleatória, 161, 162
Análise de agrupamento, 178, 236
Análise de replicação, 245
Análise exploratória, 236
Aprendizado de máquina, 2
Aprendizado em fluxos contínuos de dados, 260
Aprendizado por lotes, 260
Aprendizado por reforço, 7
Apriori, 182
AQ, 102
Aquisição de conhecimento, 2
Area Under ROC Curve, 167
Árvore, 283, 297
Árvore de decisão, 83
Árvore de modelos, 104
Árvore direcionada binária, 284
Árvores de opção, 105
Árvore geradora mínima, 232
Atributo alvo, 4
Atributo de saída, 4
Atributo meta, 4
Atributos, 4, 12
Atributos contínuos, 14
Atributos de entrada, 4
Atributos discretos, 14
Atributos previsores, 4
Avaliação de agrupamentos, 236
Average-link, 210
Back-propagation, 117
Bagplot, 26
Bias, 5, 49
Bioinformática, 320
Biologia Molecular, 320
Boosting, 38
Bootstrap, 161, 163
Borderline, 41
Boxplot, 18
Caixeiro-viajante, 307
Campos, 4
Chernoff, 26
Ciclo de RBC, 67
Classificação hierárquica, 297
Classificação multirrótulo, 288
CLICK, 216
CLIQUE, 217
Cluster, 192
CN2, 101
Códigos de correção de erros, 282
Coeficiente de interesse, 189
Coeficiente geral de similaridade, 202
Colônias de formigas, 306
Combinação de algoritmos de agrupamento, 225
Complete-link, 210
Computação bioinspirada, 313
Computação evolutiva, 310
Computação natural, 305
Conectividade, 242
Conjunto de dados, 4, 10, 12
Correlação de Pearson, 200
Covariância, 24
Critérios de agrupamento, 192
Critérios de validação, 237
Critérios externos, 238
Critérios internos, 237
Critérios relativos, 237
Cross-validation, 162

- CSPA, 225, 226
 Cubist, 105
 Curtose, 22
 Curvas ROC, 165
- DAG, 79, 297
 DAG adaptável, 283
Decision directed acyclic graph, 283
 Decomposição viés-variação, 172
 DENCLUE, 214
 Dendrograma, 211
 Descoberta de conhecimento em bases de dados, 331
 Desvio padrão, 21
Disjunctive Normal Form (DNF), 84
 Dispersão, 20
 Distância de Chebyschev, 200
 Distância de Hamming, 42, 199, 201
 Distância de Manhattan, 199
 Distância euclidiana, 198, 199
 Distância média absoluta, 160
- Early stop*, 119
 Energia, 326
Ensemble multiobjetivo, 233
Ensembles de agrupamentos, 219, 234
 Entropia, 44, 86
 Equilíbrio viés-variação, 173
 Erro do tipo I, 168
 Erro do tipo II, 168
 Erro quadrático médio, 160
Error Correcting Output Codes, 282
 Espaço de atributos, 10
 Espaço de entradas, 10
 Espaço de objetos, 10
 Espalhamento, 20
 Especificidade, 165
 Estatística de teste, 168
 Estatística descritiva, 16
 Estratégia *big-bang*, 302
 Estratégia *top-down*, 302
 Estrutura de *clusters*, 191, 194, 195, 202
 Exemplo, 4
- Falsos negativos, 163
 Falsos positivos, 163
 Figura de mérito, 246
 Filtro, 48
 Finanças, 329
 FND (Forma Normal Disjuntiva), 217
 Forma Normal Disjuntiva (FND), 84
- FP-tree, 187
 Frequência, 17
 Fronte de Pareto, 231, 233
 Função objetivo, 225
 conectividade, 232
 variância intracluster, 232
 Funções baseadas em coassociação, 222
 Funções baseadas em grafo/hipergrafo, 223
 Funções baseadas em informação mútua, 223
 Funções baseadas em votação, 223
- Ganho de informação, 86, 88
 Generalização, 4
 Genoma, 320
 Gini, 90
 Gráfico de pizza, 23
 Grafo direcionado acíclico, 283
- HBF, 225, 229
Heatmap, 27
 HGPA, 225, 226
 Hipótese, 4
 Hipótese alternativa, 168
 Hipótese estatística, 168
 Hipótese nula, 168
 Histograma, 21
Holdout, 161, 162
- Índice de Fowlkes e Mallows, 253
 Índice de validação, 237
 Índice Gap, 249
 Índice Hubert, 253
 Índice Jaccard, 252
 Índice Rand, 252
 Índice Rand corrigido, 253
 Índices Dunn, 243
 Índices estatísticos, 237
 Índices externos, 251
 Índices internos, 249
 Indução, 3
 Informação mútua normalizada, 225
 Inteligência artificial, 2
 Inteligência coletiva, 306
 Inteligência de enxames, 306
 Interpretação de *clusters*, 207
 Intervalares, 15
Itemsets frequentes, 187
Itemsets frequentes fechados, 187
Itemsets frequentes maximais, 187
- Kernel, 132

- k*-médias, 213
Leave-one-out, 162
Lift, 189
Listas de decisão, 98
Localidade, 17

M5, 105
Máquinas de vetores de suporte, 41, 107, 122
Margem, 124
Matriz de códigos, 279
Matriz de confusão, 159
Matriz de covariância, 24
Matriz de similaridade, 197, 248
MCHPF, 234
MCLA, 225, 226
Mean Absolute Distance, 160
Mean Squared Error, 160
Média, 17
Mediana, 17
Medida de similaridade, 202
Medida F, 165
Medidas de dissimilaridade, 198
Medidas de distância, 199
Medidas de similaridade, 198–200
Meta-heurísticas, 306
Métrica de Minkowski, 199
Mineração de dados, 331
Mineração de textos, 331
Mineração web, 332
MOCK, 231
MOCLE, 234
Modelagem da distribuição potencial de espécies, 324
Modelo, 4
Momento, 21
Monte Carlo, 240, 250, 251
MST, 232
Multilayer perceptron, 115
Multirrotulo, 288
Mutação, 232

Neurônio artificial, 108, 110
Níveis de refinamento, 193
Nível de significância, 169
Nominais, 15

Objetos, 4, 10, 12
Obliquidade, 21
One-against-all, 280
OneR, 98

Ordinais, 15
Otimização baseada em bandos de pássaros, 308
Otimização baseada em colônias de formigas, 306
Otimização de função consenso, 225
Otimização multiobjetivo, 219, 231, 274
Outliers, 17
Overfitting, 4

Padrão, 4
Padrões frequentes, 181
 Apriori, 184
 Convicção, 190
 Eclat, 184
 Sumarização, 187
Padrões frequentes fechados, 187
Padrões frequentes maximais, 187
Partição consenso, 220, 222, 225, 234
Partições fuzzy, 245
Percentil, 17
PESA-II, 231
Poda, 91
Pós-poda, 92
Pós-teste de Bonferroni-Dunn, 171
Pós-teste de Nemenyi, 171
Pré-poda, 92
Pré-processamento, 197
Precisão, 164
Problema dos *itemsets* frequentes, 181
Procedimento de teste, 168
Processamento de língua natural, 332
Programação genética, 313

Quartis, 18

Racionais, 16
Receiving Operating Characteristics, 165
Reconhecimento de padrões, 1
Recuperação de casos, 68
Recuperação de informação, 332
Rede adaline, 115
Rede perceptron, 113
Rede perceptron multicamadas, 115
Redes neurais artificiais, 41, 107, 108
Região de rejeição, 168
Registro, 4
Representação de adjacência baseada em lócus, 232
Representação de indivíduos, 232
Representação dos objetos, 197

- Retenção de casos, 68
Reutilização de casos, 68
Revisão de casos, 68
Revocação, 164
Robótica, 333

Saúde, 335
Scatter plot, 25
Seleção de atributos, 30
Sensibilidade, 164
Separação angular, 200
Silhueta, 243
Single-link, 210
Sistema nervoso, 108
Sistemas baseados em conhecimento, 2
Sistemas classificadores, 313
SOM, 216
Standard Deviation Reduction (SDR), 91
Star plot, 26
Support vector machines, 41, 107, 122
Support vector regression, 133
Support vectors, 128

Taxa de acerto, 159
Taxa de acerto total, 164
Taxa de erro, 159
Taxa de erro na classe negativa, 164
Taxa de erro na classe positiva, 164
Taxa de erro total, 164
Taxa de verdadeiros positivos, 164, 165
Técnicas de agrupamento, 178
Teoria de aprendizado estatístico, 122
Teste de Friedman, 169, 171
Teste de hipóteses, 168
Teste *Wilcoxon signed-rank*, 169
Tipo qualitativo, 14
Tipo quantitativo, 14
Todos-contra-todos, 280

Um-contra-todos, 280
Um-contra-um, 280
Underfitting, 4

Validação, 194, 203
Validação cruzada, 161, 162
Validação relativa, 245
Variabilidade, 247
Variância, 20, 24
Variância intracluster, 242
Variáveis, 4
Verdadeiros negativos, 163
Verdadeiros positivos, 163
Vetor de características, 10, 12
Vetores de suporte, 128
Viés, 5, 49
Wrapper, 48