Investigação de estratégias para a geração de
máquinas de vetores de suporte multiclases

Ana Carolina Lorena

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 16.01.2006

Assinatura:_____

Investigação de estratégias para a geração de máquinas de vetores de suporte multiclass

Ana Carolina Lorena

Orientador: *Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP como parte dos requisitos para obtenção do título de Doutor em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Janeiro/2006

A meu noivo e a meus pais

Agradecimentos

Gostaria de agradecer primeiramente a Deus.

Ao meu noivo Dimas, pela ajuda e paciência em diversos momentos, principalmente naqueles em que o nervosismo foi grande.

Aos meus pais, pela ajuda e conselhos.

Ao meu orientador André, pelo auxílio, dedicação e animação constante durante todo o período de Doutorado.

À professora Maria Carolina, por auxiliar a assumir a responsabilidade pela minha orientação e também pelos conselhos que me ajudaram na finalização da Tese.

Aos inúmeros amigos do Laboratório de Inteligência Computacional, pela amizade e pela alegria de poder conviver com eles durante todos esses anos.

Aos funcionários do ICMC, pela eficiência e cordialidade prestadas em diversas ocasiões.

À Fapesp pelo apoio financeiro para a realização deste projeto.

Resumo

Diversos problemas envolvem a classificação de dados em categorias, também denominadas classes. A partir de um conjunto de dados cujas classes são conhecidas, algoritmos de Aprendizado de Máquina (AM) podem ser utilizados na indução de um classificador capaz de prever a classe de novos dados do mesmo domínio, realizando assim a discriminação desejada. Dentre as diversas técnicas de AM utilizadas em problemas de classificação, as Máquinas de Vetores de Suporte (*Support Vector Machines* - SVMs) se destacam por sua boa capacidade de generalização. Elas são originalmente concebidas para a solução de problemas com apenas duas classes, também denominados binários. Entretanto, diversos problemas requerem a discriminação dos dados em mais que duas categorias ou classes. Nesta Tese são investigadas e propostas estratégias para a generalização das SVMs para problemas com mais que duas classes, intitulados multiclass. O foco deste trabalho é em estratégias que decompõem o problema multiclass original em múltiplos subproblemas binários, cujas saídas são então combinadas na obtenção da classificação final. As estratégias propostas visam investigar a adaptação das decomposições a cada aplicação considerada, a partir de informações do desempenho obtido em sua solução ou extraídas de seus dados. Os algoritmos implementados foram avaliados em conjuntos de dados gerais e em aplicações reais da área de Bioinformática. Os resultados obtidos abrem várias possibilidades de pesquisas futuras. Entre os benefícios verificados tem-se a obtenção de decomposições mais simples, que requerem menos classificadores binários na solução multiclass.

Abstract

Several problems involve the classification of data into categories, also called classes. Given a dataset containing data whose classes are known, Machine Learning (ML) algorithms can be employed for the induction of a classifier able to predict the class of new data from the same domain, thus performing the desired discrimination. Among the several ML techniques applied to classification problems, the Support Vector Machines (SVMs) are known by their high generalization ability. They are originally conceived for the solution of problems with only two classes, also named binary problems. However, several problems require the discrimination of examples into more than two categories or classes. This thesis investigates and proposes strategies for the generalization of SVMs to problems with more than two classes, known as multiclass problems. The focus of this work is on strategies that decompose the original multiclass problem into multiple binary subtasks, whose outputs are then combined to obtain the final classification. The proposed strategies aim to investigate the adaptation of the decompositions for each multiclass application considered, using information of the performance obtained for its solution or extracted from its examples. The implemented algorithms were evaluated on general datasets and on real applications from the Bioinformatics domain. The results obtained open possibilities of many future work. Among the benefits observed is the obtainment of simpler decompositions, which require less binary classifiers in the multiclass solution.

Este documento foi preparado com o formatador de textos L^AT_EX. O sistema de citações de referências bibliográficas utiliza o padrão *Apalike* do bibT_EX.

Sumário

Dedicatória	i
Agradecimentos	iii
Resumo	v
Abstract	vii
Sumário	xi
Lista de Abreviaturas	xv
Lista de Figuras	xvii
Lista de Tabelas	xix
Lista de Algoritmos	xxi
1 Introdução	1
1.1 Máquinas de Vetores de Suporte e Problemas Multiclasses	2
1.2 Objetivos do Trabalho	4
1.3 Técnicas Propostas	5
1.4 Resumo dos Resultados	6
1.5 Organização da Tese	7
2 Máquinas de Vetores de Suporte	9
2.1 Conceitos Básicos de Aprendizado de Máquina	9
2.2 A Teoria de Aprendizado Estatístico	12
2.2.1 Considerações sobre a Escolha do Classificador	13
2.2.2 Limites no Risco Esperado	14
2.3 SVMs Lineares	17
2.3.1 SVMs com Margens Rígidas	18

2.3.2	SVMs com Margens Suaves	21
2.4	SVMs Não Lineares	23
2.5	Considerações Finais	26
3	Estratégias Multiclasses	29
3.1	Estratégias Decomposicionais	29
3.1.1	Estratégias Baseadas em Matrizes de Códigos	30
3.1.2	Estratégias Hierárquicas	38
3.2	Estratégias Diretas	45
3.2.1	Extensão de Weston e Watkins	46
3.2.2	Extensão de Crammer e Singer	48
3.3	Considerações Finais	50
4	Estratégias Evolutivas	53
4.1	Algoritmos Evolutivos	54
4.1.1	Algoritmo Genético Básico	55
4.1.2	Algoritmos Genéticos em Problemas de Permutação	59
4.1.3	Algoritmos Genéticos para Problemas Multi-objetivo	62
4.2	AGs na Obtenção de Matrizes de Códigos	67
4.2.1	Representação dos Indivíduos	68
4.2.2	Procedimento de Avaliação	69
4.2.3	Operadores Genéticos	70
4.2.4	Algoritmos	72
4.3	AGs na Determinação da Estrutura de DAGs	75
4.3.1	Representação dos Indivíduos	75
4.3.2	Procedimento de Avaliação	77
4.3.3	Operadores Genéticos	78
4.3.4	Algoritmo	78
4.4	Considerações Finais	79
5	Árvores Multiclasses	81
5.1	Teoria de Grafos	81
5.1.1	Definições	82
5.1.2	Árvore Geradora Mínima	84
5.2	Exemplo de Aplicação de AGM: Agrupamento de Objetos	86
5.3	Obtenção de Classificadores Hierárquicos com AGM	88
5.3.1	Definição do Grafo Ponderado	90
5.3.2	Algoritmo	92
5.3.3	Modificações no Algoritmo	96
5.4	Considerações Finais	99

6	Experimentos	101
6.1	Conjuntos de Dados	102
6.1.1	Conjuntos de Dados Naturais	103
6.1.2	Conjuntos de Dados Reais	105
6.2	Metodologia	111
6.2.1	Pré-Processamentos dos Dados	111
6.2.2	Teste Estatístico	113
6.2.3	Procedimento Geral	114
6.3	Obtenção de Matrizes de Códigos	115
6.3.1	Configuração dos Experimentos	116
6.3.2	Resultados	118
6.3.3	Resumo	127
6.4	Determinação da Estrutura de DDAGs e ADAGs	128
6.4.1	Configuração dos Experimentos	128
6.4.2	Resultados	129
6.4.3	Resumo	132
6.5	Determinação da Estrutura de Árvores Multiclasses	133
6.5.1	Configuração dos Experimentos	133
6.5.2	Resultados	133
6.5.3	Resumo	138
6.6	Comparações de Diferentes Estratégias	139
6.6.1	Configuração dos Experimentos	139
6.6.2	Resultados	139
6.6.3	Resumo	143
6.7	Discussões	144
6.7.1	Desempenho nos Conjuntos de Dados Reais	144
6.7.2	Outras Técnicas da Literatura	146
6.7.3	Efeitos de Variações nos Parâmetros das SVMs	151
6.7.4	Outros Resultados	155
6.7.5	Resumo e Análise Crítica	157
6.8	Considerações Finais	161
7	Conclusão	163
7.1	Principais Resultados	164
7.2	Contribuições do Trabalho	166
7.3	Limitações	167
7.4	Trabalhos Futuros	169
7.5	Considerações Finais	171
	Referências Bibliográficas	172

A	Avaliação de Algoritmos de AM	191
B	Resultados Adicionais	193
B.1	Proporções de Dados por Classe em Conjuntos de Dados	193
B.2	Obtenção de Matrizes de Códigos	193
B.3	Determinação da Estrutura de Árvores Multiclasses	197
B.4	Comparações de Diferentes Estratégias	197
B.5	Desempenho nas Classes	198

Lista de Abreviaturas

AD	Árvore de Decisão	7
ADAG	<i>Adaptive Directed Acyclic Graph</i>	6
AE	Algoritmo Evolutivo	54
AG	Algoritmo Genético	5
AGM	Árvore Geradora Mínima	84
AM	Aprendizado de Máquina	9
CS	Algoritmo de Crammer & Singer (2002)	139
CV	<i>Cross-validation</i>	111
DAG	<i>Directed Acyclic Graph</i> (Grafo Direcionado Acíclico)	6
DDAG	<i>Decision Directed Acyclic Graph</i>	6
ECOC	<i>Error Correcting Output Codes</i>	34
LOO	<i>Leave-one-out</i>	43
MLP	<i>Multilayer Perceptron</i>	26
NB	<i>Naïve-Bayes</i>	109
RNA	Rede Neural Artificial	9
SVM	<i>Support Vector Machine</i> (Máquina de Vetores de Suporte)	v
TAE	Teoria de Aprendizado Estatístico	12
tct	Decomposição todos-contra-todos	34
UCI	Universidade da Califórnia em Irvine	146
WW	Algoritmo de Weston & Watkins (1998)	139
VC	Dimensão Vapnik-Chervonenkis	14
1ct	Decomposição um-contra-todos	4

Lista de Figuras

1.1	Exemplos de estratégias decomposicionais para um problema com quatro classes	3
2.1	Indução de classificador em aprendizado supervisionado	11
2.2	Conjunto de treinamento binário e três diferentes hipóteses	12
2.3	Separações de três dados em $\mathbb{R}^2$ por meio de retas	15
2.4	Separações de quatro dados em $\mathbb{R}^2$ por meio de retas	15
2.5	Princípio de minimização do risco estrutural (Smola & Schölkopf, 2002)	16
2.6	Cálculo da distância d entre os hiperplanos H_1 e H_2 (Hearst et al., 1998)	19
2.7	Tipos de SVs: livres (cor cinza) e limitados (cor preta) (Passerini, 2004)	23
2.8	(a) Conjunto de dados não linear; (b) Fronteira não linear no espaço de entradas; (c) Fronteira linear no espaço de características (Müller et al., 2001)	24
3.1	Exemplo de matriz de códigos para um problema com quatro classes	31
3.2	Resultados da aplicação das decodificações de <i>Hamming</i> e baseada em margens	33
3.3	Matrizes de diferentes decomposições para um problema com quatro classes	33
3.4	Classificadores hierárquicos para um problema com seis classes (Schwenker & Palm, 2001)	38
3.5	Exemplo de DDAG para um problema com quatro classes (Platt et al., 2000)	40
3.6	Classificação na região sombreada difere de acordo com a estrutura do DDAG	40
3.7	Exemplo de ADAG para problema com oito classes (Kijssirikul & Ussivakul, 2002)	41
3.8	Duas árvores para um problema com quatro classes (Frank & Kramer, 2004)	44
3.9	SVs em versão multiclases de Weston & Watkins (1998) (Passerini, 2004)	48
4.1	Operador de cruzamento de um ponto	57
4.2	Operador de mutação	57
4.3	Exemplo de dominância de Pareto em problema de minimização com três objetivos	63
4.4	Ilustração de dois possíveis indivíduos para um problema com quatro classes	69
4.5	Operadores de cruzamento para matrizes de códigos	71
4.6	Operadores de mutação para matrizes de códigos	71
4.7	Representação de DDAGs para três classes por meio de listas	75

4.8	Previsão da classe de um exemplo com um DDAG para três classes	76
4.9	Representação de ADAGs para três classes por meio de listas	76
4.10	Previsão da classe de um exemplo com um ADAG para três classes	77
5.1	Exemplos de grafos	82
5.2	Exemplos de subgrafos do grafo ilustrado na Figura 5.1a	83
5.3	Exemplos de árvores direcionadas	84
5.4	Exemplo de operação do algoritmo de Kruskal	85
5.5	Agrupamento hierárquico com adaptação do algoritmo de Kruskal	89
5.6	Visão divisiva do agrupamento hierárquico realizado por uma AGM	90
5.7	Grafos obtidos por diferentes critérios para um problema com três classes	92
5.8	Exemplo de operação do algoritmo para a geração de árvore multiclassess	93
5.9	Agrupamento realizado pelo algoritmo na geração de uma árvore multiclassess . .	94
5.10	Definição de árvore multiclassess a partir de AGM	95
5.11	Grafo para um problema com quatro classes e AGM extraída	96
5.12	Processo de obtenção de árvore multiclassess por meio de adaptação no algoritmo	98
6.1	Conformações estruturais das proteínas	106
6.2	Divisões dos conjuntos de dados para a realização dos experimentos	112
6.3	Quadros resumo dos testes entre as estratégias baseadas em matrizes de códigos	121
6.4	Comparações das taxas de acerto das estratégias baseadas em matrizes de códigos	122
6.5	Quadros resumo dos testes entre árvores multiclassess	135
6.6	Comparações das taxas de acerto das árvores multiclassess	136
6.7	Quadros resumo dos testes entre diferentes estratégias multiclassess	141
6.8	Comparações das taxas de acerto de diferentes estratégias multiclassess	142
6.9	Indivíduo de AG com extensão para determinar os parâmetros das SVMs	152
6.10	Efeito de variação de parâmetros das SVMs em DDAGs e ADAGs	154
6.11	Efeito de variação do número máximo de classificadores no AG lexicográfico . .	155
A.1	Exemplo de matriz de confusão para um problema com três classes	192

Lista de Tabelas

2.1	Funções Kernel mais comuns (Chang & Lin, 2004b)	26
6.1	Principais características dos conjuntos de dados	102
6.2	Números de partições de validação cruzada	111
6.3	Matriz para a comparação múltipla de algoritmos	114
6.4	Taxas de acerto das estratégias baseadas em matrizes de códigos	119
6.5	Números de classificadores das estratégias baseadas em matrizes de códigos	124
6.6	Redução do número de classificadores das matrizes dos AGs em relação à 1ct	125
6.7	Tempos de processamento de AGs e do método aleatório, em segundos	126
6.8	Taxas de acerto de DDAGs e ADAGs	130
6.9	Tempos de processamento dos AGs para a obtenção de DAGs, em segundos	131
6.10	Taxas de acerto das árvores multiclases	134
6.11	Tempos de execução de algoritmos para a obtenção de árvores multiclases	137
6.12	Taxas de acerto de diferentes estratégias multiclases	140
6.13	Números de classificadores em diferentes estratégias decomposicionais	143
6.14	Resultados de seleção de parâmetros para as SVMs em AG lexicográfico	153
B.1	Proporções de dados nas classes do conjunto de dados <i>iris</i>	193
B.2	Proporções de dados nas classes do conjunto de dados <i>car</i>	194
B.3	Proporções de dados nas classes do conjunto de dados <i>glass</i>	194
B.4	Proporções de dados nas classes do conjunto de dados <i>sat-image</i>	194
B.5	Proporções de dados nas classes do conjunto de dados <i>segment</i>	194
B.6	Proporções de dados nas classes do conjunto de dados <i>opt-digits</i>	194
B.7	Proporções de dados nas classes do conjunto de dados <i>pen-digits</i>	195
B.8	Proporções de dados nas classes do conjunto de dados <i>vowel</i>	195
B.9	Proporções de dados nas classes do conjunto de dados <i>structural</i>	195
B.10	Proporções de dados nas classes do conjunto de dados <i>lung</i>	195
B.11	Proporções de dados nas classes do conjunto de dados <i>leukocyte</i>	196
B.12	Proporções de dados nas classes do conjunto de dados <i>fungi</i>	196
B.13	Taxas de desconhecidos das estratégias baseadas em matrizes de códigos	196

B.14 Testes estatísticos entre estratégias baseadas em matrizes de códigos	197
B.15 Testes estatísticos entre árvores multiclases	197
B.16 Testes estatísticos entre diferentes estratégias multiclases	198
B.17 Taxas de acerto nas classes do conjunto de dados <i>iris</i>	198
B.18 Taxas de acerto nas classes do conjunto de dados <i>car</i>	199
B.19 Taxas de acerto nas classes do conjunto de dados <i>lung</i>	199
B.20 Taxas de acerto nas classes do conjunto de dados <i>glass</i>	199
B.21 Taxas de acerto nas classes do conjunto de dados <i>segment</i>	200
B.22 Taxas de acerto nas classes do conjunto de dados <i>leukocyte</i>	200
B.23 Taxas de acerto nas classes do conjunto de dados <i>fungi</i>	201
B.24 Taxas de acerto nas classes do conjunto de dados <i>structural</i>	201
B.25 Taxas de acerto nas classes do conjunto de dados <i>sat-image</i>	201
B.26 Taxas de acerto nas classes do conjunto de dados <i>opt-digits</i>	202
B.27 Taxas de acerto nas classes do conjunto de dados <i>pen-digits</i>	202
B.28 Taxas de acerto nas classes do conjunto de dados <i>vowel</i>	203

Lista de Algoritmos

4.1	Algoritmo Genético (Carvalho et al., 2003)	58
4.2	Algoritmo SPEA2 (Zitzler et al., 2002)	65
5.1	Algoritmo de Kruskal (Cormen et al., 2001)	85
5.2	Agrupamento hierárquico aglomerativo (Kumar, 2000)	87
5.3	Modificação do algoritmo de Kruskal para o agrupamento de objetos	88
5.4	Algoritmo para a obtenção de árvore multiclases	93
5.5	Algoritmo modificado para a obtenção de árvore multiclases	98

Capítulo 1

Introdução

Diversos problemas envolvem a discriminação de dados em categorias ou classes, uma tarefa que é comumente denominada classificação. A partir de conjuntos de dados cujas classes são conhecidas, algoritmos de Aprendizado de Máquina (AM) (Mitchell, 1997) podem ser utilizados na indução de um classificador capaz de realizar a discriminação desejada para novos dados do mesmo domínio.

No caso em que tem-se apenas duas classes, o problema de classificação é conhecido como binário. Um exemplo de problema deste tipo é o diagnóstico médico de uma determinada patologia. A partir de informações clínicas de um paciente, deve-se determinar se ele possui ou não a doença de interesse. As classes representam então a presença ou a ausência da doença. Vários problemas reais, entretanto, envolvem a discriminação de mais que duas categorias ou classes. Alguns exemplos de tais problemas são a classificação de dígitos manuscritos e a distinção de múltiplos tipos de tumores em Bioinformática.

Entre as diversas técnicas de AM existentes, as Máquinas de Vetores de Suporte (*Support Vector Machines* - SVMs) (Cristianini & Shawe-Taylor, 2000) vêm recebendo crescente atenção, apresentando bom desempenho na solução de diversos problemas de classificação (Hearst et al., 1998). Porém, elas são originalmente formuladas para a solução de problemas binários. Nesta Tese são estudadas e propostas estratégias para a generalização das SVMs a problemas com mais que duas classes, referenciados por multiclass.

Na Seção 1.1 são discutidas as principais motivações para este trabalho. A partir delas, na Seção 1.2 são apresentados os objetivos seguidos na elaboração desta Tese. Na Seção 1.3 as técnicas propostas conforme os objetivos apontados são brevemente descritas. Um resumo dos principais resultados obtidos na aplicação prática dessas técnicas é apresentado na Seção 1.4. A organização desta Tese é então comentada na Seção 1.5.

1.1 Máquinas de Vetores de Suporte e Problemas Multiclasses

As SVMs constituem uma técnica de AM com princípios na teoria de aprendizado estatístico de Vapnik (1995). Em problemas de classificação, o principal objetivo é construir um classificador (preditor), a partir de um conjunto de dados conhecidos, que seja capaz de determinar corretamente a classe de novos exemplos. As classes podem ser visualizadas como categorias dos dados, e descrevem o fenômeno de interesse sobre o qual se deseja fazer previsões (Monard & Baranauskas, 2003a).

A formulação original das SVMs é concebida para a discriminação de dados pertencentes a duas classes, um problema de classificação binário. Muitos problemas práticos, contudo, possuem mais que duas classes preditivas, sendo referenciados pelo termo multiclasses. Um problema de classificação multiclasses é intrinsecamente mais complexo que um problema binário, uma vez que o classificador gerado deve ser capaz de separar os dados em um número maior categorias, o que também aumenta a chance de erros de classificação serem cometidos. Por esse mesmo motivo, sua complexidade é crescente à medida que o número de classes aumenta. Seja, por exemplo, um problema de classificação balanceado, com números semelhantes de dados por classe, e com classes equiprováveis. Caso ele seja binário, a chance de produzir uma classificação correta aleatoriamente é de 50%. Para quatro classes, essa chance reduz para 25%.

Existem duas abordagens para a generalização das SVMs a problemas multiclasses. A mais comum é decompor o problema em vários subproblemas binários. As SVMs são utilizadas na indução de classificadores para cada um desses subproblemas. As saídas desses classificadores são então combinadas na obtenção da previsão multiclasses. A segunda abordagem modifica o algoritmo original de treinamento das SVMs, criando versões multiclasses. Hsu & Lin (2002) apontam que tais modificações, em geral, levam a algoritmos computacionalmente caros. Em vista disso, é cada vez mais frequente a alternativa de decompor o problema multiclasses em subproblemas binários, uma estratégia denominada decomposicional.

É possível indicar uma série de outras motivações para o emprego de estratégias decompositivas na solução de problemas multiclasses. Entre elas, Moreira & Mayoraz (1997) e Fürnkranz (2002) apontam que o uso de um procedimento decomposicional pode reduzir a complexidade envolvida na separação das classes. Com isso, ela pode beneficiar até mesmo técnicas de AM cujos algoritmos são facilmente extensíveis à solução de problemas multiclasses. Knerr et al. (1992), por exemplo, observaram que as classes de um problema de reconhecimento de dígitos poderiam ser separadas linearmente quando consideradas em pares. Eles optaram então por combinar classificadores lineares para todos os pares de classes na solução desse problema, uma alternativa considerada mais simples que o uso de um único classificador capaz de separar todas as classes simultaneamente.

Pimenta (2005) também aponta vantagens no uso de decomposições binárias no caso em que os erros na classificação dos dados de diferentes classes possuem custos distintos.

A decomposição abre ainda possibilidades para o uso de um processamento paralelo, uma vez que os subproblemas binários são independentes e podem assim ser solucionados em processadores diferentes.

O enfoque desse trabalho está no estudo de variações da estratégia decomposicional para a solução de problemas multiclasses com SVMs. As estratégias decompositivas aqui investigadas foram genericamente divididas em dois grupos: baseadas em matrizes de códigos e hierárquicas.

No primeiro grupo estão as estratégias que podem ser representadas por uma matriz de códigos $\mathbf{M}$ e nas quais a obtenção da classificação final envolve um passo de decodificação que considera as saídas de todos os preditores binários. As linhas da matriz $\mathbf{M}$ contêm códigos que são atribuídos a cada classe. As suas colunas definem partições binárias das 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. Um exemplo de matriz de códigos para um problema com quatro classes é apresentado na Figura 1.1a. Ela possui quatro linhas, equivalentes aos códigos das quatro classes. As quatro colunas constituem os classificadores binários empregados na decomposição, representados pelos símbolos f_1 , f_2 , f_3 e f_4 . O classificador f_1 é induzido de forma a distinguir a classe 1 das demais. O mesmo raciocínio se aplica aos classificadores f_2 , f_3 e f_4 para as classes 2, 3 e 4, respectivamente.

Figura 1.1: Exemplos de estratégias decompositivas para um problema com quatro classes

Na computação da classe de um novo dado nas estratégias baseadas em matrizes de códigos, todos os preditores binários são consultados, o que leva à geração de um vetor com l previsões. Esse vetor é então comparado às linhas da matriz. É selecionada a classe representada pela linha considerada mais próxima ao vetor, segundo alguma medida de distância.

Nas estratégias baseadas em matrizes de códigos tradicionais, a decomposição do pro-

blema multiclass é realizada a priori. O exemplo mais comum é o da matriz um-contratodos (1ct). Para um problema com k classes, ela possui k colunas. Cada coluna representa um classificador binário que distingue uma única classe das demais. A matriz 1ct para um problema com quatro classes foi ilustrada na Figura 1.1a.

O problema da decomposição a priori é que ela não leva em consideração características e propriedades da aplicação multiclass. Com isso, algumas das partições binárias das classes podem não ser adequadas ou ser difíceis de serem realizadas. Além disso, ela pode requerer mais classificadores binários que o necessário para distinguir as classes com um bom desempenho.

As estratégias do segundo grupo estudado organizam os preditores binários hierarquicamente, em camadas. Um exemplo é apresentado na Figura 1.1b. A idéia básica é realizar inicialmente discriminações mais gerais, as quais são refinadas sucessivamente até a obtenção da classificação final. Os preditores e as classes são representados por nós em grafos. Na raiz tem-se normalmente um preditor que divide todas as classes do problema em dois grupos. Esses grupos são também recursivamente subdivididos em duas partes, até restar uma única classe. As classes estarão em nós denominados folhas.

Na classificação de um novo dado, a hierarquia é percorrida da primeira camada até uma folha, que fornece a previsão final. Nesse processo, apenas uma parte dos classificadores binários são consultados. Essa é uma vantagem das estratégias hierárquicas sobre as baseadas em matrizes de códigos anteriormente descritas.

Um problema relevante no caso das estratégias hierárquicas encontra-se na definição de sua estrutura, ou seja, em como as partições das classes devem ser realizadas nos nós da hierarquia. Para determinar essa estrutura, pode-se recorrer também a características próprias de cada aplicação.

A partir dessas considerações, foram desenvolvidos os objetivos desse trabalho, os quais são apresentados a seguir.

1.2 Objetivos do Trabalho

A partir dos aspectos relacionados às estratégias decomposicionais discutidos na seção anterior, foram definidos dois problemas abordados nos estudos conduzidos nesta Tese.

O primeiro constitui investigar como informações sobre o desempenho de matrizes de códigos na solução de problemas multiclass podem auxiliar na sua adaptação a uma dada aplicação. Dessa forma, as decomposições realizadas por essas matrizes passam a ser definidas de acordo com cada problema multiclass. Isso levou a uma outra investigação: se esse tipo de abordagem poderia levar à obtenção de decomposições mais simples, que necessitassem de menos classificadores binários.

O segundo problema tratado, relacionado às estratégias hierárquicas, diz respeito a como definir a organização de seus preditores binários. Dois tipos básicos de estruturas hierárquicas foram estudados. O primeiro organiza preditores binários para todos os pares de classes em grafos direcionados acíclicos. Investigou-se como o desempenho de diferen-

tes organizações desses preditores nos grafos poderia ser utilizado na definição de sua estrutura final, adequando-a assim a cada problema multiclass. O outro tipo de estratégia hierárquica utiliza uma estrutura de árvore, um tipo especial de grafo. Nesse caso, estudou-se como utilizar as informações extraídas dos dados de cada aplicação multiclass na definição das partições binárias das classes contidas nos nós da árvore.

O ponto comum entre essas pesquisas está na investigação da adaptação de soluções decomposicionais, utilizando SVMs binárias, a cada problema multiclass particular. Seguiu-se basicamente duas abordagens na realização dessa tarefa. A primeira utiliza informações do desempenho das soluções para adaptá-las ao problema. A segunda, empregada no caso das árvores, utiliza informações extraídas a partir dos conjuntos de dados para definir as decomposições realizadas.

Os algoritmos e resultados obtidos encontram-se voltados às SVMs. Isso ocorre principalmente com as técnicas que utilizam informações de desempenho na adaptação das decomposições, uma vez que as SVMs são utilizadas na indução dos preditores binários cujo desempenho conjunto será avaliado na solução multiclass. É possível, contudo, adaptar as propostas introduzidas a outras técnicas de AM.

1.3 Técnicas Propostas

A determinação de matrizes de códigos adequadas a cada problema multiclass foi formulada como uma busca. Procura-se a combinação de preditores binários na matriz que leve a um bom desempenho na solução multiclass. Além disso, tenta-se minimizar também o número de preditores binários contidos nessas matrizes, representando assim a busca por decomposições mais simples, que apresentem um bom desempenho de classificação. As matrizes de códigos obtidas devem ainda obedecer à restrição de não possuir classificadores iguais, o que representaria o emprego de preditores binários idênticos em uma mesma decomposição.

Introduziu-se o uso de Algoritmos Genéticos (AGs) (Mitchell, 1999) na solução da busca formulada. Os AGs são algoritmos de busca e otimização com princípios embasados na teoria de evolução natural e na genética. Parte-se de uma população de soluções iniciais para o problema, as quais são modificadas por meio de um processo evolutivo. Nesse processo, as soluções são continuamente adaptadas de forma a se tornarem mais aptas a solucionar o problema. Com esse procedimento, as matrizes de códigos podem então ser adaptadas à solução de cada problema multiclass particular.

Dois tipos de AGs foram implementados. Ambos manipulam soluções na forma de matrizes de códigos. Eles procuram matrizes sem classificadores idênticos com bom desempenho na solução multiclass, buscando também reduzir o número de classificadores (colunas) das mesmas. O primeiro AG, denominado lexicográfico, lida com os objetivos anteriormente delineados impondo uma ordem nos mesmos. Privilegia-se a minimização do erro na tarefa multiclass, o qual mede o desempenho das matrizes, sobre a redução do número de preditores binários. Classificadores idênticos nas matrizes são evitados com o

uso de um procedimento de penalização. O segundo AG codificado lida simultaneamente com os objetivos de minimizar o erro multiclases, o número de preditores binários e a quantidade de classificadores idênticos nas matrizes. Ele faz isso utilizando o conceito de dominância de Pareto, que provê meios para a análise e a comparação de soluções segundo múltiplos objetivos.

AGs também foram empregados para determinar a ordem dos preditores binários em dois tipos de grafos direcionados acíclicos (*Directed Acyclic Graphs* - DAGs): os DDAGs (*Decision DAGs*) (Platt et al., 2000) e os ADAGs (*Adaptive DAGs*) (Kijisirikul & Ussivakul, 2002). Esses DAGs provêm classificações multiclases por meio da organização hierárquica de classificadores binários para pares de classes. Como o uso dos DAGs pode ser resumido à operação de uma lista de classes, realizou-se analogias com a utilização de AGs na solução de problemas de permutação. Essas listas são assim adaptadas pelo AG de acordo com o seu erro na solução multiclases. Busca-se com isso permutações de classes que levem a um DAG com bom desempenho em cada problema multiclases considerado.

Como mencionado na seção anterior, outra forma de organizar os preditores binários hierarquicamente em uma solução multiclases é por meio do uso de árvores. Uma vantagem das árvores é que elas requerem o menor número de preditores entre as principais estratégias decomposicionais encontradas na literatura, sendo igual a $k - 1$ para um problema com k classes. Foram então desenvolvidos dois algoritmos para determinar os preditores binários nesse tipo de hierarquia, a partir de adaptações em algoritmos da teoria de grafos. Eles realizam um agrupamento hierárquico das classes do problema, o qual define diretamente as partições binárias das classes nos nós da árvore. Para realizar o agrupamento, parte-se de um grafo que pondera a relação entre todas as classes a partir de informações coletadas de seus dados. Dessa forma, a estrutura da árvore é determinada iterativamente com base em informações extraídas dos dados que pertencem a cada classe.

1.4 Resumo dos Resultados

A versão lexicográfica de AG para a determinação de matrizes de códigos foi capaz de gerar soluções com desempenho comparável ao de outras estratégias baseadas em matrizes de códigos encontradas na literatura. As matrizes determinadas em geral apresentaram a vantagem de utilizar menos classificadores binários nas soluções. Logo, a adaptação de matrizes de códigos considerando o desempenho obtido em cada aplicação pode levar a matrizes simplificadas que apresentam bons resultados na solução multiclases. O AG baseado na dominância de Pareto, por sua vez, embora tenha reduzido o número de preditores binários nas matrizes, em alguns casos não foi capaz de obter matrizes com taxas de acerto comparáveis às das estratégias da literatura. A consideração simultânea dos objetivos de minimizar o erro multiclases, o número de preditores binários e a quantidade de classificadores iguais não se mostrou então adequada.

Verificou-se que a determinação da ordem dos preditores binários em DDAGs e ADAGs

pode não ser um problema crucial em seu uso. Houve pouca variação de desempenho entre estruturas distintas geradas de maneira aleatória, as quais alcançaram bons resultados. Com isso, as estruturas determinadas pelos AGs tiveram desempenho semelhante ao das aleatórias.

Confirmou-se nos experimentos com os algoritmos para a definição da estrutura de árvores que as informações obtidas dos dados podem auxiliar esse processo. As árvores obtidas apresentaram um bom desempenho, o qual foi em geral comparável ao de outras estratégias multiclases. Novamente, verificou-se pouca variação entre os resultados de estruturas de árvores distintas, embora ela tenha sido maior que a obtida no caso dos DDAGs e ADAGs.

Dois critérios foram testados na ponderação da relação entre as classes sobre a qual os algoritmos operam. O primeiro considera a distância entre os centróides das classes, calculados a partir dos dados da aplicação. O segundo utiliza informações da confusão encontrada por Árvore de Decisão (ADs) (Quinlan, 1986) na separação dos dados. Embora geralmente os dois critérios tenham alcançado resultados semelhantes, em alguns casos verificou-se uma maior adequabilidade de um sobre o outro.

Também realizou-se uma comparação entre as matrizes determinadas pelo AG, os DDAGs, as árvores e dois algoritmos que reformulam as SVMs em versões diretas. Em geral, os resultados obtidos foram semelhantes, embora as matrizes de códigos determinadas pelo AG e um dos algoritmos diretos tenham se sobressaído um pouco.

Os resultados das SVMs são largamente influenciados pela escolha de parâmetros adequados. Ainda que não tenha sido o objetivo desse trabalho, verificou-se em experimentos preliminares que os AGs podem também ser empregados na busca de parâmetros diferenciados para cada SVM binária contida em uma decomposição. Efetuar esse ajuste empiricamente é um processo muito custoso, principalmente para problemas com números elevados de classes. No caso dos DAGs, a realização desse tipo de ajuste pode inclusive levar a maiores variações entre o desempenho de estruturas distintas.

Os experimentos realizados envolveram conjuntos de dados gerais e reais da área de Bioinformática. Apesar do enfoque não ter sido o de obter os melhores resultados em cada aplicação, foi possível verificar que as técnicas propostas são também capazes de solucionar problemas práticos reais que apresentam alguns desafios na predição multiclases, como o desbalanceamento na proporção de dados por classe.

1.5 Organização da Tese

Esta Tese encontra-se organizada conforme a seguinte estrutura:

- No Capítulo 2 é descrita a formulação original das SVMs, utilizada na solução de problemas com duas classes.
- No Capítulo 3 são apresentadas diversas técnicas descritas na literatura para a extensão das SVMs a aplicações multiclases.

- No Capítulo 4 são apresentadas as duas técnicas evolutivas para a obtenção de SVMs multiclasss a partir de preditores binários.
- No Capítulo 5 introduz-se as técnicas baseadas em grafos, que são utilizadas na definição da estrutura de árvores para a solução multiclasss.
- No Capítulo 6, diversos experimentos e resultados da aplicação das técnicas discutidas nos capítulos 3, 4 e 5 são apresentados.
- No Capítulo 7 finaliza-se esta Tese com discussões das contribuições e limitações da pesquisa realizada. Também são delineados possíveis trabalhos futuros.

Capítulo 2

Máquinas de Vetores de Suporte

As Máquinas de Vetores de Suporte (*Support Vector Machines* - SVMs) constituem uma técnica de aprendizado que vem recebendo crescente atenção da comunidade de Aprendizado de Máquina (AM) (Mitchell, 1997). Os resultados da aplicação dessa técnica são comparáveis e muitas vezes superiores aos obtidos por outros algoritmos de aprendizado, como as Redes Neurais Artificiais (RNAs) (Haykin, 1999; Braga et al., 2000). Exemplos de aplicações de sucesso podem ser encontrados em diversos domínios, como na categorização de textos (Joachims, 2002), na análise de imagens (Pontil & Verri, 1998; Kim et al., 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 & Chervonenkis, 1971). Essa teoria estabelece uma série de princípios que devem ser seguidos na obtenção de classificadores com boa generalização, definida como a sua capacidade de prever corretamente a classe de novos dados do mesmo domínio em que o aprendizado ocorreu.

Iniciando este capítulo, na Seção 2.1 são apresentados alguns conceitos básicos de AM. Uma breve introdução aos principais conceitos da teoria de aprendizado estatístico é então apresentada na Seção 2.2. A partir deles, na Seção 2.3 as SVMs são formuladas para a definição de fronteiras lineares para a separação de conjuntos de dados binários. A seguir, na Seção 2.4 as SVMs são estendidas de forma a definir fronteiras não lineares. Concluindo, na Seção 2.5 são apresentadas algumas discussões dos conceitos vistos e as considerações finais do capítulo.

2.1 Conceitos Básicos de Aprendizado de Máquina

As técnicas de AM empregam um princípio de inferência denominado indução, no qual obtém-se conclusões genéricas a partir de um conjunto particular de exemplos. O

aprendizado indutivo pode ser dividido em dois tipos principais: supervisionado e não-supervisionado.

No aprendizado supervisionado tem-se a figura de um professor externo, o qual apresenta o conhecimento do ambiente por conjuntos de exemplos na forma: entrada, saída desejada (Haykin, 1999). O algoritmo de AM extrai a representação do conhecimento a partir desses exemplos. O objetivo é que a representação gerada seja capaz de produzir saídas corretas para novas entradas não apresentadas previamente.

No aprendizado não-supervisionado não há a presença de um professor, ou seja, não existem exemplos rotulados. O algoritmo de AM aprende a representar (ou agrupar) as entradas submetidas segundo uma medida de qualidade. Essas técnicas são utilizadas principalmente quando o objetivo for encontrar padrões ou tendências que auxiliem no entendimento dos dados (Souto et al., 2003).

O tipo de aprendizado empregado neste trabalho é o supervisionado. Neste caso, dado um conjunto de exemplos rotulados na forma $(\mathbf{x}_i, y_i)$, em que $\mathbf{x}_i$ representa um exemplo e y_i denota seu rótulo, deve-se produzir um classificador, também denominado modelo, preditor ou hipótese, capaz de predizer precisamente o rótulo de novos dados. Esse processo de indução de um classificador a partir de uma amostra de dados é denominado treinamento. O classificador obtido também pode ser visto como uma função f , a qual recebe um dado $\mathbf{x}$ e fornece uma predição y .

Os rótulos ou classes representam o fenômeno de interesse sobre o qual se deseja fazer previsões. Neste trabalho, os rótulos assumem valores discretos $1, \dots, k$. Tem-se então um problema de classificação. Caso os rótulos possuam valores contínuos, tem-se uma regressão (Mitchell, 1997). Um problema de classificação no qual $k = 2$ é denominado binário. Para $k > 2$ configura-se um problema multiclases, que é objeto de estudo desta Tese.

Cada exemplo, também referenciado por dado ou caso, é tipicamente representado por um vetor de características. Cada característica, também denominada atributo, expressa um determinado aspecto do exemplo (Monard & Baranauskas, 2003a). Normalmente, há dois tipos básicos de atributos: nominal e contínuo. Um atributo é definido como nominal (ou categórico) quando não existe uma ordem entre os valores que ele pode assumir (por exemplo, entre cores). No caso de atributos contínuos, é possível definir uma ordem linear nos valores assumidos (por exemplo, entre pesos $\in \mathbb{R}$).

Um requisito importante para as técnicas de AM é que elas sejam capazes de lidar com dados imperfeitos, denominados ruídos. Muitos conjuntos de dados apresentam esse tipo de caso, sendo alguns erros comuns a presença de dados com rótulos e/ou atributos incorretos. A técnica de AM deve idealmente ser robusta a ruídos presentes nos dados, procurando não fixar a obtenção dos classificadores sobre esse tipo de caso. Deve-se também minimizar a influência de *outliers* no processo de indução. Os *outliers* são exemplos muito distintos dos demais presentes no conjunto de dados. Esses dados podem ser ruídos ou casos muito particulares, raramente presentes no domínio.

Os conceitos referentes à geração de um classificador a partir do aprendizado supervisionado são representados de forma simplificada na Figura 2.1. Tem-se nessa figura um conjunto com n dados. Cada dado $\mathbf{x}_i$ possui m atributos, ou seja, $\mathbf{x}_i = (x_{i1}, \dots, x_{im})$. As variáveis y_i representam as classes. A partir dos dados e os seus respectivos rótulos, o algoritmo de AM extrai um classificador. Pode-se considerar que o modelo gerado fornece uma descrição compacta dos dados fornecidos (Baranauskas & Monard, 2000).

Figura 2.1: Indução de classificador em aprendizado supervisionado

A obtenção de um classificador por um algoritmo de AM a partir de uma amostra de dados também pode ser considerada um processo de busca (Mitchell, 1997). Procura-se, entre todas as hipóteses que o algoritmo é capaz de gerar a partir dos dados, aquela com melhor capacidade de descrever o domínio em que ocorre o aprendizado.

Para estimar a taxa de predições corretas ou incorretas (também denominadas taxa de acerto e taxa de erro, respectivamente) obtidas por um classificador sobre novos dados, o conjunto de exemplos é, em geral, dividido em dois subconjuntos disjuntos: de treinamento e de teste. O subconjunto de treinamento é utilizado no aprendizado do conceito e o subconjunto de teste é utilizado para medir o grau de efetividade do conceito aprendido na predição da classe de novos dados. Nesse processo, os dados de teste são apresentados ao classificador induzido e as predições produzidas pelo modelo são comparadas aos rótulos conhecidos, produzindo assim a estimativa de acerto/erro.

Um conceito comumente empregado em AM é o de generalização de um classificador, definida como a sua capacidade de prever corretamente a classe de novos dados. No caso em que o modelo se especializa nos dados utilizados em seu treinamento, apresentando uma baixa taxa de acerto quando confrontado com novos dados, tem-se a ocorrência de um super-ajustamento (*overfitting*). É também possível induzir hipóteses que apresentem uma baixa taxa de acerto mesmo no subconjunto de treinamento, configurando uma condição de sub-ajustamento (*underfitting*). Essa situação pode ocorrer, por exemplo, quando os exemplos de treinamento disponíveis são pouco representativos ou quando o modelo obtido é muito simples (Monard & Baranauskas, 2003a). Na Seção 2.2, esses conceitos são ilustrados e discutidos novamente. São feitas então considerações e motivações sobre a escolha de classificadores com boa capacidade de generalização.

Alguns conceitos adicionais quanto à avaliação do desempenho dos classificadores gerados por técnicas de AM são apresentados no Apêndice A.

2.2 A Teoria de Aprendizado Estatístico

Seja f um classificador e F 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 T , composto de n pares $(\mathbf{x}_i, y_i)$, para gerar um classificador particular $\hat{f} \in F$.

Considere, por exemplo, o conjunto de treinamento da Figura 2.2 (Smola & Schölkopf, 2002). O objetivo do processo de aprendizado é encontrar um classificador que separe os dados das classes “círculo” e “triângulo”. As funções ou hipóteses consideradas são ilustradas na figura por meio das bordas, também denominadas fronteiras de decisão, traçadas entre as classes.

Figura 2.2: Conjunto de treinamento binário e três diferentes hipóteses

Na imagem da Figura 2.2a, tem-se uma hipótese que classifica corretamente todos os exemplos do conjunto de treinamento, incluindo dois possíveis ruídos. Por ser muito específica para o conjunto de treinamento, essa função apresenta elevada suscetibilidade a cometer erros quando confrontada com novos dados. Esse caso representa a ocorrência de um super-ajustamento (*overfitting*) do modelo aos dados de treinamento.

Um outro classificador poderia desconsiderar pontos pertencentes a classes opostas que estejam muito próximos entre si. A ilustração da Figura 2.2c representa essa alternativa. A nova hipótese considerada, porém, comete muitos erros, mesmo para casos que podem ser considerados simples. Tem-se assim a ocorrência de um sub-ajustamento (*underfitting*), pois o classificador não é capaz de se ajustar mesmo aos exemplos de treinamento.

Um compromisso entre as duas funções descritas é representado na Figura 2.2b. Esse preditor tem complexidade intermediária e classifica corretamente grande parte dos dados, sem se fixar demasiadamente em qualquer ponto individual.

A Teoria de Aprendizado Estatístico (TAE) estabelece condições matemáticas que auxiliam na escolha de um classificador particular $\hat{f}$ 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.

2.2.1 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 dados e os seus rótulos (Burgess, 1998; Smola & Schölkopf, 2002). O erro (também denominado risco) esperado de um classificador f para dados de teste pode então ser quantificado pela Equação 2.1 (Müller et al., 2001). O risco esperado mede então a capacidade de generalização de f (Passerini, 2004). Na Equação 2.1, $c(f(\mathbf{x}), y)$ é uma função de custo relacionando a previsão $f(\mathbf{x})$ quando a saída desejada é y . Um tipo de função de custo comumente empregada em problemas de classificação é a 0/1, definida por $c(f(\mathbf{x}), y) = \frac{1}{2} |y - f(\mathbf{x})|$ (Smola & Schölkopf, 2002). Essa função retorna o valor 0 se $\mathbf{x}$ é classificado corretamente e 1 caso contrário.

$$R(f) = \int c(f(\mathbf{x}), y) dP(\mathbf{x}, y) \quad (2.1)$$

Infelizmente, não é possível minimizar o risco esperado apresentado na Equação 2.1 diretamente, uma vez que em geral a distribuição de probabilidade $P(\mathbf{x}, y)$ é desconhecida (Müller et al., 2001). 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 $\hat{f}$ que minimize o erro sobre esses dados e espera-se que esse procedimento leve também a um menor erro sobre os dados de teste (Smola & Schölkopf, 2002). O risco empírico de f , fornecido pela Equação 2.2, mede o desempenho do classificador nos dados de treinamento, por meio da taxa de classificações incorretas obtidas em T (Müller et al., 2001).

$$R_{emp}(f) = \frac{1}{n} \sum_{i=1}^n c(f(\mathbf{x}_i), y_i) \quad (2.2)$$

Esse processo de indução com base nos dados de treinamento conhecidos constitui o princípio de minimização do risco empírico (Smola & 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 que memoriza todos os dados 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, permitindo que $\hat{f}$ seja escolhida a

partir de um conjunto de funções amplo F , é sempre possível encontrar uma f 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 super-ajustar a eles. Deve-se então restringir a classe de funções da qual $\hat{f}$ 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 obter (Smola & 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 próxima seção relaciona alguns dos principais limites sobre os quais as SVMs se baseiam.

2.2.2 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 Equação 2.3, é garantido com probabilidade $1 - \theta$, em que $\theta \in [0, 1]$ (Burges, 1998).

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{h \left(\ln \left(\frac{2n}{h} \right) + 1 \right) - \ln \left(\frac{\theta}{4} \right)}{n}} \quad (2.3)$$

Nessa equação, h denota a dimensão Vapnik-Chervonenkis (VC) (Vapnik, 1995) da classe de funções F à qual f pertence, n representa a quantidade de exemplos no conjunto de treinamento T e a parcela de raiz na soma é referenciada como termo de capacidade.

A dimensão VC h mede a capacidade do conjunto de funções F (Burges, 1998). Quanto maior o seu valor, mais complexas são as funções de classificação que podem ser induzidas a partir de F . Dado um problema de classificação binário, essa dimensão é definida como o número máximo de exemplos que podem ser particionados em duas classes pelas funções contidas em F , para todas as possíveis combinações binárias desses dados.

Para ilustrar esse conceito, considere os três dados apresentados na Figura 2.3 (Smola & Schölkopf, 2002). Pode-se verificar que, para qualquer conformação arbitrária dos rótulos “círculo” e “triângulo” que esses dados possam assumir, é possível determinar retas capazes de separá-los. Porém, para os quatro pontos em $\mathbb{R}^2$ ilustrados na Figura 2.4, existem rótulos para os dados que podem ser separados por uma reta (Figura 2.4a), mas também é possível definir rótulos tal que uma só reta seja incapaz de realizar a separação em classes (Figura 2.4b) (Haykin, 1999). Para uma divisão binária arbitrária desses quatro pontos em $\mathbb{R}^2$, deve-se então recorrer a funções de complexidade superior à das retas. Essa observação se aplica a quaisquer quatro pontos no espaço bidimensional. Portanto, a dimensão VC do conjunto de funções lineares no espaço bidimensional é 3, uma vez que existe (pelo menos) uma configuração de três pontos nesse espaço que pode ser particionada por retas em todas as $2^3 = 8$ combinações binárias de rótulos.

A contribuição principal da Inequação 2.3 está em afirmar a importância de se controlar

 Figura 2.3: Separações de três dados em $\mathbb{R}^2$ por meio de retas

 Figura 2.4: Separações de quatro dados em $\mathbb{R}^2$ por meio de retas

a capacidade do conjunto de funções F 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 $\hat{f}$ que minimize o risco empírico e que pertença a uma classe de funções F com baixa dimensão VC h . Com esses objetivos, definiu-se um princípio de indução denominado minimização do risco estrutural (Vapnik, 1998).

Como no limite apresentado o termo de capacidade diz respeito à classe de funções F e o risco empírico refere-se a um classificador particular f , para minimizar ambas as parcelas divide-se inicialmente F em subconjuntos de funções com dimensão VC crescente (Vapnik, 1998). É comum referir-se a esse processo como introduzir uma estrutura em F , sendo os subconjuntos definidos também denominados estruturas (Smola & Schölkopf, 2002). Minimiza-se então o limite sobre as estruturas introduzidas.

Considera-se subconjuntos F_i da seguinte forma: $F_0 \subset F_1 \subset \dots \subset F_q \subset F$. Como cada F_i é maior com o crescimento do índice i , a capacidade do conjunto de funções que ele representa também é maior à medida que i cresce, ou seja, $h_0 < h_1 < \dots < h_q < h$. Para um subconjunto particular F_k , seja $\hat{f}_k$ o classificador com o menor risco empírico. A medida que k cresce o risco empírico de $\hat{f}_k$ diminui, uma vez que a complexidade

do conjunto de classificadores é maior. Porém, o termo de capacidade aumenta com k . Como resultado, deve haver um valor ótimo $\bar{k}$ em que se obtém uma soma mínima do risco empírico e do termo de capacidade, minimizando assim o limite sobre o risco esperado. A escolha da função $\hat{f}_{\bar{k}}$ constitui o princípio da minimização do risco estrutural. Os conceitos discutidos são ilustrados na Figura 2.5.

Figura 2.5: Princípio de minimização do risco estrutural (Smola & Schölkopf, 2002)

Embora o limite representado na Equação 2.3 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 de h poder ser desconhecido ou infinito (Müller et al., 2001).

Para funções de decisão lineares do tipo $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$, 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, sendo uma medida da confiança da previsão do classificador. Para um problema binário, em que $y_i \in \{-1, +1\}$, dada uma função f e um exemplo $\mathbf{x}_i$, a margem $\varrho(f(\mathbf{x}_i), y_i)$ com que esse dado é classificado por f pode ser calculada pela Equação 2.4 (Smola et al., 1999). Logo, um valor negativo de $\varrho(\mathbf{x}_i, y_i)$ denota uma classificação incorreta.

$$\varrho(f(\mathbf{x}_i), y_i) = y_i f(\mathbf{x}_i) \quad (2.4)$$

Para obter a margem geométrica de um dado $\mathbf{x}_i$, a qual mede efetivamente a distância de $\mathbf{x}_i$ à fronteira de decisão, divide-se o termo à direita da Equação 2.4 pela norma de $\mathbf{w}$, ou seja, por $\|\mathbf{w}\|$ (Smola & Schölkopf, 2002). Para exemplos incorretamente classificados, o valor obtido equivale à distância com sinal negativo. Para realizar uma diferenciação, a

margem da Equação 2.4 será referenciada como margem de confiança.

A partir do conceito introduzido, é possível definir o erro marginal de uma função f ($R_\rho(f)$) 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 2.5) (Smola et al., 1999).

$$R_\rho(f) = \frac{1}{n} \sum_{i=1}^n I(y_i f(\mathbf{x}_i) < \rho) \quad (2.5)$$

Na Equação 2.5, $I(q) = 1$ 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 F correspondendo à classe de funções lineares $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ com $\|\mathbf{x}\| \leq R$ e $\|\mathbf{w}\| \leq 1$, o seguinte limite se aplica (Smola et al., 1999):

$$R(f) \leq R_\rho(f) + \sqrt{\frac{c}{n} \left(\frac{R^2}{\rho^2} \log^2 \left(\frac{n}{\rho} \right) + \log \left(\frac{1}{\theta} \right) \right)} \quad (2.6)$$

Como na Equação 2.3, tem-se na Expressão 2.6 novamente o erro esperado limitado pela soma de uma medida de erro no conjunto de treinamento, neste caso o erro marginal, a um termo de capacidade. A interpretação do presente limite é de que uma maior margem ρ implica em 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 teste e de treinamento, respectivamente. Esse hiperplano é denominado ótimo (Smola & Schölkopf, 2002).

Existem diversos outros limites reportados na literatura, assim como outros tipos de medida de complexidade de uma classe de funções (Müller et al., 2001). Um exemplo é a dimensão *fat-shattering*, que caracteriza o poder de um conjunto de funções em separar os dados com uma margem ρ (Shawe-Taylor et al., 1998). Os limites apresentados anteriormente, embora possam ser considerados simplificados, provêm uma base teórica suficiente à compreensão das SVMs.

2.3 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 dados pertencentes a duas classes. A primeira formulação, mais simples, lida com problemas linearmente separáveis, definidos adiante (Boser et al., 1992). Essa formulação foi poste-

riormente estendida para definir fronteiras lineares sobre conjuntos de dados mais gerais (Cortes & Vapnik, 1995). A partir desses conceitos iniciais, na Seção 2.4 descreve-se a obtenção de fronteiras não lineares com SVMs.

2.3.1 SVMs com Margens Rígidas

As SVMs lineares com margens rígidas definem fronteiras lineares a partir de dados linearmente separáveis. Seja T um conjunto de treinamento com n dados $\mathbf{x}_i \in X$ e seus respectivos rótulos $y_i \in Y$, em que X constitui o espaço dos dados e $Y = \{-1, +1\}$. T é linearmente separável se é possível separar os dados das classes $+1$ e -1 por um hiperplano (Smola & Schölkopf, 2002).

Classificadores que separam os dados por meio de um hiperplano são denominados lineares (Campbell, 2000). A equação de um hiperplano é apresentada na Equação 2.7, 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 $b \in \mathbb{R}$.

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0 \quad (2.7)$$

Essa equação divide o espaço dos dados X em duas regiões: $\mathbf{w} \cdot \mathbf{x} + b > 0$ e $\mathbf{w} \cdot \mathbf{x} + b < 0$. Uma função sinal $g(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$ pode então ser empregada na obtenção das classificações, conforme ilustrado na Equação 2.8 (Smola et al., 1999).

$$g(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \begin{cases} +1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases} \quad (2.8)$$

A partir de $f(\mathbf{x})$, é possível obter um número infinito de hiperplanos equivalentes, pela multiplicação de $\mathbf{w}$ e b por uma mesma constante (Passerini, 2004). Define-se o hiperplano canônico em relação ao conjunto T como aquele em que $\mathbf{w}$ e b são escalados de forma que os exemplos mais próximos ao hiperplano $\mathbf{w} \cdot \mathbf{x} + b = 0$ satisfaçam a Equação 2.9 (Müller et al., 2001).

$$|\mathbf{w} \cdot \mathbf{x}_i + b| = 1 \quad (2.9)$$

Essa forma implica nas inequações 2.10, resumidas na Expressão 2.11.

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i + b \geq +1 & \text{se } y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 & \text{se } y_i = -1 \end{cases} \quad (2.10)$$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad \forall (\mathbf{x}_i, y_i) \in T \quad (2.11)$$

Seja $\mathbf{x}_1$ um ponto no hiperplano $H_1: \mathbf{w} \cdot \mathbf{x} + b = +1$ e $\mathbf{x}_2$ um ponto no hiperplano $H_2: \mathbf{w} \cdot \mathbf{x} + b = -1$, conforme ilustrado na Figura 2.6. Projetando $\mathbf{x}_1 - \mathbf{x}_2$ na direção de $\mathbf{w}$, perpendicular ao hiperplano separador $\mathbf{w} \cdot \mathbf{x} + b = 0$, é possível obter a distância entre

os hiperplanos H_1 e H_2 (Campbell, 2000). Essa projeção é apresentada na Equação 2.12.

$$(\mathbf{x}_1 - \mathbf{x}_2) \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \frac{(\mathbf{x}_1 - \mathbf{x}_2)}{\|\mathbf{x}_1 - \mathbf{x}_2\|} \right) \quad (2.12)$$

Figura 2.6: Cálculo da distância d entre os hiperplanos H_1 e H_2 (Hearst et al., 1998)

Tem-se que $\mathbf{w} \cdot \mathbf{x}_1 + b = +1$ e $\mathbf{w} \cdot \mathbf{x}_2 + b = -1$. A diferença entre essas equações fornece $\mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 2$ (Hearst et al., 1998). Substituindo esse resultado na Equação 2.12, tem-se:

$$\frac{2(\mathbf{x}_1 - \mathbf{x}_2)}{\|\mathbf{w}\| \|\mathbf{x}_1 - \mathbf{x}_2\|} \quad (2.13)$$

Como deseja-se obter o comprimento do vetor projetado, toma-se a norma da Equação 2.13, obtendo:

$$\frac{2}{\|\mathbf{w}\|} \quad (2.14)$$

Essa é a distância d , ilustrada na Figura 2.6, entre os hiperplanos H_1 e H_2 , paralelos ao hiperplano separador. Como $\mathbf{w}$ e b foram escalados de forma a não haver exemplos entre H_1 e H_2 , $\frac{1}{\|\mathbf{w}\|}$ é a distância mínima entre o hiperplano separador e os dados de treinamento. Essa distância é definida como a margem geométrica do classificador (Campbell, 2000).

A partir das considerações anteriores, verifica-se que a maximização da margem de separação dos dados em relação a $\mathbf{w} \cdot \mathbf{x} + b = 0$ pode ser obtida pela minimização de $\|\mathbf{w}\|$ (Burges, 1998). Dessa forma, recorre-se ao seguinte problema de otimização (Smola & Schölkopf, 2002):

$$\text{Minimizar}_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad (2.15)$$

$$\text{Com as restrições: } y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0, \quad \forall i = 1, \dots, n \quad (2.16)$$

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.

O problema de otimização obtido é quadrático, cuja solução possui uma ampla e estabelecida teoria matemática (Smola & Schölkopf, 2002). 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 (Passerini, 2004). 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 2.17) (Smola & Schölkopf, 2002).

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1) \quad (2.17)$$

A função Lagrangiana deve ser minimizada, o que implica em maximizar as variáveis α_i e minimizar $\mathbf{w}$ e b (Müller et al., 2001). Tem-se então um ponto de sela, no qual:

$$\frac{\partial L}{\partial b} = 0 \quad \text{e} \quad \frac{\partial L}{\partial \mathbf{w}} = 0 \quad (2.18)$$

A resolução dessas equações leva aos resultados representados nas expressões 2.19 e 2.20.

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (2.19)$$

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (2.20)$$

Substituindo as equações 2.19 e 2.20 na Equação 2.17, obtém-se o seguinte problema de otimização:

$$\text{Maximizar}_{\boldsymbol{\alpha}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (2.21)$$

$$\text{Com as restrições:} \quad \begin{cases} \alpha_i \geq 0, \quad \forall i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad (2.22)$$

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 dados, o que será útil na posterior não-linearização das SVMs (Seção 2.4). É interessante observar também que o problema dual é formulado utilizando apenas os dados de treinamento e os seus rótulos.

Seja $\boldsymbol{\alpha}^*$ 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 2.20. O parâmetro b^* é definido por α^* e por condições de Kühn-Tucker, provenientes da teoria de otimização com restrições e que devem ser satisfeitas no ponto ótimo. Para o problema dual formulado, tem-se (Pontil & Verri, 1998):

$$\alpha_i^* (y_i (\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1) = 0, \forall i = 1, \dots, n \quad (2.23)$$

Observa-se nessa equação que α_i^* pode ser diferente de 0 somente para os dados que se encontram sobre os hiperplanos H_1 e H_2 . Estes são os exemplos que se situam mais próximos ao hiperplano separador, exatamente sobre as margens. Para os outros casos, a condição apresentada na Equação 2.23 é obedecida apenas com $\alpha_i^* = 0$. Esses pontos não participam então do cálculo de $\mathbf{w}^*$ (Equação 2.20). Os dados que possuem $\alpha_i^* > 0$ são denominados vetores de suporte (*Support Vectors* - SVs) e podem ser considerados os dados mais informativos do conjunto de treinamento (Burgess, 1998). O valor de b^* é calculado a partir dos SVs e das condições representadas na Equação 2.23 (Smola & Schölkopf, 2002). Computa-se a média apresentada na Equação 2.24 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.

$$b^* = \frac{1}{n_{SV}} \sum_{\mathbf{x}_j \in SV} \frac{1}{y_j} - \mathbf{w}^* \cdot \mathbf{x}_j = \frac{1}{n_{SV}} \sum_{\mathbf{x}_j \in SV} \left(\frac{1}{y_j} - \sum_{\mathbf{x}_i \in SV} \alpha_i^* y_i \mathbf{x}_i \cdot \mathbf{x}_j \right) \quad (2.24)$$

Como resultado final, tem-se o classificador $g(\mathbf{x})$ apresentado na Equação 2.25.

$$g(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn} \left(\sum_{\mathbf{x}_i \in SV} y_i \alpha_i^* \mathbf{x}_i \cdot \mathbf{x} + b^* \right) \quad (2.25)$$

2.3.2 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 dados ou à própria natureza do problema, que pode ser não linear. Nesta seção as SVMs lineares de margens rígidas são estendidas para lidar com conjuntos de treinamento mais gerais. Para realizar essa tarefa, permite-se que alguns dados possam violar a restrição da Equação 2.16. Isso é feito com a introdução de variáveis de folga ξ_i , para todo $i = 1, \dots, n$ (Smola et al., 1999). Essas variáveis relaxam as restrições impostas ao problema de otimização primal, que se tornam (Smola & Schölkopf, 2002):

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \dots, n \quad (2.26)$$

A aplicação desse procedimento suaviza as margens do classificador linear, permitindo que alguns dados permaneçam entre os hiperplanos H_1 e H_2 e também a ocorrência de alguns erros de classificação. Por esse motivo, as SVMs obtidas neste 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 (Burgess, 1998). Para levar em consideração esse termo, minimizando assim o erro sobre os dados de treinamento, a função objetivo da Equação 2.15 é reformulada como (Burgess, 1998):

$$\underset{\mathbf{w}, b, \boldsymbol{\xi}}{\text{Minimizar}} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \left(\sum_{i=1}^n \xi_i \right) \quad (2.27)$$

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 (Passerini, 2004). 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 dado entre as margens. Tem-se então uma formulação de acordo com os princípios da TAE discutidos na Seção 2.2.

Novamente o problema de otimização gerado é quadrático, com as restrições lineares apresentadas na Equação 2.26. A 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:

$$\underset{\boldsymbol{\alpha}}{\text{Maximizar}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (2.28)$$

$$\text{Com as restrições: } \begin{cases} 0 \leq \alpha_i \leq C, \quad \forall i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad (2.29)$$

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 2.20. As variáveis ξ_i^* podem ser calculadas pela Equação 2.30 (Cristianini & Shawe-Taylor, 2000).

$$\xi_i^* = \max \left\{ 0, 1 - y_i \sum_{j=1}^n y_j \alpha_j^* \mathbf{x}_j \cdot \mathbf{x}_i + b^* \right\} \quad (2.30)$$

A variável b^* provém novamente de $\boldsymbol{\alpha}^*$ e de condições de Kühn-Tucker, que neste caso são (Pontil & Verri, 1998):

$$\alpha_i^* (y_i (\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1 + \xi_i^*) = 0 \quad (2.31)$$

$$(C - \alpha_i^*) \xi_i^* = 0 \quad (2.32)$$

Como nas SVMs de margens rígidas, os pontos $\mathbf{x}_i$ para os quais $\alpha_i^* > 0$ são denominados vetores de suporte (SVs), sendo os dados que participam da formação do hiperplano separador. Porém, neste caso, pode-se distinguir tipos distintos de SVs (Pontil & Verri, 1998). Se $\alpha_i^* < C$, pela Equação 2.32, $\xi_i^* = 0$ e então, da Equação 2.31, estes SVs encontram-se sobre as margens e também são denominados livres. Os SVs para os quais $\alpha_i^* = C$ podem representar três casos (Pontil & Verri, 1998): 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 2.7 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 dados, em branco, são corretamente classificados e encontram-se fora das margens, possuindo $\xi_i^* = 0$ e $\alpha_i^* = 0$.

Figura 2.7: Tipos de SVs: livres (cor cinza) e limitados (cor preta) (Passerini, 2004)

Para calcular b^* , computa-se a média da Equação 2.24 sobre todos SVs $\mathbf{x}_j$ entre as margens, ou seja, com $\alpha_i^* < C$ (Smola & Schölkopf, 2002).

Tem-se como resultado final a mesma função de classificação representada na Equação 2.25, porém neste caso as variáveis α_i^* são determinadas pela solução da Expressão 2.28.

2.4 SVMs Não Lineares

As SVMs lineares são eficazes na classificação de conjuntos de dados linearmente separáveis ou que possuam uma distribuição aproximadamente linear, sendo que 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 2.8a, 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 $\Phi : X \rightarrow \mathfrak{S}$ um mapeamento, em que X é o espaço de entradas e $\mathfrak{S}$ denota o espaço

Figura 2.8: (a) Conjunto de dados não linear; (b) Fronteira não linear no espaço de entradas; (c) Fronteira linear no espaço de características (Müller et al., 2001)

de características. A escolha apropriada de Φ faz com que o conjunto de treinamento mapeado em $\mathfrak{S}$ 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 $\mathfrak{S}$ no qual com alta probabilidade os dados 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.

Para ilustrar esses conceitos, considere o conjunto de dados apresentado na Figura 2.8a (Müller et al., 2001). Transformando os dados de $\mathbb{R}^2$ para $\mathbb{R}^3$ com o mapeamento representado na Equação 2.33, o conjunto de dados não linear em $\mathbb{R}^2$ torna-se linearmente separável em $\mathbb{R}^3$ (Figura 2.8c). É possível então encontrar um hiperplano capaz de separar esses dados, descrito na Equação 2.34. Pode-se verificar que a função apresentada, embora linear em $\mathbb{R}^3$ (Figura 2.8c), corresponde a uma fronteira não linear em $\mathbb{R}^2$ (Figura 2.8b).

$$\Phi(\mathbf{x}) = \Phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \quad (2.33)$$

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + b = w_1x_1^2 + w_2\sqrt{2}x_1x_2 + w_3x_2^2 + b = 0 \quad (2.34)$$

Logo, mapea-se inicialmente os dados para um espaço de maior dimensão utilizando Φ e aplica-se a SVM linear sobre este espaço. Essa 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 2.28, conforme ilustrado a seguir:

$$\underset{\boldsymbol{\alpha}}{\text{Maximizar}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\boldsymbol{\Phi}(\mathbf{x}_i) \cdot \boldsymbol{\Phi}(\mathbf{x}_j)) \quad (2.35)$$

De forma semelhante, o classificador extraído se torna:

$$g(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn}\left(\sum_{\mathbf{x}_i \in \text{SV}} \alpha_i^* y_i \boldsymbol{\Phi}(\mathbf{x}_i) \cdot \boldsymbol{\Phi}(\mathbf{x}) + b^*\right) \quad (2.36)$$

Em que b^* é adaptado da Equação 2.24 também aplicando o mapeamento aos dados:

$$b^* = \frac{1}{n_{\text{SV}:\alpha^* < C}} \sum_{\mathbf{x}_j \in \text{SV}:\alpha_j^* < C} \left(\frac{1}{y_j} - \sum_{\mathbf{x}_i \in \text{SV}} \alpha_i^* y_i \boldsymbol{\Phi}(\mathbf{x}_i) \cdot \boldsymbol{\Phi}(\mathbf{x}_j) \right) \quad (2.37)$$

Como $\mathfrak{S}$ pode ter dimensão muito alta (até mesmo infinita), a computação de $\boldsymbol{\Phi}$ pode ser extremamente custosa ou inviável. Porém, percebe-se pelas equações 2.35, 2.36 e 2.37 que a única informação necessária sobre o mapeamento é de como realizar o cálculo de produtos escalares entre os dados 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 $\mathbf{x}_j$ do espaço de entradas e computa o produto escalar desses dados no espaço de características (Herbrich, 2001). Tem-se então:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\Phi}(\mathbf{x}_i) \cdot \boldsymbol{\Phi}(\mathbf{x}_j) \quad (2.38)$$

Para o mapeamento apresentado na Equação 2.33 e dois dados $\mathbf{x}$ e $\mathbf{x}'$ em $\mathbb{R}^2$, por exemplo, o Kernel é dado por (Hearst et al., 1998):

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2 \quad (2.39)$$

É comum empregar a função Kernel sem conhecer o mapeamento $\boldsymbol{\Phi}$, 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 2.35 e também que o Kernel represente mapeamentos nos quais seja possível o cálculo de produtos escalares conforme a Equação 2.38, utiliza-se funções Kernel que seguem as condições estabelecidas pelo teorema de Mercer (Mercer, 1909; Smola et al., 1999). De forma simplificada, um Kernel que satisfaz as condições de Mercer é caracterizado por dar origem a matrizes positivas semi-definidas $\mathbf{K}$, 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 Gaussianos ou RBF (*Radial-Basis Function*) e os Sigmoidais, listados na Tabela 2.1. 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 . Os Kernels Polinomiais com $d = 1$ também são denominados lineares.

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
Gaussiano	$\exp \left(-\sigma \ \mathbf{x}_i - \mathbf{x}_j\ ^2 \right)$	σ
Sigmoidal	$\tanh (\delta (\mathbf{x}_i \cdot \mathbf{x}_j) + \kappa)$	δ e κ

Tabela 2.1: Funções Kernel mais comuns (Chang & Lin, 2004b)

2.5 Considerações Finais

Neste capítulo foram descritos os conceitos básicos a respeito das SVMs, os quais também podem ser consultados em (Lorena & Carvalho, 2003a,d). Com princípios embasados na teoria de aprendizado estatístico, essa técnica de AM se caracteriza por apresentar uma boa capacidade de generalização. As SVMs também são robustas diante de dados de grande dimensão, sobre os quais outras técnicas de aprendizado comumente obtêm classificadores super ou sub ajustados. Outra característica atrativa é a convexidade do problema de otimização formulado em seu treinamento, que implica na existência de um único mínimo global. Essa é uma vantagem das SVMs sobre, por exemplo, as Redes Neurais Artificiais (RNAs) Perceptron Multicamadas (*Multilayer Perceptron* - MLP) (Haykin, 1999; Braga et al., 2000), 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.

Observou-se no decorrer deste capítulo 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 & Shawe-Taylor, 2000).

O presente capítulo também se limitou a apresentar a formulação original das SVMs,

a qual é capaz de lidar apenas com problemas de classificação binários. Os próximos capítulos descrevem uma série de técnicas que podem ser empregadas na generalização das SVMs para a solução de problemas multiclass.

Capítulo 3

Estratégias Multiclasses

Diversas técnicas de AM são originalmente formuladas para a solução de problemas de classificação binários. Entre elas pode-se citar as SVMs, descritas no capítulo anterior. Muitos problemas de classificação, contudo, apresentam mais que duas classes.

A generalização de técnicas de classificação binária para problemas multiclasses pode ser realizada basicamente por meio de duas estratégias. A primeira consiste na combinação de preditores gerados em subproblemas binários, enquanto na segunda realiza-se adaptações nos algoritmos originais das técnicas consideradas. A extensão direta de um algoritmo binário a uma versão multiclasses nem sempre é possível ou fácil de realizar (Passerini, 2004). Para as SVMs, em particular, Hsu & Lin (2002) observaram que a reformulação dessa técnica em versões multiclasses leva a algoritmos computacionalmente caros. É comum recorrer-se então à alternativa de decompor o problema multiclasses em subproblemas binários, uma estratégia denominada decomposicional.

Na Seção 3.1 é apresentada uma revisão das principais técnicas da literatura que seguem uma estratégia decomposicional na solução do problema multiclasses. A solução direta do problema multiclasses por meio de reformulações no algoritmo das SVMs é abordada na Seção 3.2. Na Seção 3.3 este capítulo é concluído com algumas considerações finais.

3.1 Estratégias Decomposicionais

A forma mais direta de geração de classificadores multiclasses a partir de técnicas de classificação binária consiste em decompor o problema multiclasses em subproblemas binários. As saídas dos preditores binários gerados na solução de cada um desses subproblemas são então combinadas na obtenção do classificador final.

Segundo Moreira & Mayoraz (1997), existe uma série de motivações para empregar estratégias decomposicionais na solução de problemas multiclasses. 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. Mesmo que o algoritmo seja capaz de trabalhar com problemas 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.

Pimenta (2005) também aponta vantagens no uso da decomposição em problemas cujos erros de classificação das diferentes classes possuem pesos distintos. Pode-se assim gerar preditores binários de maneira a impor preferências para algumas classes. Esse autor menciona ainda o fato dos subproblemas de classificação gerados em uma decomposição serem independentes, podendo ser solucionados paralelamente.

O emprego de técnicas decompositivas envolve dois passos. No primeiro realiza-se a divisão do problema multiclasses em subproblemas binários, determinando assim os classificadores binários a serem gerados. A segunda etapa pode ser denominada reconstrução e refere-se à forma como as saídas dos classificadores binários são combinadas na determinação da classe de um exemplo (Mayoraz & Moreira, 1996).

Nesta seção são apresentadas diversas estratégias decompositivas. De forma geral, é possível agrupá-las em dois tipos: baseadas em matrizes de códigos e hierárquicas. Cada um desses tipos é descrito a seguir.

3.1.1 Estratégias Baseadas em Matrizes de Códigos

A abordagem de matrizes de códigos unifica uma série de técnicas decompositivas (Allwein et al., 2000). Essas estratégias podem ser representadas de maneira geral 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 multiclasses. Na Figura 3.1 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.

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 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 f_j . Classificadores binários são então treinados de forma a aprender os rótulos representados nas colunas de $\mathbf{M}$.

Figura 3.1: Exemplo de matriz de códigos para um problema com quatro classes

► Etapa de Reconstrução

Um novo dado $\mathbf{x}$ pode ser classificado avaliando-se as predições dos l classificadores, que geram um vetor $\mathbf{f}(\mathbf{x})$ de tamanho l na forma $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_l(\mathbf{x}))$. 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 $\mathbf{f}(\mathbf{x})$ de acordo com alguma medida. Esse processo de reconstrução também é referenciado como decodificação (Passerini et al., 2004). 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 3.1, em que $f(\mathbf{x})$ representa o classificador multiclases final e d denota uma função de decodificação.

$$f(\mathbf{x}) = \arg \min_{1 \leq q \leq k} (d(\mathbf{m}_q, \mathbf{f}(\mathbf{x}))) \quad (3.1)$$

Caso mais de um código se encontre mais próximo a $\mathbf{f}(\mathbf{x})$, tem-se a ocorrência de uma classificação desconhecida. Escolhe-se então uma das classes envolvidas no empate aleatoriamente ou com o uso de alguma informação a priori (Rätsch et al., 2003).

Existem diversas funções de decodificação que podem ser utilizadas na integração dos classificadores binários na Equação 3.1 (Passerini et al., 2004). A mais simples é a de *Hamming*, que conta o número de ocorrências diferentes entre o vetor $\mathbf{f}(\mathbf{x})$ e cada um dos códigos. Essa função pode ser visualizada na Equação 3.2. 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 (Passerini, 2004).

$$d_H(\mathbf{m}_q, \mathbf{f}(\mathbf{x})) = \sum_{i=1}^l \frac{1 - \text{sgn}(m_{qi} * f_i(\mathbf{x}))}{2} \quad (3.2)$$

Para as SVMs, Allwein et al. (2000) sugerem o uso de uma decodificação baseada no princípio de maximização de margens seguido por essa técnica. Recordando a teoria apresentada no Capítulo 2, tem-se que o problema de otimização das SVMs é dado pela

Equação 3.3, em que $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) + b$.

$$\begin{aligned} &\text{Minimizar } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ &\text{Com as restrições: } \begin{cases} y_i f(\mathbf{x}_i) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \end{aligned} \quad (3.3)$$

Pode-se simplificar esse problema à forma apresentada na Equação 3.4, em que $(z)_+ = \max\{0, z\}$ (Allwein et al., 2000).

$$\text{Minimizar } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n (1 - y_i f(\mathbf{x}_i))_+ \quad (3.4)$$

Como $y_i f(\mathbf{x}_i)$ denota a margem de confiança com que $\mathbf{x}_i$ é classificado por f (Equação 2.4), o segundo termo na soma da Equação 3.4 representa uma função de custo baseada nas margens, a qual é minimizada pelas SVMs em seu treinamento. Pode-se interpretar a minimização desse termo como um estímulo a que cada dado seja classificado com margem elevada, pois dessa forma garante-se uma contribuição nula na soma realizada. A partir desse raciocínio, Allwein et al. (2000) desenvolveram uma função de decodificação que neste documento será referenciada como baseada em margem. Essa função é apresentada na Equação 3.5. A motivação de sua formulação reside no fato da distância de *Hamming* ignorar a função de custo utilizada no treinamento das SVMs, assim como confidências ligadas às previsões feitas por esses tipos de classificadores, as quais podem ser quantificadas pelas margens com que o exemplo é classificado.

$$d_M(\mathbf{m}_q, \mathbf{f}(\mathbf{x})) = \sum_{i=1}^l \max\{0, 1 - (m_{qi} * f_i(\mathbf{x}))\} \quad (3.5)$$

Esse mesmo procedimento pode ser empregado com outras técnicas que utilizam o princípio de maximização de margens, como as descritas em (Lorena & Carvalho, 2003a), bastando para isto adaptar a função de custo baseada em margens a cada técnica (Allwein et al., 2000).

No exemplo da Figura 3.2, baseado em um exemplo fornecido em (Allwein et al., 2000), ilustra-se como as decodificações de *Hamming* e baseada em margens podem levar a classificações diferentes. Tem-se um problema com quatro classes solucionado com o uso de sete classificadores binários. Enquanto a função de *Hamming* leva à predição da classe 3, o uso das margens com que o dado é classificado indica uma maior confiança quanto à predição da classe 4.

Existem várias outras funções de decodificação que podem ser empregadas na integração dos classificadores binários (Windeatt & Ghaderi, 2003; Passerini et al., 2004). Em um trabalho recente, Passerini et al. (2004) apresentaram uma função baseada em probabilidades condicionais para as SVMs, em que, a partir das margens de classificação, se obtém probabilidades de que o dado pertença a cada classe. A sua motivação reside

saída	Decodificações	
$\mathbf{f}(\mathbf{x}) = (0,5 \quad -7 \quad -1 \quad -2 \quad -10 \quad -12 \quad 9)$	<i>Hamming</i>	Baseada em margens
Matriz de códigos	$d_H(\mathbf{m}_1, \mathbf{f}(\mathbf{x})) = 3,5$ $d_H(\mathbf{m}_2, \mathbf{f}(\mathbf{x})) = 4,5$ $d_H(\mathbf{m}_3, \mathbf{f}(\mathbf{x})) = 2,0$ $d_H(\mathbf{m}_4, \mathbf{f}(\mathbf{x})) = 2,5$	$d_M(\mathbf{m}_1, \mathbf{f}(\mathbf{x})) = 23,5$ $d_M(\mathbf{m}_2, \mathbf{f}(\mathbf{x})) = 37,5$ $d_M(\mathbf{m}_3, \mathbf{f}(\mathbf{x})) = 21,5$ $d_M(\mathbf{m}_4, \mathbf{f}(\mathbf{x})) = 4,5$
$\begin{pmatrix} -1 & 0 & -1 & -1 & +1 & -1 & -1 \\ +1 & -1 & 0 & +1 & +1 & +1 & -1 \\ +1 & +1 & -1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & 0 & -1 & -1 & +1 \end{pmatrix}$	$\Downarrow$ classe = 3	$\Downarrow$ classe = 4

Figura 3.2: Resultados da aplicação das decodificações de *Hamming* e baseada em margens

no fato da confiança representada pela margem de classificação ser na realidade uma medida relativa, uma vez que o intervalo das margens pode variar muito entre diferentes preditores. Embora interessante, essa função não foi empregada no presente trabalho, pois o processo de obtenção das probabilidades é computacionalmente custoso frente às decodificações previamente discutidas. Ele envolve o ajuste de uma função paramétrica às saídas das SVMs, cujos parâmetros devem ser estimados em um procedimento de *3-fold cross-validation*¹ sobre o conjunto de treinamento. No caso das funções apresentadas anteriormente, esse tipo de ajuste não é necessário.

► Etapa de Decomposição

As descrições realizadas anteriormente referiam-se à etapa de reconstrução das saídas multiclasse a partir dos preditores binários representados na matriz $\mathbf{M}$. Na decomposição do problema, diversas alternativas podem ser empregadas. 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 & Moreira, 1996). Um exemplo de matriz compacta para um problema com quatro classes é apresentado na Figura 3.3a.

$\begin{pmatrix} +1 & +1 \\ +1 & -1 \\ -1 & +1 \\ -1 & -1 \end{pmatrix}$	$\begin{pmatrix} +1 & +1 & +1 & +1 & +1 & +1 \\ -1 & -1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & +1 & -1 & -1 \\ -1 & +1 & -1 & +1 & -1 & +1 \end{pmatrix}$
(a)	(b)
$\begin{pmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{pmatrix}$	$\begin{pmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{pmatrix}$
(c)	(d)

Figura 3.3: Matrizes de diferentes decomposições para um problema com quatro classes

¹O procedimento de *cross-validation* encontra-se descrito no Apêndice A.

O número total de diferentes preditores binários para um problema com k classes é $0,5(3^k + 1) - 2^k$, considerando que $f = -f$, ou seja, que a inversão das classes positivas e negativas produz o mesmo classificador (Mayoraz & Moreira, 1996). Desses, $2^{k-1} - 1$ incluem todas as classes simultaneamente, ou seja, possuem somente rótulos $+1$ e -1 , sem o elemento 0 (Mayoraz & Moreira, 1996). Para quatro classes, tem-se nesse caso a matriz representada na Figura 3.3b.

Entre as estratégias decomposicionais mais comuns encontradas na literatura estão a um-contra-todos (1ct) (Vapnik, 1995), a todos-contra-todos (tct) (Knerr et al., 1990; Hastie & Tibshirani, 1998) e a baseada em códigos de correção de erros (*Error Correcting Output Codes* - ECOC) (Dietterich & Bariki, 1995), que são descritas a seguir.

Na estratégia 1ct, dado um problema com k classes, k classificadores binários $f_i(\mathbf{x})$ 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 1ct para um problema de quatro classes é apresentada na Figura 3.3c. Usualmente, a função de integração utilizada com a decomposição 1ct é a baseada em margens, que equivale à escolha da classe correspondente ao classificador que produz a maior saída (Rifkin & Klautau, 2004).

A decomposição 1ct pode apresentar desvantagens quando a proporção de exemplos de uma classe é muito pequena em relação à do conjunto formado pelos dos dados das outras classes. Esse tipo de desbalanceamento pode dificultar a indução de um preditor que apresente bom desempenho no reconhecimento da classe considerada.

Na decomposição todos-contra-todos (tct), 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 3.3d tem-se uma matriz tct para um problema com quatro classes.

Embora o número de classificadores gerados na decomposição tct 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.

A integração usual dos classificadores gerados na estratégia tct é realizada por meio de um esquema de votação por maioria (Kreβel, 1999). 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. A implementação dessa solução corresponde ao uso da função de *Hamming* sobre a matriz de códigos descrita anteriormente.

Um problema apontado na decomposição 1ct é que a resposta de um preditor para um par de classes (i, j) na realidade não fornece informação alguma quando o exemplo não pertence às classes i ou j (Alpaydin & Mayoraz, 1999).

Em uma estratégia decomposicional alternativa, Dietterich & Bariki (1995) propuseram o uso de códigos de correção de erros para representar as k classes de um problema. Para tal, os autores sugerem que o valor do número de classificadores l seja maior que o número mínimo necessário para diferenciar cada classe unicamente. Os bits adicionais introduzem uma redundância na codificação das classes e têm a utilidade de prover ao sistema a capacidade de se recuperar de eventuais erros cometidos por alguns preditores no processo de classificação de um novo exemplo. Por esse motivo, essa técnica possui a denominação de decomposição por códigos de correção de erros (*Error Correcting Output Codes* - ECOC). As matrizes de códigos nesse caso assumem valores em $\{-1, +1\}$ e utiliza-se a função de *Hamming* na decodificação das saídas dos classificadores.

Com esse procedimento de correção de erros, Dietterich & Bariki (1995) sugerem que a solução de um problema multiclases por uma técnica de AM seja vista como uma tarefa de comunicação. Dado um novo exemplo, a sua classe correta é transmitida por meio de um canal, o qual é constituído de seus atributos, dos dados de treinamento e do algoritmo de aprendizado. Devido a erros que podem estar presentes nos atributos do exemplo, nos dados de treinamento utilizados ou em falhas no processo de aprendizado, a informação pode ser corrompida. Para prover ao sistema a capacidade de se recuperar desses erros de transmissão, a classe é codificada por um código de correção de erros e cada bit do mesmo é transmitido separadamente, ou seja, por execuções separadas do algoritmo de aprendizado.

Para possibilitar a correção de erros, Dietterich & Bariki (1995) sugerem 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 multiclases final é capaz de corrigir ao menos $\lfloor \frac{d_l-1}{2} \rfloor$ bits incorretos em uma predição. Como segundo a função de decodificação de *Hamming* cada bit incorreto implica em se afastar em uma unidade do código da classe correta, cometendo $\lfloor \frac{d_l-1}{2} \rfloor$ erros, o código mais próximo ainda será o correto (Dietterich & Bariki, 1995). Segundo esse princípio, a codificação 1ct é incapaz de se recuperar de qualquer erro, uma vez que nesta $d_l = 2$.

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, $f = -f$), como no caso das SVMs, então deve-se também fazer com que a distância de *Hamming* entre cada coluna e o complemento das outras seja grande.

Com base nessas observações, Dietterich & 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 da combinação de $2^{k-1} - 1$ classificadores binários unicamente com rótulos $+1$ e -1 , conforme ilustrado na Figura 3.3b 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 & Ray-Chaudhuri, 1960), os quais provêm da teoria de códigos de correção de erros em comunicação.

Em um trabalho recente, Pimenta (2005) propôs um algoritmo para o desenvolvimento de ECOCs que, usados com SVMs, apresentaram resultados compatíveis ou superiores aos de estratégias decomposicionais tradicionais. Nele é proposta uma função para a avaliação da qualidade de ECOCs segundo suas propriedades de correção de erros. Um algoritmo iterativo de perseguição é então utilizado na construção dos ECOCs. Esse algoritmo adiciona ou retira colunas de um ECOC inicial, procurando maximizar a sua qualidade. Pimenta (2005) também apresentou um método para a determinação do número de colunas no ECOC, por meio do exame de uma função de avaliação baseada no número de erros corrigíveis por ECOCs de diferentes tamanhos.

Em (Allwein et al., 2000) é apontado que, apesar dos códigos gerados pelo ECOC possuírem boa propriedade de correção, vários dos subproblemas binários criados podem ser difíceis de aprender. Por esse motivo, as técnicas mais simples 1ct e tct têm apresentado resultados comparáveis ou superiores ao ECOC em várias aplicações (Allwein et al., 2000; Rifkin & Klautau, 2004).

Uma crítica comum às estratégias 1ct, tct e ECOC é que todas realizam a decomposição do problema a priori, sem levar em consideração as propriedades e as características de cada aplicação (Mayoraz & Moreira, 1996; Mayoraz & Alpaydim, 1998; Alpaydin & Mayoraz, 1999; Allwein et al., 2000; Dekel & Singer, 2003; Rätsch et al., 2003; Ko et al., 2004). Essa é a motivação do presente trabalho para a proposição de uma das técnicas discutidas no Capítulo 4. Alguns trabalhos relacionados a esse tema são brevemente discutidos a seguir.

Em (Mayoraz & Moreira, 1996) é apresentado um algoritmo iterativo para o desenvolvimento de matrizes de códigos levando em consideração três critérios. Os dois primeiros são os mesmos requeridos por Dietterich & Bariki (1995) na construção de códigos de correção de erros, ou seja, que a distância de *Hamming* mínima entre as linhas da matriz seja grande e que as distâncias entre cada coluna e as outras, assim como aos complementos das outras, sejam grandes. O terceiro critério é que cada coluna inserida na matriz seja pertinente de acordo com a posição das classes no espaço de entradas. Uma partição binária das classes é considerada pertinente se ela é fácil de aprender. Com isso, a construção dos códigos leva em consideração as características dos dados e também da técnica de aprendizado utilizada na geração dos preditores binários. A maior contribuição nos resultados alcançados por esse procedimento foi a obtenção de classificadores multiclases

mais simples que os do ECOC.

Em (Alpaydin & Mayoraz, 1999) é proposta a combinação de preditores binários lineares na obtenção de um classificador multiclases final não linear. A decomposição do problema é integrada ao processo de treinamento dos classificadores lineares. Chega-se como resultado ao treinamento de uma RNA do tipo MLP, em que a primeira camada de pesos representa os parâmetros dos classificadores lineares, os nós intermediários correspondem aos preditores lineares e a camada de pesos de saída equivale à matriz de códigos. Essa rede tem a arquitetura e os pesos da segunda camada iniciados conforme alguma matriz de códigos usual, como a do ECOC. Como resultado, otimiza-se os parâmetros dos classificadores lineares e também a matriz de códigos utilizada na decomposição do problema com base nos dados de treinamento. Comparado a empregar as decomposições ECOC, 1ct e tct com classificadores lineares, o método proposto obteve alguns ganhos em termos de taxa de acerto.

Em (Dekel & Singer, 2003) é introduzido um algoritmo de aprendizado para problemas multiclases denominado *Bunching*, que durante o seu aprendizado adapta uma matriz de códigos ao problema. Nesse processo, os dados de treinamento e os seus rótulos são transformados para um espaço comum, no qual é também definida uma função que mede a divergência entre vetores. Dado então um novo exemplo, este é mapeado para o espaço comum e a classe prevista é aquela da qual o exemplo está mais próximo nesse novo espaço. No processo de mapeamento são definidas duas matrizes, uma para os dados e outra para os rótulos, sendo que a segunda constitui a matriz de códigos do problema. Essas duas matrizes são iterativamente adaptadas pelo algoritmo de forma a obter um erro mínimo no conjunto de treinamento. Esse erro é medido pela divergência dos dados de treinamento em relação aos seus rótulos no espaço comum. No algoritmo proposto as matrizes de códigos são probabilísticas, ou seja, seus elementos são contínuos e não discretos como no ECOC. De fato, Dekel & Singer (2003) afirmam que o ECOC pode ser visto como um tipo de *Bunching*. Dada então uma matriz de códigos inicial, o algoritmo de *Bunching* iterativamente modifica-a de acordo com o raciocínio descrito anteriormente. Empiricamente, os autores verificaram bons resultados na adaptação de matrizes dos tipos 1ct e aleatórias, utilizando regressores logísticos como classificadores binários.

Em (Rätsch et al., 2003) também é proposta uma metodologia para adaptar os códigos das classes ao problema. Formula-se um problema de otimização no qual os códigos e também pesos para as funções binárias utilizadas no aprendizado dos rótulos representados nos códigos são determinados em conjunto. Utiliza-se o princípio da maximização de margens nesse processo, no sentido de que a diferença entre a distância do vetor de previsões $\mathbf{f}(\mathbf{x})$ ao código da classe correta e a distância de $\mathbf{f}(\mathbf{x})$ à classe equivalente ao segundo código mais próximo seja maximizada. Alguns resultados experimentais iniciais apresentados indicaram uma vantagem da aplicação desse método com Árvores de Decisão em relação ao uso direto dessa técnica de aprendizado na solução multiclases.

Em (Crammer & Singer, 2002) é apresentada uma formulação de uma versão multi-

classes para as SVMs que surgiu a partir de um formalismo para a obtenção de matrizes de códigos a partir de seu desempenho na aplicação multiclasses. Esse trabalho é discutido na Seção 3.2.2.

3.1.2 Estratégias Hierárquicas

Uma maneira alternativa de solucionar um problema multiclasses com preditores binários pode ser conduzida com sua decomposição de uma forma hierárquica. Em diversos domínios, existem hierarquias naturais entre as classes. Um exemplo é a categorização de textos, na qual algumas classes de textos podem ser agrupadas em sub-tópicos, os quais por sua vez também podem ser reunidos em tópicos principais. De forma geral, a introdução de uma hierarquia em uma aplicação multiclasses pode reduzir a complexidade de sua solução. A idéia é realizar inicialmente discriminações mais gerais, as quais são refinadas sucessivamente até a obtenção da classificação final.

Na Figura 3.4 são apresentados dois exemplos de classificadores multiclasses hierárquicos utilizados na solução de um problema com seis classes (Schwenker & Palm, 2001). O classificador da Figura 3.4a possui uma estrutura de árvore direcionada binária, em que exatamente uma aresta chega a cada nó e no máximo duas arestas partem dos mesmos. Na Figura 3.4b tem-se uma estrutura mais geral, de um grafo direcionado acíclico, em que mais de uma aresta pode apontar para um mesmo nó. Uma definição formal de grafos e árvores é apresentada no Capítulo 5. Segundo a 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. Iniciando pelo nó raiz, as classes são particionadas recursivamente em dois subconjuntos, até que todos os subconjuntos gerados contenham apenas uma classe.

Figura 3.4: Classificadores hierárquicos para um problema com seis classes (Schwenker & Palm, 2001)

Na previsão da classe de um novo exemplo, este é inicialmente submetido à raiz da hierarquia. Baseado na decisão desse preditor, um novo nó é visitado. Esse processo é iterado até que uma folha seja atingida, na qual a classe é determinada. Esse é o procedimento de reconstrução comumente empregado na obtenção de classificações com o uso de uma estrutura hierárquica. Os preditores e ramificações percorridos na classificação de um exemplo formam um caminho no classificador hierárquico comumente referenciado como caminho de avaliação do exemplo.

Em geral, pode-se afirmar que as estratégias hierárquicas apresentam tempos menores de predição do que as baseadas em matrizes de códigos anteriormente discutidas. Isso ocorre porque, na classificação de um dado, apenas uma parte dos preditores binários são consultados, enquanto a decodificação das matrizes de códigos necessita das saídas de todos classificadores na decomposição.

Deve-se observar, contudo, que os preditores contidos nas estratégias hierárquicas também podem ser representados por uma matriz de códigos. Entretanto, a forma de integração deles é dependente de cada dado e não envolve a consulta de todos os preditores binários, como na aplicação das funções de decodificação apresentadas na Seção 3.1.1. No caso das estratégias hierárquicas, a decodificação aplicada é conhecida por eliminação (Klautau et al., 2003) e opera 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 um único classificador binário reste, provendo a classificação final.

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, seguindo de estruturas de árvores direcionadas binárias.

► Grafos Direcionados Acíclicos

Em (Platt et al., 2000), é sugerido que os classificadores produzidos pela decomposição tet sejam dispostos em um grafo de decisão direcionado acíclico (*Decision Directed Acyclic Graph* - DDAG). Logo, cada nó do grafo corresponde a um preditor binário para um par de classes. Essa estratégia será referenciada como DDAG neste documento. Platt et al. (2000) determinaram limites na capacidade de generalização desse tipo de estrutura, os quais são baseados no tamanho do grafo e nas margens dos preditores binários. Isso indica que o uso de técnicas que utilizam o princípio de maximização de margens, como as SVMs, pode produzir bons resultados.

Na Figura 3.5 é ilustrado um exemplo de DDAG para um problema com quatro classes. A cada nó está associada uma lista de classes. Inicialmente, todas as classes são candidatas e o nó raiz distingue as classes 1 e 4, equivalentes à primeira e à última classes da lista, respectivamente. A partir de cada resultado dessa partição, uma das classes é eliminada

da lista e um novo nó é definido, separando as classes inicial e final da nova lista formada. Esse processo é iterado até a obtenção das folhas, que equivalem a listas com apenas uma classe.

Figura 3.5: Exemplo de DDAG para um problema com quatro classes (Platt et al., 2000)

Pode-se verificar que, na classificação de um novo exemplo com essa estrutura, $k - 1$ classificadores binários são avaliados. Logo, essa estrutura acelera a fase de predição da abordagem tct.

Uma desvantagem do DDAG, apontada por Kijisirikul & Ussivakul (2002), é sua dependência em relação à sequência de classificadores binários nos nós do grafo. Essa característica afeta a sua confiabilidade, uma vez que diferentes permutações de nós no grafo podem produzir resultados distintos. Considere, por exemplo, o problema com três classes ilustrado na Figura 3.6. Pode-se recorrer a hiperplanos para separar os pares de classes, os quais encontram-se indicados pelas fronteiras traçadas entre as classes. Na região sombreada encontram-se pontos para os quais, dependendo da estrutura de DDAG utilizada, classificações diferentes são produzidas. Esse fato também encontra-se ilustrado na mesma figura, a qual apresenta os três possíveis DDAGs para um problema com três classes. Verifica-se que a classificação de um dado na região sombreada por cada um deles produz saídas distintas. Logo, a estrutura do DDAG tem influência na saída final produzida.

Figura 3.6: Classificação na região sombreada difere de acordo com a estrutura do DDAG

Outra deficiência do DDAG apontada por Kijirikul & Ussivakul (2002) é que, dependendo da posição da classe correta no grafo, o número de avaliações com esta classe é desnecessariamente grande, resultando em um alto erro cumulativo. Se a classe correta de um exemplo encontra-se no nó raiz, por exemplo, ela será testada $k - 1$ vezes contra as outras para a geração da saída desejada. Se há uma probabilidade de 1% de erro em cada nó, isto irá causar uma taxa de erro cumulativo de $1 - (0.99)^{k-1}$, que se torna crítica especialmente para valores altos de k . Logo, se na predição do rótulo de um dado a classe correta deste encontra-se em níveis próximos da raiz, essa classe é exposta a um maior risco de ser rejeitada erroneamente.

Esses fatores motivaram Kijirikul & Ussivakul (2002) a desenvolverem uma nova estratégia hierárquica para combinar as saídas produzidas por classificadores obtidos pela decomposição tct. A nova estrutura, denominada DAG Adaptável (*Adaptive Directed Acyclic Graph* - ADAG), corresponde a um DAG com estrutura reversa. Na Figura 3.7 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 $\lceil \frac{k}{2} \rceil$ nós, seguidos por $\lceil \frac{k}{2^2} \rceil$ na segunda camada e assim por diante, até que uma camada com um único nó é atingida, a qual produz a saída final.

Figura 3.7: Exemplo de ADAG para problema com oito classes (Kijirikul & Ussivakul, 2002)

Para a classificação de um novo dado utilizando o ADAG, inicialmente ele é avaliado por todos os classificadores do primeiro nível, os quais diferenciam pares distintos de todas as classes. No caso do número de classes ser ímpar, um dos nós contém apenas uma classe, a qual é diretamente passada para o segundo nível. Cada um dos preditores binários produz como saída sua classe preferida. Baseado nas escolhas do nível anterior, os classificadores das próximas camadas são determinados. A cada passagem de nível, o número de classes candidatas é reduzido pela metade. Como no DDAG, $k - 1$ nós são avaliados em cada predição. Porém, a classe correta é testada contra outras classes $\lceil \log_2 k \rceil$ ou menos vezes, número menor que no DDAG, que pode requerer até $k - 1$ avaliações com essa classe.

Utilizando essa estrutura, o ADAG minimiza a ocorrência de erros cumulativos, espe-

cialmente 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 ou DDAGs (Phetkaew et al., 2003a,b; Takahashi & Abe, 2003; Feng et al., 2005). O número possível de diferentes DDAGs para um problema com k classes é $\frac{k!}{2}$ e de ADAGs é $\frac{k!}{2^{\lfloor k/2 \rfloor}}$. 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 $f = -f$.

Em (Phetkaew et al., 2003b) é proposta uma modificação no algoritmo original do ADAG, denominada *Reordering ADAG* (RADAG), que permite adequar a estrutura do ADAG ao problema multiclasses sendo solucionado. Esses autores verificaram que classificadores binários que separam as classes com maior margem produzem menos erros de classificação. Eles propuseram então o uso dessa informação na obtenção da estrutura do grafo. Como maximizar as margens das SVMs equivale a minimizar $\|\mathbf{w}\|$, propôs-se que em cada nível do ADAG se utilize pares de classes (i, j) tal que o classificador binário possua $\|\mathbf{w}_{ij}\|$ pequeno. No primeiro nível agrupa-se então os pares de classes tal que a soma dos $\|\mathbf{w}_{ij}\|$ dos classificadores binários seja minimizada. Na classificação de um exemplo, como no ADAG, cada classificador do primeiro nível produz como saída sua classe preferida. O novo nível é então determinado agrupando as classes preditas em pares, tal que novamente a soma dos $\|\mathbf{w}_{ij}\|$ seja mínima. Esse processo é repetido até a obtenção do classificador do último nível, que produz a saída final. Logo, o RADAG mantém a característica de adaptar a estrutura do grafo de acordo com cada dado de teste, porém utiliza-se a informação dos $\|\mathbf{w}_{ij}\|$ neste processo. O agrupamento das classes em pares em cada nível é realizado com o uso de um algoritmo de pareamento perfeito de peso mínimo (*minimum-weight perfect matching*) (Grimaldi, 2000). Os autores reportaram resultados com o uso do procedimento descrito comparáveis ou superiores aos de ADAGs gerados aleatoriamente e do tct com votação por maioria.

Em (Phetkaew et al., 2003a) o mesmo algoritmo é apresentado, porém agrupa-se em cada nível pares de classes com erro mínimo segundo o limite fornecido pela Equação 2.6. O uso dessa heurística levou a resultados similares aos reportados em (Phetkaew et al., 2003b), porém em alguns conjuntos de dados uma das duas abordagens superou um pouco a outra.

Em (Takahashi & Abe, 2003) é sugerido que, na determinação da estrutura de um DDAG, os nós das primeiras camadas contenham os classificadores com maior capacidade de generalização. Como observado na Figura 3.6, existe uma região na qual diferentes DDAGs levam a classificações distintas. Cada DDAG dá preferência a uma determinada classe nesse processo. Takahashi & Abe (2003) apontam que essa preferência é determinada pelos classificadores contidos no último nível do DDAG. Logo, colocando pares de

classes fáceis de serem separados próximos à raiz, a preferência recai sobre classes difíceis de serem distinguidas e, portanto, mais suscetíveis a erros. Com isso, aumenta-se a oportunidade de produzir essas classes como saída, minimizando possíveis erros.

A avaliação da facilidade de separar as classes foi medida pela generalização dos classificadores para cada par de classes, segundo um limite de erro de *leave-one-out* (LOO). Esse tipo de limite permite estimar o erro obtido por uma SVM binária em um procedimento de LOO, descrito no Apêndice A, a partir de informações obtidas em seu treinamento no conjunto de dados total, tal como o seu número de SVs (Vapnik, 1995). Takahashi & Abe (2003) reportam que essa heurística levou à obtenção de estruturas de DDAGs com desempenho semelhante à média de estruturas geradas de maneira aleatória. Takahashi & Abe (2003) também apontam que o algoritmo proposto pode ser facilmente estendido para os ADAGs.

Em (Feng et al., 2005) o mesmo raciocínio anterior foi empregado na determinação de estruturas de DDAGs, porém utilizou-se um limite de erro de LOO alternativo na avaliação dos classificadores, de Jaakkola & Haussler (1999). Esses autores argumentam também que, posicionando os classificadores com maior capacidade de generalização em níveis superiores, minimiza-se a ocorrência de erros cumulativos no DDAG.

Nesta Tese também é introduzida uma proposta para a determinação da estrutura de DDAGs e ADAGs, a qual encontra-se descrita no Capítulo 4.

► Árvores Direcionadas Binárias

Diversos trabalhos empregam uma estrutura em forma de árvore direcionada binária na obtenção do classificador multiclases hierárquico. As árvores possuem $k - 1$ classificadores binários e, portanto, envolvem o treinamento de $k - 1$ preditores, menor número entre todas as estratégias decomposicionais principais mencionadas anteriormente. Além disso, os nós de níveis inferiores envolvem cada vez menos classes e, portanto, menos dados de treinamento para os classificadores binários correspondentes. Na fase de teste, 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 frente às anteriormente discutidas.

Para um problema com $k \geq 3$ classes, existem $\prod_{i=3}^k 2i - 3$ estruturas de árvores distintas (Frank & Kramer, 2004). Duas possíveis árvores para um problema com quatro classes são ilustradas na Figura 3.8. 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, conseqüentemente, 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.

Em (Takahashi & Abe, 2002) é proposto que os nós das camadas iniciais da árvore particionem as classes mais facilmente separáveis. É argumentado que, se os nós das primeiras

Figura 3.8: Duas árvores para um problema com quatro classes (Frank & Kramer, 2004)

camadas produzem muitos erros, o resultado final pode ser deteriorado. Foram propostos então quatro métodos para a obtenção de árvores. No primeiro, separa-se recursivamente uma classe de outras utilizando como medida de separabilidade a distância Euclidiana entre os centros das classes, calculados a partir do conjunto de treinamento. No segundo, em cada nó separa-se subconjuntos de uma ou mais classes. Na obtenção desses subconjuntos, emprega-se um procedimento iterativo de agrupamento das classes segundo os seus centros. No terceiro método, utiliza-se um classificador simples que separa os dados de acordo com a sua distância de Mahalanobis aos centros das classes. A classe que possui menos dados de treinamento classificados incorretamente é separada das demais em cada nó. No quarto método, realiza-se um procedimento de agrupamento de subconjuntos de uma ou mais classes, como no segundo método. Entretanto, utiliza-se nesse processo as informações de dados incorretamente classificados pela distância de Mahalanobis discutida anteriormente. Os quatro métodos foram comparados por Takahashi & Abe (2002) em um experimento abreviado e apresentaram resultados similares, porém superiores à decomposição 1ct.

Em (Schwenker, 2000) e (Schwenker & Palm, 2001) utilizou-se o conceito de classes de confusão na definição das partições binárias de classes na hierarquia. Uma classe de confusão representa um subconjunto de classes para as quais há uma grande semelhança entre os dados, tal que mesmo uma pequena quantidade de ruídos pode levar a classificações errôneas (Schwenker & Palm, 2001). A idéia é realizar uma discriminação mais geral entre classes de confusão distintas e iterativamente refinar o processo de discriminação dentro dessas classes. Na obtenção das classes de confusão, os autores consideraram duas estratégias: a geração de matrizes de confusão e o uso de uma técnica de agrupamento. A principal desvantagem do uso de uma matriz de confusão apontada por esses autores está no fato de que algum outro algoritmo de aprendizado que realiza predições multi-classes deve ser empregado na solução de todo o problema para a sua geração. A avaliação das combinações de partições dos dados a serem utilizadas também pode se tornar com-

putacionalmente cara. A alternativa considerada mais viável foi empregar o algoritmo de agrupamento k-médias (Mitchell, 1997), com $k = 2$. Nesse procedimento, os dados são particionados em dois grupos. Verifica-se então a frequência relativa com que membros de uma classe i aparecem em cada grupo formado. As classes são então particionadas em dois subconjuntos de acordo com essa informação, ou seja, elas são atribuídas ao subconjunto correspondente ao grupo no qual a maioria de seus dados se encontram. Esse processo é repetido recursivamente até a obtenção da árvore. Avaliando o desempenho da árvore obtida por esse método na classificação de dígitos manuscritos, Schwenker & Palm (2001) verificaram que a sua taxa de acerto foi similar à das estratégias 1ct e tct com votação por maioria.

Em (Vural & Dy, 2004) são sugeridos 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, efetivamente aplicado nos experimentos reportados no artigo, as classes são divididas em dois subconjuntos tal que a diferença entre o número de dados em cada um deles é mínima. Esse critério é rápido e se mostrou vantajoso principalmente se as classes encontram-se desbalanceadas, ou seja, apresentam grandes diferenças na quantidade de dados. Outra contribuição desse trabalho foi a obtenção de um limite de generalização similar ao dos DDAGs para as árvores, também indicando que o uso de técnicas que utilizam o princípio de maximização de margens constitui uma boa alternativa para essas estruturas.

Lei & Govindaraju (2005) utilizaram uma técnica de agrupamento hierárquico para definir as partições binárias das classes. Agrupou-se em cada nível da hierarquia subconjuntos de classes mais semelhantes de acordo com a distância média entre os seus dados. Os resultados experimentais obtidos foram similares aos das estratégias 1ct, tct com votação por maioria e DDAG.

Em (Frank & Kramer, 2004), como alternativa a empregar heurísticas na definição de uma única árvore para um problema multiclases, é proposta a combinação de diversas estruturas distintas. Porém, esse trabalho emprega uma abordagem probabilística na integração das saídas dos classificadores binários nas hierarquias. Logo, o processo de reconstrução é probabilístico.

No Capítulo 5, utilizando conceitos introduzidos pelos trabalhos discutidos, é apresentado um algoritmo baseado na teoria dos grafos para a definição de classificadores hierárquicos com estrutura de árvore direcionada binária.

3.2 Estratégias Diretas

Os algoritmos de treinamento das SVMs podem ser reformulados para solucionar o problema multiclases em um único passo, sem a necessidade de criar múltiplos preditores binários. Para isso, deve-se recorrer a um problema de otimização maior, no qual as

funções que discriminam as classes são determinadas em conjunto. A principal crítica a esse tipo de procedimento é que os problemas de otimização gerados são complexos. Com isso, eles se tornam computacionalmente caros e são geralmente difíceis de implementar (Hsu & Lin, 2002; Rifkin & Klautau, 2004). Embora como resultado o tempo de treinamento seja maior, essas técnicas muitas vezes são mais rápidas na fase de teste que as estratégias decomposicionais (Hsu & Lin, 2002).

Entre os principais trabalhos da literatura que reformulam o algoritmo das SVMs pode-se citar: (Weston & Watkins, 1998; Vapnik, 1998; Bredensteiner & Bennet, 1999; Guermer, 2000; Lee et al., 2001; Crammer & Singer, 2002). Os algoritmos de Weston & Watkins (1998) e de Crammer & Singer (2002) são apresentados em mais detalhes nas seções 3.2.1 e 3.2.2, respectivamente. É demonstrado em (Guermer, 2000) e (Hsu & Lin, 2002) que as formulações de Vapnik (1998), Bredensteiner & Bennet (1999) e Guermer (2000) são equivalentes à de Weston & Watkins (1998). A estratégia de Lee et al. (2001) provê uma análise do comportamento assintótico das SVMs que levou à formulação de uma versão multiclasses dessa técnica. Rifkin & Klautau (2004) apontam, contudo, alguns problemas nesse algoritmo. Entre eles, tem-se o fato da análise ser assintótica e, portanto, ser garantida somente quando o número de dados tende ao infinito. Além disso, o problema de otimização gerado é grande e Lee et al. (2001) fornecem um meio de solucioná-lo apenas quando os conjuntos de dados são pequenos.

3.2.1 Extensão de Weston e Watkins

O trabalho de Weston & Watkins (1998), simultaneamente ao de Vapnik (1998), apresentou a primeira reformulação das SVMs para soluções multiclasses. Para k classes, k funções do tipo $f_j(\mathbf{x}) = \mathbf{w}_j \cdot \Phi(\mathbf{x}) + b_j$ são utilizadas na separação das classes, porém todas são determinadas em um único problema de otimização. Tem-se a seguinte formulação:

$$\underset{\mathbf{w}, \mathbf{b}, \boldsymbol{\xi}}{\text{Minimizar}} \quad \frac{1}{2} \sum_{j=1}^k \|\mathbf{w}_j\|^2 + C \sum_{i=1}^n \sum_{j \neq y_i} \xi_{ij} \quad (3.6)$$

$$\text{Com as restrições: } \begin{cases} \mathbf{w}_{y_i} \cdot \Phi(\mathbf{x}_i) + b_{y_i} - (\mathbf{w}_j \cdot \Phi(\mathbf{x}_i) + b_j) \geq 2 - \xi_{ij} \\ \xi_{ij} \geq 0, \quad i = 1, \dots, n, \quad j \in \{1, \dots, k\} \setminus y_i \end{cases} \quad (3.7)$$

Como nas SVMs binárias, o objetivo é maximizar as margens de cada função de classificação que será utilizada na separação das classes. Para tal, na função objetivo da Equação 3.6, minimiza-se as normas $\|\mathbf{w}_j\|$, para $j = 1, \dots, k$. Cada uma das j funções tem associada variáveis de folga ξ_{ij} para os n dados de treinamento, que relaxam as restrições nas margens. Valores de ξ_{ij} maiores que 0 indicam dados $\mathbf{x}_i$ entre as margens ou incorretamente classificados. Deve-se então minimizar também a soma dos ξ_{ij} , reduzindo os erros marginais no conjunto de treinamento, como realizado para as SVMs binárias. Essa soma é multiplicada por uma constante de regularização C , que impõe um peso à minimização dos erros no conjunto de treinamento em relação à minimização da

complexidade do modelo.

Weston & Watkins (1998) requerem que cada dado $\mathbf{x}_i$ seja classificado pela função f_{y_i} com uma confiança de ao menos 2 em relação às saídas de todos os outros classificadores f_j , com $j \neq y_i$. Com o conceito de margens suaves, na realidade essa confiança é relaxada para $2 - \xi_{ij}$, como pode ser visualizado na primeira restrição na Equação 3.7. O valor 2 pode ser substituído, por exemplo, por 1, sem afetar a saída do problema. Rifkin & Klautau (2004) apontam que a escolha desse valor por Weston & Watkins (1998) pode ter sido motivada por uma decisão de tornar o sistema mais semelhante às SVMs binárias. Nestas toma-se os hiperplanos canônicos e requer-se que os exemplos das classes positiva e negativa sejam classificados com uma margem de confiança de 1 ($-\xi_i$ com as margens suaves) em relação à fronteira de decisão. Logo, a separação entre as classes positiva e negativa possui margem de confiança de 2.

No processo de solução do problema apresentado é introduzida uma função Lagrangiana e, assim como no caso das SVMs binárias, chega-se a um problema dual, apresentado a seguir.

$$\text{Maximizar}_{\boldsymbol{\alpha}} \quad 2 \sum_{i=1}^n \sum_{j=1}^k \alpha_{ij} + \sum_{i=1}^n \sum_{j=1}^n \sum_{l=1}^k \left[-\frac{1}{2} c_{jy_i} A_i A_j + \alpha_{il} \alpha_{jy_i} - \frac{1}{2} \alpha_{il} \alpha_{jl} \right] K(\mathbf{x}_i, \mathbf{x}_j) \quad (3.8)$$

$$\text{Com as restrições: } \begin{cases} \sum_{i=1}^n \alpha_{ij} = \sum_{i=1}^n c_{ij} A_i, & j = 1, \dots, k \\ 0 \leq \alpha_{ij} \leq C, & \alpha_{iy_i} = 0, \quad i = 1, \dots, n, \quad j = 1, \dots, k \end{cases} \quad (3.9)$$

Em que:

$$A_i = \sum_{j=1}^k \alpha_{ij} \quad (3.10)$$

$$c_{ij} = \begin{cases} 1 & \text{se } y_i = j \\ 0 & \text{se } y_i \neq j \end{cases} \quad (3.11)$$

Pode-se observar que no problema dual apresentado $\boldsymbol{\alpha}$ tem $k.n$ elementos. Como n deles são sempre nulos pela restrição $\alpha_{iy_i} = 0$, para todo $i = 1, \dots, n$, tem-se $(k-1)n$ variáveis a serem determinadas, número $k-1$ vezes maior que o das SVMs binárias. Tem-se também mais restrições que as SVMs binárias, sendo $k-1$ restrições de igualdade e $n.(k-2)$ restrições de desigualdade a mais. O problema dual apresentado é então claramente mais complexo.

Em (Hsu & Lin, 2002) é apresentado o procedimento para a obtenção da solução desse problema, ou seja, para determinar os valores ótimos α_{ij}^* , com $i = 1, \dots, n$ e $j = 1, \dots, k$. A partir desses valores, a função de decisão é fornecida pela Equação 3.12. Ela é

equivalente à reconstrução comumente empregada na estratégia decomposicional 1ct, em que escolhe-se a função de classificação que produz a maior saída.

$$f(\mathbf{x}) = \arg \max_{1 \leq j \leq k} \left(\sum_{i=1}^n (c_{ij} A_i - \alpha_{ij}^*) K(\mathbf{x}_i, \mathbf{x}_j) + b_j^* \right) = \arg \max_{1 \leq j \leq k} (\mathbf{w}_j^* \cdot \Phi(\mathbf{x}) + b_j^*) \quad (3.12)$$

Os valores dos b_j^* podem ser determinados pelas condições de Kuhn-Tucker para esse problema. Ainda segundo essas condições, tem-se que $\mathbf{x}_i$ é um SV se ao menos um dos α_{ij}^* associados ao mesmo é maior que 0. Como nas SVMs binárias, os SVs são dados de treinamento incorretamente classificados ou aqueles mais próximos à fronteira de decisão multiclasses induzida, se localizando neste caso entre ou sobre as margens do classificador. Esses tipos de SVs encontram-se ilustrados na Figura 3.9 (Passerini, 2004). Os exemplos em cinza correspondem aos SVs sobre as margens e os dados em preto são SVs entre as margens. Exemplos em preto com bordas extras são erros de treinamento. Todos os outros dados, em branco, não são SVs. A fronteira de decisão induzida na separação das classes encontra-se indicada pelas linhas contínuas e as margens são ilustradas por linhas tracejadas.

Figura 3.9: SVs em versão multiclasses de Weston & Watkins (1998) (Passerini, 2004)

3.2.2 Extensão de Crammer e Singer

Em (Crammer & Singer, 2001a, 2002) é descrita uma técnica para a adaptação de códigos de correção de erros à aplicação multiclasses sendo solucionada. Dado um conjunto de classificadores binários $f_1(\mathbf{x}), \dots, f_l(\mathbf{x})$, procura-se uma matriz de códigos $\mathbf{M}$ de dimensão $k \times l$. Os preditores binários podem ser gerados por técnicas decompositivas tradicionais, como a 1ct, tct e ECOC. Realiza-se então uma adaptação da matriz de códigos considerando o desempenho do sistema multiclasses no conjunto de treinamento. Crammer & Singer (2002) argumentam que encontrar uma matriz de códigos discreta a partir dos classificadores e dos dados de treinamento pode ser considerado um problema NP-completo e o relaxam permitindo que $\mathbf{M}$ possua elementos contínuos.

Seja $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_l(\mathbf{x}))$ o vetor de previsões dos classificadores binários para um exemplo $\mathbf{x}$. Crammer & Singer (2001a) definem a confiança com que $\mathbf{x}$ é classificado em uma classe q por $\mathbf{m}_q \cdot \mathbf{f}(\mathbf{x})$, em que $\mathbf{m}_q$ corresponde à q -ésima linha de $\mathbf{M}$. A classe de um

dado é então fornecida pela linha que apresenta a maior medida de confiança, conforme a Equação 3.13, em que $f(\mathbf{x})$ representa o classificador multiclases final.

$$f(\mathbf{x}) = \arg \max_{1 \leq q \leq k} \{\mathbf{m}_q \cdot \mathbf{f}(\mathbf{x})\} \quad (3.13)$$

O erro em um conjunto de treinamento T do classificador multiclases codificado em $\mathbf{M}$ pode ser então dado por:

$$\varepsilon_T(\mathbf{M}) = \frac{1}{n} \sum_{i=1}^n I(f(\mathbf{x}_i) \neq y_i) \quad (3.14)$$

O objetivo é encontrar uma matriz $\mathbf{M}$ com erro pequeno em T e que tenha boa capacidade de generalização. Esse erro pode ser limitado pela Equação 3.15, em que $\delta_{ij} = 1$ se $i = j$ e $\delta_{ij} = 0$ caso contrário.

$$\varepsilon_T(\mathbf{M}) \leq \frac{1}{n} \sum_{i=1}^n \left(\max_{1 \leq q \leq k} \{\mathbf{f}(\mathbf{x}_i) \cdot \mathbf{m}_q + 1 - \delta_{y_i q}\} - \mathbf{f}(\mathbf{x}_i) \cdot \mathbf{m}_{y_i} \right) \quad (3.15)$$

Por essa equação, tem-se que a confiança associada à classe correta deve ser maior que as atribuídas às outras classes em pelo menos uma unidade. Caso contrário, soma-se ao erro um valor proporcional à diferença entre a confiança da classe correta e do máximo das confidências das outras classes. Com essas considerações, chega-se ao seguinte problema de otimização:

$$\underset{\mathbf{M}, \boldsymbol{\xi}}{\text{Minimizar}} \quad \frac{1}{2} \|\mathbf{M}\|^2 + C \sum_{i=1}^n \xi_i \quad (3.16)$$

$$\text{Com as restrições: } \mathbf{f}(\mathbf{x}_i) \cdot \mathbf{m}_{y_i} - \mathbf{f}(\mathbf{x}_i) \cdot \mathbf{m}_q \geq 1 - \delta_{y_i q} - \xi_i, \quad i = 1, \dots, n, \quad q = 1, \dots, k \quad (3.17)$$

Na função objetivo, $\|\mathbf{M}\|^2 = \sum_{i,j} m_{ij}^2$ e C é uma constante de regularização. A minimização da norma de $\mathbf{M}$ tem como objetivo aumentar a capacidade de generalização do classificador final, em analogia com a minimização de $\|\mathbf{w}\|$ nas SVMs binárias. As restrições quanto às confidências das previsões são relaxadas, como nas SVMs binárias de margens suaves, permitindo que, para alguns dados, a confiança quanto à classe correta seja maior que as confidências das outras classes em $1 - \xi_i$. Deve-se minimizar então os ξ_i , reduzindo a ocorrência de penalidades nas restrições originalmente propostas. Para $q = y_i$, pode-se observar que as restrições se tornam $\xi_i \geq 0$.

Como a confiança da classe de cada dado pode ser comparada à maior das confidências, tem-se apenas n elementos em $\boldsymbol{\xi}$, contra $n.k$ na formulação de Weston & Watkins (1998), na qual exige-se que a confiança da classe correta em relação a todas as outras classes seja maior que uma constante.

Em Crammer & Singer (2001b) é apresentada uma versão multiclases para as SVMs

em que toma-se $\mathbf{f}(\mathbf{x}) = \mathbf{x}$, ou seja, tem-se um número de classificadores binários igual à dimensão dos dados e cada $f_i(\mathbf{x})$ corresponde à i -ésima coordenada de $\mathbf{x}$. Com isso, pelo produto escalar de $\mathbf{M}$ e $\mathbf{x}$, obtém-se k hiperplanos do tipo $\mathbf{w}_j \cdot \mathbf{x} = 0$ (Rifkin & Klautau, 2004). Não se tem, portanto, os termos b_j nessa formulação e os hiperplanos determinados passam pela origem. Pode-se ainda aplicar uma função de mapeamento a $\mathbf{x}$, definindo assim uma fronteira não linear no espaço de entradas, como nas SVMs binárias. O problema de otimização obtido pode então ser formulado como (Hsu & Lin, 2002):

$$\underset{\mathbf{w}, \boldsymbol{\xi}}{\text{Minimizar}} \quad \frac{1}{2} \sum_{j=1}^k \|\mathbf{w}_j\|^2 + C \sum_{i=1}^n \xi_i \quad (3.18)$$

$$\text{Com as restrições: } \mathbf{w}_{y_i} \cdot \boldsymbol{\Phi}(\mathbf{x}_i) - \mathbf{w}_j \cdot \boldsymbol{\Phi}(\mathbf{x}_i) \geq 1 - \delta_{y_i j} - \xi_i, \quad i = 1, \dots, n, \quad j = 1, \dots, k \quad (3.19)$$

Aplicando a teoria Lagrangiana a esse problema, chega-se à seguinte forma dual, formulada em (Hsu & Lin, 2002) como uma minimização:

$$\underset{\boldsymbol{\alpha}}{\text{Minimizar}} \quad \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j) \boldsymbol{\alpha}_i \cdot \boldsymbol{\alpha}_j + \sum_{i=1}^n \boldsymbol{\alpha}_i \cdot (1 - \boldsymbol{\delta}_i) \quad (3.20)$$

$$\text{Com as restrições : } \begin{cases} \sum_{j=1}^k \alpha_{ij} = 0, & i = 1, \dots, n \\ \alpha_{ij} \leq 0 \text{ se } y_i \neq j, & \alpha_{ij} \leq C \text{ se } y_i = j, \quad i = 1, \dots, n, \quad j = 1, \dots, k \end{cases} \quad (3.21)$$

Em que $\boldsymbol{\alpha}_i = (\alpha_{i1}, \dots, \alpha_{ik})$ e $\boldsymbol{\delta}_i = (\delta_{i1}, \dots, \delta_{ik})$. Como no problema de otimização dual da seção anterior, a solução $\boldsymbol{\alpha}^*$ tem $n \cdot k$ elementos. Tem-se também mais restrições que nas SVMs binárias, sendo $n - 1$ a mais de igualdade e $n \cdot (k - 1)$ a mais de desigualdade. A partir da solução dual $\boldsymbol{\alpha}^*$, o classificador final é dado por:

$$f(\mathbf{x}) = \arg \max_{1 \leq j \leq k} \left[\sum_{i=1}^n \alpha_{ij}^* K(\mathbf{x}_i, \mathbf{x}) \right] = \arg \max_{1 \leq j \leq k} (\mathbf{w}_j^* \cdot \boldsymbol{\Phi}(\mathbf{x})) \quad (3.22)$$

Nessa abordagem, os SVs, exemplos $\mathbf{x}_i$ com ao menos um $\alpha_{ij}^* > 0$, correspondem aos dados de treinamento cujos rótulos são considerados incertos (Crammer & Singer, 2001b). Os outros exemplos não participam na determinação do classificador multiclasses final.

3.3 Considerações Finais

Embora algumas técnicas de aprendizado, como as SVMs, sejam originalmente formuladas para a solução de problemas de classificação binários, o seu uso pode ser estendido a aplicações multiclasses. 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 multiclass. Nas estratégias diretas, reformula-se o algoritmo original da técnica de aprendizado em uma versão multiclass. Entretanto, esse procedimento nem sempre é simples de ser realizado. Este capítulo revisou as principais estratégias decomposicionais e duas reformulações das SVMs para aplicações multiclass. Outras técnicas também podem ser consultadas em (Lorena & Carvalho, 2003e). É importante ressaltar que o estudo das estratégias decomposicionais, embora em alguns casos tenha sido focado nas SVMs, se aplica a qualquer técnica de aprendizado, bastando para isto utilizá-la na geração de classificadores binários.

Ainda com relação às estratégias decomposicionais, foram apresentadas duas abordagens para a representação e a agregação dos classificadores binários produzidos: baseada em matrizes de códigos e hierárquicas. Porém, há também trabalhos que combinam as saídas dos preditores binários com o 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 multiclass. Entre os trabalhos que fizeram uso desse tipo de estratégia, pode-se mencionar: (Mayoraz & Alpaydim, 1998; Savicky & Fürnkranz, 2003; Lorena & Carvalho, 2004a).

O levantamento bibliográfico realizado evidencia que as pesquisas sobre a geração de SVMs multiclass têm sido intensas, o que também pode ser constatado pela presença de várias publicações recentes na área. Contudo, apenas as técnicas mais empregadas na prática ou relevantes para o trabalho sendo realizado foram apresentadas neste capítulo. Existem diversos outros algoritmos na literatura e cada publicação aponta as vantagens de uma técnica particular sobre outras. De forma geral, porém, pode-se dizer que cada técnica tem vantagens e também deficiências.

A partir dos estudos e observações realizados, os dois próximos capítulos introduzem novas alternativas para a geração de SVMs multiclass a partir de preditores binários, adaptando as decomposições realizadas a cada aplicação multiclass considerada.

Capítulo 4

Estratégias Evolutivas

Embora existam várias estratégias para a geração de SVMs multiclases a partir de preditores binários, não há uma abordagem reconhecidamente superior para todo o tipo de aplicação. Em (Ding & Dubchak, 2001), por exemplo, a estratégia tct com integração por votação por maioria se mostrou mais eficaz que a 1ct no reconhecimento de dobras de proteínas, enquanto em (Yeang et al., 2001) a abordagem 1ct apresentou melhores resultados na classificação de tumores frente à decomposição tct.

No caso das estratégias baseadas em matrizes de códigos tradicionais, o fato da decomposição do problema multiclases realizada por essas técnicas ser fixa pode ocasionar dificuldades em algumas aplicações. Pode-se gerar subproblemas difíceis de serem solucionados ou inadequados para a separação das classes. Quanto aos DDAGs e ADAGs, segundo (Kijssirikul & Ussivakul, 2002; Takahashi & Abe, 2003), deve-se dar atenção à definição da conformação dos preditores binários nesses grafos, uma vez que a estrutura imposta em ambos influencia os seus desempenhos.

Este capítulo aborda esses dois problemas, propondo o uso de procedimentos de busca na solução das questões levantadas. Procura-se, no primeiro caso, a combinação de preditores binários em uma matriz de códigos que seja adequada à solução da aplicação multiclases. Como esses classificadores definem como as classes são particionadas, por esse processo determina-se a decomposição do problema multiclases. Similarmente, no caso dos DDAGs e ADAGs procura-se as estruturas desses grafos a partir de seu desempenho na aplicação multiclases. Empregou-se Algoritmos Genéticos (AGs) (Mitchell, 1999) na solução de ambos os problemas de busca formulados. Os AGs são algoritmos de busca e otimização pertencentes à classe dos Algoritmos Evolutivos, os quais utilizam procedimentos baseados na teoria da evolução natural e na genética.

Iniciando este capítulo, na Seção 4.1 os Algoritmos Evolutivos são descritos. É dado maior enfoque aos tipos de AGs empregados na solução dos problemas identificados an-

teriormente. A seguir, a proposta do uso de AGs na especificação dos classificadores em uma matriz de códigos é introduzida na Seção 4.2. A determinação da estrutura dos DDAGs e ADAGs por meio de AGs é abordada na Seção 4.3. Finalizando, na Seção 4.4 são apresentadas as considerações finais deste capítulo.

4.1 Algoritmos Evolutivos

Os Algoritmos Evolutivos (AEs) são técnicas computacionais com princípios baseados na teoria da evolução natural e na genética (Mitchell, 1999).

Segundo a teoria de evolução natural, introduzida por Charles Darwin (1859), organismos que se adaptam melhor ao seu ambiente possuem maiores chances de terem suas características transmitidas para uma nova geração. Logo, o ambiente exerce uma seleção sobre os organismos que compõem uma determinada população, a qual privilegia os indivíduos mais adaptados.

Em 1900, a genética, baseada nos experimentos de Mendel em 1865, foi integrada ao trabalho de Darwin, introduzindo conceitos que complementaram suas observações sobre a seleção natural. Um dos conceitos mais importantes é o de hereditariedade, que define o modo como as características de um indivíduo são transmitidas aos seus descendentes (Eiben & Smith, 2003).

Além da hereditariedade, outros dois fatores são necessários para a ocorrência da seleção natural: a reprodução e a presença de variações nas características dos indivíduos em uma população, ou seja, a ocorrência de variabilidade genética (Pinto, 2004). Havendo variabilidade genética em uma população, a seleção natural pode agir de forma a privilegiar indivíduos que possuam características que os tornam mais adaptados (ou mais aptos) ao ambiente. Além disso, novas características podem ser introduzidas, as quais podem ocasionalmente melhorar a adaptabilidade do indivíduo. O processo de reprodução representa a introdução de novos indivíduos na população.

Os AEs imbuem os conceitos discutidos por meio da operação de um conjunto de possíveis soluções para o problema sendo resolvido. Esse conjunto, em analogia com os conceitos biológicos introduzidos anteriormente, é usualmente denominado população. Essa população é iterativamente adaptada de forma a produzir soluções cada vez mais aptas a resolver o problema. Realiza-se com isso um procedimento de busca, no qual a solução ótima com aptidão máxima representa o objetivo a ser encontrado ou aproximado. Por esse motivo, os AEs são comumente referenciados como algoritmos de busca e otimização e vêm sendo aplicados a uma grande gama de problemas, que incluem aplicações das áreas de controle, planejamento, otimização combinatorial e de AM (Carvalho et al., 2003; Beasley, 2000).

De forma geral, é possível distinguir quatro tipos principais de AEs (Eiben & Smith, 2003): Estratégias de Evolução, Programação Evolutiva, Programação Genética e Algoritmos Genéticos.

As Estratégias de Evolução foram introduzidas nas décadas de 1960 e 1970 (Rechen-

berg, 1965, 1973). O objetivo era solucionar problemas de hidrodinâmica e de controle (Carvalho et al., 2003). As soluções neste caso são comumente codificadas por valores reais (Whitley, 2002). A Programação Evolutiva foi introduzida na década de 1960 para produzir máquinas de estado finito, que codificam as possíveis soluções (Fogel et al., 1966). Mais recentemente, Koza (1992) propôs a Programação Genética (PG) como uma forma de desenvolver programas computacionais por meio de um procedimento adaptativo. Para tal, manipula-se soluções codificadas por listas ou árvores. Alguns autores classificam a PG como um tipo de Algoritmo Genético (AG) (Kinnear Jr, 2000). Os AGs, popularizados por Goldberg (1989), foram propostos por John Holland (1975), com o objetivo inicial de estudar os mecanismos de adaptação natural ocorridos na natureza e incorporá-los a sistemas computacionais. As soluções neste caso são normalmente codificadas por cadeias de bits.

As próximas seções descrevem os AGs em maiores detalhes. Estes foram os AEs empregados no presente trabalho, conforme será descrito nas seções 4.2 e 4.3. Inicialmente discute-se o funcionamento de um AG básico, seguido de versões para problemas de permutação e problemas com múltiplos objetivos.

4.1.1 Algoritmo Genético Básico

Dada uma população inicial de possíveis soluções para o problema a ser resolvido, as quais são referenciadas como indivíduos, os Algoritmos Genéticos (AGs) buscam atingir a solução global a partir de um processo iterativo. A cada iteração, também denominada geração, uma nova população é produzida, a qual contém evoluções de indivíduos selecionados da geração anterior.

Os indivíduos são codificados por uma estrutura de dados denominada cromossomo. No AG básico ou padrão, os cromossomos são representados por cadeias de bits. Cada bit, também referenciado como gene, representa a presença (valor 1) ou a ausência (valor 0) de alguma característica no indivíduo (Bäck, 2000). Esta é a codificação empregada nas figuras 4.1 e 4.2. Existem, porém, diversos outros tipos de codificação e a representação é normalmente determinada de acordo com as características do problema investigado. Para problemas com parâmetros contínuos, pode-se recorrer ao uso de números reais e em problemas de permutação, definidos na Seção 4.1.2, é comum utilizar-se uma representação de listas dos elementos a serem ordenados (Michalewicz, 1996).

A população inicial é geralmente definida por soluções geradas de maneira aleatória. Entretanto, pode-se fornecer soluções consideradas relevantes previamente, que são então utilizadas pelo AG como pontos iniciais no processo de busca.

Em seguida, deve-se definir uma forma de avaliar os indivíduos, quantificando assim a aptidão de cada um deles em solucionar o problema. Essa avaliação é realizada por meio do uso de uma função de avaliação, a qual decodifica a informação contida no cromossomo do indivíduo e obtém uma medida de sua qualidade. Como a forma de codificação, essa função é dependente do problema investigado.

A partir da população avaliada, aplica-se um mecanismo para a seleção de indivíduos que passarão para a fase de reprodução, produzindo descendentes que formarão uma nova população. Esta seleção deve privilegiar a escolha de indivíduos mais aptos, em conformidade com os princípios da seleção natural.

Existem diversos métodos de seleção, sendo o mais simples o da roleta. Neste método, atribui-se a cada indivíduo uma fatia de uma roleta. O tamanho da fatia deve ser proporcional ao valor da aptidão do indivíduo. A roleta é girada e a posição em que esta pára indica o indivíduo selecionado. Para selecionar q indivíduos, a roleta é girada q vezes. Esse é o método de seleção mais comumente empregado. Porém, na prática, ele aumenta a suscetibilidade do AG apresentar uma convergência prematura, levando a soluções finais sub-ótimas. Este fenômeno ocorre porque essa seleção faz com que indivíduos com alta aptidão em uma população se multipliquem muito rapidamente, impedindo que o AG explore outras regiões do espaço de soluções. Com isso a variabilidade é diminuída e, em fases posteriores da busca, restam poucas diferenças entre as aptidões dos indivíduos para a seleção agir (Mitchell, 1999).

Uma das alternativas à roleta é o método de seleção por torneio. Segundo Mitchell (1999), o torneio pode ser implementado da seguinte maneira: a cada seleção, dois indivíduos da população corrente são escolhidos aleatoriamente. Um número entre 0 e 1 é então gerado. Se esse número é menor que uma constante p_s no intervalo $[0, 1]$ (por exemplo, 0,75), o indivíduo mais apto é selecionado. Caso contrário, o menos apto é escolhido.

Na reprodução dos indivíduos selecionados, busca-se combinar as suas características ou genes na obtenção dos descendentes, representando o conceito de hereditariedade. Essa combinação é realizada pela aplicação de um operador genético denominado cruzamento. O cruzamento é um operador binário, sendo aplicado sobre dois indivíduos. Estes indivíduos são denominados pais e os seus cromossomos são combinados na produção de dois novos indivíduos, os filhos.

Para a representação com cadeias de bits discutida anteriormente, um operador de cruzamento comumente aplicado é o cruzamento de um ponto. Dados dois pais, escolhe-se aleatoriamente um ponto de corte. Os filhos são obtidos trocando-se as partes dos pais posteriores ao ponto de corte, conforme ilustrado na Figura 4.1.

Simulando a natureza estocástica da evolução, o cruzamento é usualmente aplicado segundo uma taxa p_c (Zitzler et al., 2004). A partir de dois candidatos a pais determinados pelo processo de seleção, implementa-se este procedimento gerando um número aleatório entre 0 e 1. Se esse número é menor que p_c , realiza-se o cruzamento. Normalmente, adota-se $0,6 \leq p_c \leq 0,99$ (Carvalho et al., 2003). Caso sejam produzidos, os filhos farão parte da nova população. Caso contrário, os pais são introduzidos nessa população.

Por meio da combinação produzida pelo cruzamento, obtém-se também uma variabilidade nas novas soluções. O conceito de variabilidade genética é reforçado pela aplicação de um operador unário denominado mutação. A mutação altera genes dos indivíduos da nova população gerada na etapa de cruzamento. Dado um indivíduo, pode-se implementar

Figura 4.1: Operador de cruzamento de um ponto

a mutação alterando o valor de um de seus genes segundo uma taxa p_m . Dessa forma, para cada indivíduo, gera-se novamente um número aleatório entre 0 e 1. Caso esse número seja menor que p_m , efetua-se a mutação. Geralmente emprega-se taxas de mutação pequenas ($0,001 \leq p_m \leq 0,1$) (Carvalho et al., 2003), para prevenir que o algoritmo altere muito a população, o que poderia prejudicar a sua convergência. Na Figura 4.2 é ilustrado um exemplo de mutação em um indivíduo.

Figura 4.2: Operador de mutação

Embora não esteja presente no algoritmo original de Holland (1975), outro tipo de seleção comumente aplicado sobre uma população é o elitismo. No elitismo, uma proporção p_e dos melhores indivíduos da população corrente é passada diretamente para a nova população que será formada. Com isso, previne-se que boas soluções sejam perdidas no decorrer do processamento do AG.

Os procedimentos de geração de uma população, avaliação de seus indivíduos, seleção e aplicação dos operadores genéticos são iterados, formando a base dos AGs. No Algoritmo 4.1 é ilustrado um pseudo-código de um AG. Cada iteração dos passos 3 a 10 desse algoritmo constitui uma geração. O conjunto total de gerações, juntamente com os passos 1 e 2, é referenciado como uma execução do AG. Dependendo da população inicial provida, a execução do AG pode levar a soluções distintas para o problema analisado. Dessa forma, é usual executar-se o algoritmo várias vezes com populações iniciais diferentes, obtendo então uma média de seu desempenho. Deve-se observar também que o tamanho da população é geralmente mantido constante no decorrer das gerações.

Para a parada do AG, diferentes critérios podem ser utilizados. Pode-se parar o AG quando um número máximo de gerações for atingido, quando a aptidão média da

Algoritmo 4.1 Algoritmo Genético (Carvalho et al., 2003)

```

1:  $t = 0$ ;
2: Gerar população inicial de indivíduos  $P(0)$ ;
3: repita
4:   Avaliar a aptidão dos indivíduos em  $P(t)$ ;
5:    $t = t + 1$ ;
6:   Aplicar o elitismo a  $P(t - 1)$  e inserir os indivíduos selecionados em  $P(t)$ ;
7:   Selecionar indivíduos de  $P(t - 1)$  para reprodução;
8:   Aplicar cruzamento aos indivíduos do passo 7 e inserir indivíduos obtidos em  $P(t)$ ;
9:   Aplicar operador de mutação a  $P(t)$  (excluindo os indivíduos advindos de elitismo);
10: até que (critério de parada seja atingido)

```

---

população ou do melhor indivíduo não se modificar por um número estipulado de gerações ou quando as aptidões dos indivíduos da população se tornarem muito semelhantes (Carvalho et al., 2003). Se o valor de aptidão máximo é conhecido, também pode-se parar o AG quando uma solução com essa aptidão é encontrada.

Com a aplicação dos procedimentos apresentados no Algoritmo 4.1, indivíduos cada vez mais aptos a solucionar o problema são produzidos, enquanto os menos aptos tendem a se extinguir (Souto et al., 2003). Logo, em gerações iniciais, os AGs exploram o espaço de soluções em múltiplas direções e, em etapas posteriores, regiões mais promissoras são enfatizadas. Com o uso da abordagem populacional em conjunto com os operadores genéticos, a chance de se encontrar a solução ótima no espaço de busca, embora não seja garantida, é maior que a de vários métodos de busca e otimização tradicionais. Muitos desses algoritmos, como por exemplo o de subida de colina (*hill-climbing*), exploram apenas uma vizinhança ao redor de uma única solução inicial (Michalewicz & Fogel, 2004). Além disso, diversos métodos tradicionais de otimização também necessitam de informações de derivadas e gradientes para guiar a busca pela solução e possuem a tendência de não conseguir evitar ótimos locais (Michalewicz & Fogel, 2004).

Os AGs possuem ainda a vantagem de serem capazes de realizar buscas em espaços de soluções contendo várias partes complexas, em que o impacto de cada parte na solução do problema pode ser difícil de modelar por meio de técnicas tradicionais (Mitchell, 1997). Eles são também facilmente paralelizáveis, podendo tirar proveito do crescente uso de tecnologias distribuídas.

Entretanto, a exploração de populações de possíveis soluções tornam o AG geralmente mais oneroso computacionalmente. Sua formulação também pode ser considerada muito geral. Com isso, embora os AGs sejam robustos, Whitley (1994) sugere que, caso exista um método especializado para um dado problema, os AGs podem não ser a melhor alternativa para solucioná-lo.

Deve-se observar também que os AGs têm um conjunto de parâmetros a serem definidos, como o tamanho da população e as taxas de elitismo p_e , de cruzamento p_c e de mutação p_m . A definição desses parâmetros influencia fortemente o seu desempenho (Carvalho

et al., 2003).

4.1.2 Algoritmos Genéticos em Problemas de Permutação

Um problema de permutação pode ser definido como encontrar a ordem de m elementos $\{e_1, \dots, e_m\}$ que seja considerada ótima segundo algum critério. Um exemplo clássico de problema desse tipo é o do caixeiro viajante, no qual, dadas m cidades a serem visitadas, deseja-se encontrar a ordem de visita que implique em percorrer um caminho de custo mínimo. Cada cidade deve ser visitada uma única vez, com exceção da cidade inicial, para a qual o caixeiro deve retornar ao final de sua viagem. Tem-se $m-1!$ possíveis soluções, que correspondem às diferentes ordens ou permutações das cidades, considerando equivalentes as ordens que impliquem em uma mesma relação de vizinhança entre as cidades. Esse problema é NP-completo (Michalewicz, 1996).

Para representar as soluções para problemas de permutação, é usual fazer-se uso de listas contendo uma ordem dos elementos. Esta representação codifica a permutação de forma mais natural que o uso de um vetor de bits. Deve-se observar que uma solução para um problema de permutação é válida se cada elemento e_i está presente na lista somente uma vez. Além disso, todos os elementos devem estar contidos na lista. Segundo essa codificação, um exemplo de permutação é dado pelo vetor $(e_1, e_2, \dots, e_m)$.

Para gerar indivíduos que representem permutações válidas, o operador de cruzamento deve ser adaptado, prevenindo que sejam produzidas listas com elementos duplicados ou omitidos. Deve-se ainda combinar as características dos pais de forma adequada, tal que os descendentes obtidos apresentem alguma similaridade com os pais. Para o problema do caixeiro viajante, três dos tipos de operadores de cruzamento mais comuns são: o parcialmente mapeado, usualmente denotado por PMX (Goldberg & Lingle, 1985), o de ciclo, representado por CX (Oliver et al., 1987), e o de ordem, referenciado por OX (Davis, 1985). Esses operadores são descritos a seguir. Optou-se por apresentar os operadores da literatura do caixeiro viajante por este ser o problema de permutação mais difundido e cuja forma de solução com o uso de AGs pode ser facilmente estendida a outras aplicações de busca por permutações.

Na aplicação do cruzamento PMX para a obtenção de um filho, escolhe-se uma subsequência de elementos de um pai e mantém-se a ordem e a posição do maior número de classes possível do outro pai (Michalewicz, 1996). A subsequência é obtida pela seleção de dois pontos de corte aleatórios. Por simplicidade, considera-se um exemplo no qual cada e_i é igual a i e $m = 9$, ou seja, os elementos a serem permutados são os inteiros $1, 2, \dots, 9$. Sejam os pais a serem combinados representados a seguir, com pontos de corte indicados por “|” (Michalewicz, 1996):

$$p_1 = (1 \ 2 \ 3 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 8 \ 9)$$

$$p_2 = (4 \ 5 \ 2 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 9 \ 3)$$

Inicialmente, os segmentos entre os pontos de corte são trocados, produzindo os des-

cendentes:

$$\begin{aligned} f_1 &= (x \ x \ x \ | \ 1 \ 8 \ 7 \ 6 \ | \ x \ x) \\ f_2 &= (x \ x \ x \ | \ 4 \ 5 \ 6 \ 7 \ | \ x \ x) \end{aligned}$$

Essas trocas definem uma série de mapeamentos: $1 \leftrightarrow 4$, $8 \leftrightarrow 5$, $7 \leftrightarrow 6$.

Preenche-se os outros elementos para os quais não há repetições:

$$\begin{aligned} f_1 &= (x \ 2 \ 3 \ | \ 1 \ 8 \ 7 \ 6 \ | \ x \ 9) \\ f_2 &= (x \ x \ 2 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 9 \ 3) \end{aligned}$$

O primeiro x em f_1 , que deveria ser 1, já presente na lista, é substituído pelo elemento 4 por meio do mapeamento $1 \leftrightarrow 4$. Da mesma maneira, os outros elementos indicados por x são preenchidos. Como resultado final, tem-se:

$$\begin{aligned} f_1 &= (4 \ 2 \ 3 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 5 \ 9) \\ f_2 &= (1 \ 8 \ 2 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 9 \ 3) \end{aligned}$$

No cruzamento CX, constrói-se os filhos tal que cada elemento e a sua posição venha de um dos pais. Por meio desse procedimento, o CX é capaz de preservar a posição absoluta dos elementos das seqüências pais (Michalewicz, 1996). Sejam p_1 e p_2 os pais:

$$\begin{aligned} p_1 &= (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \\ p_2 &= (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5) \end{aligned}$$

Insere-se inicialmente o primeiro elemento do pai 1 no filho 1, obtendo:

$$f_1 = (1 \ x \ x \ x \ x \ x \ x \ x)$$

O segundo elemento a ser inserido em f_1 deve vir do pai 2 na mesma posição em que o primeiro foi colocado. Deve-se então selecionar o elemento 4, o elemento de p_2 na posição 1. Em p_1 o elemento 4 está na posição 4, então insere-se da seguinte forma:

$$f_1 = (1 \ x \ x \ 4 \ x \ x \ x \ x)$$

Seguindo o mesmo raciocínio, essa operação implica na escolha do elemento 8, o elemento de p_2 na posição 4. Logo, tem-se:

$$f_1 = (1 \ x \ x \ 4 \ x \ x \ 8 \ x)$$

A seguir, vêm os elementos 3 e 2. Porém, o elemento 2 requer a seleção do elemento 1, o qual já está presente em f_1 . Completou-se então um ciclo:

$$f_1 = (1 \ 2 \ 3 \ 4 \ x \ x \ 8 \ x)$$

Os outros elementos são preenchidos de acordo com o conteúdo de p_2 , excluindo os presentes na lista:

$$f_1 = (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5)$$

Aplicando o mesmo procedimento com o papel dos pais invertido, tem-se:

$$f_2 = (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9)$$

O operador OX obtém os filhos impondo uma subsequência de um pai e preenchendo as posições remanescentes a partir da ordem relativa dos elementos do outro pai (Michalewicz, 1996; Davis, 1985). Sejam p_1 e p_2 os mesmos pais apresentados no exemplo do cruzamento PMX, com os mesmos pontos de corte indicados por “|” (Michalewicz, 1996):

$$p_1 = (1\ 2\ 3\ |4\ 5\ 6\ 7|\ 8\ 9)$$

$$p_2 = (4\ 5\ 2\ |1\ 8\ 7\ 6|\ 9\ 3)$$

Primeiramente copia-se os segmentos entre os pontos de corte nos filhos:

$$f_1 = (x\ x\ x\ |4\ 5\ 6\ 7|\ x\ x)$$

$$f_2 = (x\ x\ x\ |1\ 8\ 7\ 6|\ x\ x)$$

Em seguida, a partir do segundo ponto de corte, os elementos do outro pai são copiados na mesma ordem, omitindo-se elementos já presentes. Chegando ao final da cadeia, continua-se da primeira posição. Seguindo esse procedimento, tem-se para p_2 a seguinte ordem: 9-3-4-5-2-1-8-7-6. Omitindo os elementos 4, 5, 6 e 7 já presentes em f_1 , obtém-se 9-3-2-1-8. Preenche-se então f_1 com essa sequência a partir do segundo ponto de corte. Tem-se como resultado:

$$f_1 = (2\ 1\ 8\ |4\ 5\ 6\ 7|\ 9\ 3)$$

Aplicando os mesmos passos com o papel dos pais invertido, tem-se:

$$f_2 = (3\ 4\ 5\ |1\ 8\ 7\ 6|\ 9\ 2)$$

Em uma variação do operador OX, Syswerda (1991) propôs como alternativa a escolher uma subsequência de elementos entre dois pontos de corte a serem diretamente copiados, selecionar vários elementos em posições aleatórias (Michalewicz, 1996). O restante do processo de obtenção dos filhos é análogo. Esse operador é denominado cruzamento baseado em posições e será denotado por PX.

Como o cruzamento, o operador de mutação também deve ser adaptado de forma a produzir indivíduos que correspondam a permutações válidas. Um operador de mutação comumente empregado na literatura do caixeiro viajante é a inserção. A sua aplicação sobre um indivíduo consiste na seleção de um elemento e a sua inserção em uma posição aleatória do indivíduo (Michalewicz, 1996). Seja, por exemplo, o indivíduo:

$$i_1 = (1 \ 4 \ 3 \ 5 \ 6 \ 7 \ 9 \ 2 \ 8)$$

Deseja-se inserir a classe 7 na terceira posição da lista. O resultado é o indivíduo:

$$i_1' = (1 \ 4 \ 7 \ 3 \ 5 \ 6 \ 9 \ 2 \ 8)$$

Dada a representação e os operadores genéticos de cruzamento e mutação discutidos, o AG pode ser diretamente aplicado segundo o Algoritmo 4.1. Ele irá então buscar a permutação ótima segundo o problema de interesse, por meio de iterações em que uma população de possíveis permutações são continuamente evoluídas.

4.1.3 Algoritmos Genéticos para Problemas Multi-objetivo

No AG apresentado na Seção 4.1.1, a aptidão de cada indivíduo é calculada segundo o seu desempenho em cumprir um determinado objetivo. Este objetivo pode ser genericamente caracterizado por uma função h , a qual deve ser maximizada ou minimizada. Essa função é comumente denominada função objetivo. Por simplicidade, as discussões posteriores se referirão a problemas de minimização, podendo ser diretamente estendidas ao caso de maximização sem perda de generalidade. Sendo $\mathbf{z}$ uma possível solução, o seu desempenho é medido pelo valor de sua avaliação em h , ou seja, por $h(\mathbf{z})$. Deve-se encontrar então uma solução $\mathbf{z}^*$ que leve a um valor $h(\mathbf{z}^*)$ mínimo. A busca por essa solução pode ainda estar sujeita a restrições.

Existem diversas aplicações reais, porém, que possuem mais de um objetivo a ser minimizado ou maximizado, os quais muitas vezes são conflitantes e podem também estar sujeitos a restrições. Em (Cancino et al., 2003), por exemplo, deseja-se encontrar alinhamentos de proteínas com extensão e qualidade maximizadas, com a restrição dos alinhamentos produzidos serem válidos.

Dados q objetivos, seja h_i a i -ésima função objetivo a ser minimizada. Assume-se também que cada solução é codificada por um vetor $\mathbf{z}$, o qual está relacionado a um vetor de objetivos $\mathbf{v}$ de dimensão q que contém a avaliação de $\mathbf{z}$ nas q funções objetivo h_i . As soluções $\mathbf{z}$ encontram-se em um espaço Z , denominado espaço de decisão, enquanto os vetores de objetivos $\mathbf{v}$ encontram-se em um espaço de objetivos O .

No caso de um único objetivo a ser minimizado, ou seja, com $q = 1$, dadas duas soluções $\mathbf{z}_1$ e $\mathbf{z}_2$, pode-se considerar que $\mathbf{z}_1$ é melhor ou mais apta que $\mathbf{z}_2$ se $v_1 < v_2$, em que $v_i = h(\mathbf{z}_i)$. Podem existir diversas soluções que levem a um valor de h mínimo. Porém, o valor mínimo de h é único, ou seja, há apenas um ótimo global no espaço de objetivos (Zitzler et al., 2004).

Na presença de vários objetivos sendo minimizados, a comparação de duas soluções e a definição de ponto ótimo é mais complexa (Coello, 1999). É comum utilizar-se o conceito de dominância de Pareto nesse processo (Pareto, 1896). Sejam $\mathbf{v}_1$ e $\mathbf{v}_2$ os vetores de objetivos de $\mathbf{z}_1$ e $\mathbf{z}_2$, respectivamente. Diz-se que $\mathbf{v}_1$ domina $\mathbf{v}_2$ (denotado por $\mathbf{v}_1 \succ \mathbf{v}_2$) se nenhum componente de $\mathbf{v}_1$ é maior que o elemento correspondente em $\mathbf{v}_2$ e ao menos um

componente é menor (Zitzler et al., 2004). Logo, diz-se que a solução $\mathbf{z}_1$ é melhor que $\mathbf{z}_2$, isto é, que $\mathbf{z}_1$ domina $\mathbf{z}_2$ ($\mathbf{z}_1 \succ \mathbf{z}_2$), se $\mathbf{v}_1 \succ \mathbf{v}_2$. Na Figura 4.3 é ilustrado um exemplo em que deseja-se minimizar três objetivos e a solução $\mathbf{z}_1$ domina a solução $\mathbf{z}_2$, pois os valores de suas avaliações nos objetivos, representados em $\mathbf{v}_1$, são iguais aos de $\mathbf{v}_2$ em dois casos e menor em um caso.

Figura 4.3: Exemplo de dominância de Pareto em problema de minimização com três objetivos

A partir desses conceitos, soluções que não são dominadas por qualquer outra são consideradas ótimas. Como conseqüência, podem existir diferentes soluções ótimas no espaço de objetivos, as quais representam compromissos distintos entre os objetivos sendo minimizados. O grupo de soluções ótimas em Z é denominado conjunto de Pareto e a sua imagem em O é referenciada por frente de Pareto (Zitzler et al., 2004). Um AG pode ser empregado para encontrar diversas soluções que aproximem o conjunto de Pareto ou para obter uma solução particular se há conhecimento do domínio para guiar a busca por essa solução.

Existem diversos AEs na literatura para lidar com problemas multi-objetivo (Deb, 2001). Eles podem ser divididos, segundo Coello (1999), em três categorias: abordagens que utilizam funções de agregação, abordagens baseadas na relação de Pareto e um terceiro grupo geral que reúne os algoritmos que não podem ser classificados nos tipos anteriores. Como no AG básico, as soluções manipuladas pelos AGs multi-objetivo são comumente representadas no espaço de decisão e pode-se então utilizar codificações usuais na definição dos cromossomos. Porém, algumas modificações são introduzidas no Algoritmo 4.1.

Nas abordagens que utilizam funções de agregação, a avaliação da aptidão dos indivíduos é calculada por uma função que combina os objetivos $h_i(\mathbf{z})$ de alguma forma. Um exemplo de algoritmo desse tipo é o de soma ponderada, no qual a função combinada $H(\mathbf{z})$ é dada pela Equação 4.1, em que $\sum_{i=1}^q w_i = 1$. Logo, cada objetivo recebe um peso w_i e minimiza-se a soma dos objetivos ponderados. Para que cada w_i reflita corretamente a importância dos objetivos, estes devem estar contidos em intervalos numéricos semelhantes (Coello, 1999). O AG básico do Algoritmo 4.1 pode então ser aplicado diretamente, com a avaliação dos indivíduos realizada segundo $H(\mathbf{z})$. Logo, o problema multi-objetivo é reduzido a um problema com um único objetivo (Cvetkovic & Parmee, 1998). Dado um

vetor de pesos particular, uma única solução é encontrada.

$$H(\mathbf{z}) = \sum_{i=1}^q w_i h_i(\mathbf{z}) \quad (4.1)$$

Embora esse método possa ser considerado simples e eficiente do ponto de vista computacional, sua maior deficiência está na determinação dos pesos de cada objetivo. Normalmente, deve-se ter algum conhecimento do domínio para estipular esses pesos de forma adequada ou realizar-se testes empíricos variando os seus valores.

Uma maneira alternativa de lidar com os múltiplos objetivos, classificada por Coello (1999) no terceiro grupo discutido anteriormente, é impor uma ordem lexicográfica nos mesmos. Neste caso, os objetivos são primeiramente ordenados pelo desenvolvedor e a minimização é realizada de acordo com a ordem estabelecida. Novamente um conhecimento do domínio torna-se necessário para a definição de uma ordem apropriada. Para incorporar a ordem lexicográfica ao AG, pode-se realizar adaptações nos processos de elitismo e de seleção do Algoritmo 4.1. Ambas as etapas realizam seleções de indivíduos com base em suas aptidões. Cada escolha de indivíduo é então adaptada de maneira a se considerar inicialmente o objetivo mais importante. No caso de haver empate entre diferentes soluções, utiliza-se o segundo objetivo em ordem de importância e assim por diante.

A desvantagem da versão lexicográfica encontra-se justamente no processo de favorecimento de objetivos, pois a importância dada a cada objetivo é crucial em seu desempenho (Cvetkovic & Parmee, 1998). Porém, ela é simples e muitas vezes competitiva com outras estratégias (Coello, 1999). Como no algoritmo de ponderação, a aplicação do procedimento lexicográfico descrito leva diretamente a uma única solução (Cvetkovic & Parmee, 1998).

Nas abordagens baseadas na relação de Pareto, os AGs são empregados na aproximação do conjunto de Pareto. Logo, diversas soluções são encontradas e o usuário pode fazer uso de todas elas ou escolher alguma particular de acordo com o seu interesse. Um algoritmo recente que utiliza os princípios de Pareto é o SPEA2 (*Strength Pareto Evolutionary Algorithm 2*) (Zitzler et al., 2002).

O SPEA2 utiliza duas populações: uma regular P e um arquivo A . O arquivo implementa o conceito de elitismo e constitui a saída final do algoritmo, apresentando um conjunto de soluções não dominadas segundo a relação de Pareto. Por este ser o AG multi-objetivo discutido que apresenta maiores modificações em relação ao AG básico, um pseudo-código do SPEA2 é apresentado no Algoritmo 4.2.

Como no AG básico, cada ciclo dos passos 3 a 19 é denominado uma geração e esses passos em conjunto com as etapas 1 e 2 constituem uma iteração do algoritmo. As soluções finais também dependem da população inicial. Assim, é recomendado realizar-se múltiplas execuções do algoritmo para verificar o seu desempenho médio.

Na etapa de avaliação (passo 3), inicialmente atribui-se um valor de força $F_o(i)$ a

Algoritmo 4.2 Algoritmo SPEA2 (Zitzler et al., 2002)

```

1:  $t = 0$ ;
2: Gerar uma população inicial aleatória  $P(t)$  e um arquivo vazio  $A(t)$ ;
3: Avaliar a aptidão dos indivíduos em  $P(t)$  e  $A(t)$ ;
4:  $t = t + 1$ ;
5: Elistimo: inserir indivíduos não dominados de  $P(t - 1)$  e  $A(t - 1)$  em  $A(t)$ ;
6: se (tamanho de  $A(t) > n_a$ ) então
7:   Reduzir  $A(t)$  com a aplicação de operador de truncamento;
8: fim-se
9: se (tamanho de  $A(t) < n_a$ ) então
10:   Preencher  $A(t)$  com soluções dominadas de  $P(t - 1)$  e  $A(t - 1)$ ;
11: fim-se
12: se (critério de parada for atingido) então
13:   Manter em  $A(t)$  apenas soluções não dominadas;
14:   Parar o algoritmo;
15: fim-se
16: Selecionar indivíduos de  $A(t)$  para reprodução por torneio;
17: Aplicar cruzamento aos indivíduos do passo 16 e inserir indivíduos obtidos em  $P(t)$ ;
18: Aplicar o operador de mutação a  $P(t)$ ;
19: Voltar ao passo 3;

```

---

cada indivíduo i em A e P , que representa o número de soluções que ele domina. Para simplificar a notação, cada solução ou indivíduo $\mathbf{z}_i$ será denotado por i . O procedimento mencionado é ilustrado na Equação 4.2, em que $|\cdot|$ simboliza o operador de cardinalidade.

$$F_o(i) = |j : j \in A + P \text{ e } i \succ j| \quad (4.2)$$

Com base nos valores de força, uma aptidão bruta $B(i)$ é estimada. Esta considera as forças dos dominantes de i (indivíduos que dominam i) em A e P . Essa relação está representada na Equação 4.3. Um valor de $B(i)$ nulo corresponde a um indivíduo não dominado, enquanto um $B(i)$ alto indica um indivíduo dominado por vários outros, que por sua vez também podem ser dominados por outros.

$$B(i) = \sum_{j \in A+P, j \succ i} F_o(j) \quad (4.3)$$

Utiliza-se ainda uma informação de densidade na avaliação dos indivíduos, a qual permite discriminar soluções com o mesmo valor de $B(i)$. O uso da densidade também contribui para que o algoritmo explore regiões menos visitadas no espaço de objetivos, garantindo uma maior variabilidade das soluções.

A densidade de um ponto é definida como uma função decrescente da distância deste ao seu j -ésimo vizinho mais próximo. Essa distância é tomada no espaço de objetivos. Logo, para cada indivíduo, armazena-se a distância de seu vetor de objetivos aos de todos os outros elementos em A e P . Seja d_i^j a j -ésima menor distância. Zitzler et al. (2002) recomendam o uso de $j = 1$. A densidade de i é então definida pela Equação 4.4. O valor

2 é adicionado no denominador para garantir que $den(i)$ seja maior que 0 e menor que 1.

$$den(i) = \frac{1}{d_i^1 + 2} \quad (4.4)$$

A função de avaliação final de um indivíduo i , a qual deve ser minimizada, é então dada pela função $h_a(i)$, representada na Equação 4.5.

$$h_a(i) = B(i) + den(i) \quad (4.5)$$

Na etapa de elitismo (passo 5 do Algoritmo 4.2), copia-se inicialmente todas as soluções não dominadas de $A(t-1)$ e $P(t-1)$ em $A(t)$, ou seja, todas aquelas com $h_a(i) < 1$. Se o tamanho do novo arquivo $A(t)$ permanece igual a n_a , estipulado pelo usuário, a seleção é finalizada. Se o tamanho de $A(t)$ torna-se menor que n_a , as melhores soluções dominadas em $A(t-1)$ e $P(t-1)$ são copiadas para $A(t)$, completando-a. Caso $A(t)$ tenha tamanho maior que n_a , aplica-se um operador de truncamento. Este remove iterativamente indivíduos de $A(t)$ até que esse conjunto possua n_a elementos. O indivíduo com menor distância a outro é escolhido em cada estágio. Se vários possuem o mesmo valor de distância mínima, considera-se o de segunda menor distância e assim por diante.

Na etapa 12 do Algoritmo 4.2, pode-se utilizar diversos critérios de parada. O mais comum é quando um número máximo de gerações é atingido.

Os operadores de cruzamento e de mutação podem ser os usuais do AG básico ou definidos pelo usuário de acordo com sua aplicação e representação dos indivíduos. Como no AG básico, eles são aplicados segundo taxas p_c e p_m . A seleção de indivíduos para reprodução é usualmente realizada pelo método do torneio.

Ao final do algoritmo SPEA2, tem-se um conjunto de soluções não dominadas. Para selecionar uma delas, pode-se utilizar algum conhecimento do domínio sendo otimizado. Um exemplo é escolher a solução cujo vetor de objetivos apresenta a menor distância Euclidiana em relação a um ponto de referência, o qual contém os valores ótimos de todos os objetivos em suas coordenadas (Deb, 1999).

O SPEA2 possui um conjunto de parâmetros que devem ser estipulados pelo usuário. Entre eles, tem-se: tamanho da população regular, tamanho do arquivo (n_a), taxa de cruzamento (p_c) e taxa de mutação (p_m). Como no AG básico, a escolha dos parâmetros determina o desempenho do algoritmo.

Uma das principais vantagens dos algoritmos que utilizam o conceito de dominância de Pareto, como o SPEA2, encontra-se no fato de todos os objetivos poderem ser levados em consideração simultaneamente (Cvetkovic & Parmee, 1998). Entretanto, se é necessário dar preferência a um objetivo sobre outro, adaptações devem ser realizadas (Deb, 1999). Esses algoritmos podem ser considerados também computacionalmente mais caros, uma vez que o número de possíveis soluções não dominadas pode ser grande. Além disso, eles podem ter seus resultados deteriorados e são pouco intuitivos quando o número de

objetivos é elevado (Cvetkovic & Parmee, 1998).

4.2 AGs na Obtenção de Matrizes de Códigos

Conforme discutido na Seção 3.1.1, as estratégias multiclasse decomposicionais baseadas em matrizes de códigos tradicionais 1ct, tct e ECOC realizam a decomposição do problema multiclasse de forma fixa, a priori. Com isso, elas não levam em consideração as propriedades e as características próprias do domínio em que são aplicadas.

Também foi apontado que, para um problema com k classes, existem $0,5(3^k + 1) - 2^k$ diferentes preditores binários que podem ser gerados. Cada uma das estratégias mencionadas utiliza uma parte desses preditores na definição de sua decomposição do problema multiclasse. A abordagem 1ct utiliza unicamente os classificadores que separam uma classe das demais. A decomposição tct, por sua vez, emprega somente classificadores que distinguem pares de classes. Na estratégia ECOC de Dietterich & Bariki (1995) sugere-se o uso de todos ou uma parte dos $2^{k-1} - 1$ classificadores binários que envolvem todas as classes simultaneamente. Entretanto, podem existir combinações alternativas dos possíveis preditores binários que produzam classificadores multiclasse finais com melhor desempenho. É também possível que existam decomposições mais simples, que façam uso de menos preditores binários, com um bom desempenho na tarefa multiclasse.

A partir dessas observações, desenvolveu-se uma proposta de uso de AGs na determinação de matrizes de códigos para aplicações multiclasse, baseado no desempenho obtido por essas matrizes na solução dos problemas considerados. O AG é responsável por determinar a combinação de classificadores binários em uma matriz de códigos $\mathbf{M}^*$. Com isso, o conteúdo das linhas de $\mathbf{M}^*$, que correspondem aos códigos atribuídos a cada classe, é automaticamente definido. Também embutiu-se no AG a determinação do número de classificadores binários a serem empregados na solução multiclasse, isto é, do número de colunas contidas em $\mathbf{M}^*$. Os processos de busca por uma combinação adequada de classificadores binários em conjunto com o número desses preditores constitui um problema combinatorial. A medida que o número de classes se torna maior o espaço de busca cresce consideravelmente, uma vez que o número de possíveis classificadores binários a serem combinados é da ordem de 3^k .

A avaliação das matrizes geradas pelo AG é realizada segundo o seu desempenho na solução do problema multiclasse. Dessa maneira as matrizes são adaptadas de acordo com cada aplicação. Para atender a esse requisito, o AG busca matrizes que minimizem o erro obtido na solução multiclasse. Procurou-se também minimizar o número de colunas contidas nas matrizes de códigos, controlando assim o número de classificadores binários integrados. Este critério representa a busca por soluções mais simples, estando de acordo com o princípio de Occam, que afirma que, entre diversas hipóteses corretas, a mais simples deve ser selecionada (Mitchell, 1997). Deve-se também evitar a presença de colunas idênticas nas matrizes, o que representaria o uso de preditores binários iguais em uma mesma decomposição. Duas colunas que possuam os elementos com valores

complementares também são consideradas iguais, uma vez que os classificadores binários que elas representam são os mesmos, apresentando apenas rótulos invertidos para as classes. As colunas com elementos exatamente iguais ou complementares serão denotadas por equivalentes nas considerações posteriores.

São então basicamente três os objetivos com que se deve lidar: minimizar o erro da matriz e o seu número de colunas e evitar a presença de colunas equivalentes na matriz. Deve-se assim buscar uma matriz sem colunas equivalentes com bom desempenho na tarefa multiclases e com um número reduzido de classificadores binários. Aplicou-se duas variantes de AG multi-objetivo na realização da tarefa descrita: uma lexicográfica e outra baseada no algoritmo SPEA2. Embora existam outros AGs multi-objetivo baseados na relação de Pareto, a escolha do SPEA2 foi motivada por bons resultados de uma versão anterior desse algoritmo reportados por Zitzler (2002) em análises comparativas.

Prosseguindo a apresentação da proposta introduzida nesta seção, inicialmente são descritos pontos comuns às duas variantes multi-objetivo testadas, como a forma de codificação dos cromossomos (Seção 4.2.1), aspectos comuns na avaliação dos indivíduos (Seção 4.2.2) e os operadores genéticos definidos para este problema (Seção 4.2.3). A seguir, na Seção 4.2.4 discute-se como cada variante foi implementada.

4.2.1 Representação dos Indivíduos

Como o problema considerado envolve determinar matrizes de códigos para problemas multiclases, adotou-se uma estrutura de matriz na representação dos indivíduos. Cada indivíduo corresponde então a uma possível matriz de códigos $\mathbf{M}$ de tamanho $k \times l$ com elementos no conjunto $\{-1, 0, +1\}$, como definido na Seção 3.1.1.

Empregou-se a representação matricial por esta ser mais intuitiva à aplicação em questão. Em (Wallet et al., 1996) é argumentado que, se o problema possui uma estrutura bidimensional inerente, o AG pode obter melhores resultados com o uso de indivíduos codificados de forma matricial. Esses autores empregaram matrizes em AGs com sucesso na remoção de ruídos de imagens. Na codificação utilizada, cada elemento das matrizes podia assumir valores inteiros de 0 a 255, representando os possíveis níveis de cinza contidos em uma imagem. Wallet et al. (1996) apontam que o uso dessa representação também permite definir mais claramente operadores de cruzamento e de mutação adequados ao problema bidimensional sendo resolvido.

Na determinação do número de classificadores binários a compor a matriz final procurada pelo algoritmo, a alternativa empregada foi de permitir que os indivíduos de uma mesma população possuíssem números de colunas l variados. Com isso, este valor também é determinado pelo AG. De acordo com essa estratégia, dois possíveis indivíduos para um problema com quatro classes são ilustrados na Figura 4.4.

Nos algoritmos codificados, o usuário limita o tamanho máximo das matrizes que o AG gera em seu processamento, fornecendo um número máximo de classificadores binários a compô-las. As matrizes geradas pelo AG também devem possuir ao menos $\lceil \log_2 k \rceil$

Figura 4.4: Ilustração de dois possíveis indivíduos para um problema com quatro classes

classificadores binários, que corresponde ao número mínimo necessário para particionar um conjunto com k classes (Mayoraz & Moreira, 1996). O algoritmo então manipula matrizes com números de classificadores entre os valores mínimo e máximo.

A população inicial foi implementada com a definição de indivíduos de tamanhos variados gerados de forma aleatória. Um teste de consistência é aplicado a esses indivíduos, para assegurar que cada classificador binário contido nos mesmos tenha rótulos positivos e negativos definidos. Os códigos das estratégias decomposicionais 1ct, tct e ECOC também podem ser fornecidos à população inicial, provendo então uma informação adicional à busca. Dessa maneira, estimula-se que os classificadores utilizados por essas abordagens possam ser combinados pelo AG na obtenção de melhores soluções.

4.2.2 Procedimento de Avaliação

Nesta seção são descritos aspectos comuns na avaliação dos indivíduos para as duas variantes multi-objetivo empregadas. A forma de cálculo das aptidões dos indivíduos a partir dos pontos discutidos é apresentada na Seção 4.2.4.

Para realizar a avaliação de cada indivíduo, levou-se em consideração o poder preditivo do conjunto de classificadores binários, representados em sua matriz de códigos, na tarefa multiclases. Na decodificação das saídas dos classificadores binários em cada previsão, o usuário pode optar pelo uso das funções de decodificação de *Hamming* (Equação 3.2) ou baseada em margens (Equação 3.5). Calcula-se então, para cada indivíduo i , a medida de erro representada na Equação 4.6. Os AGs implementados devem buscar soluções que minimizem essa medida.

$$e(i) = \text{erro\_val} \quad (4.6)$$

Nessa equação, erro_val refere-se ao erro obtido pelo classificador multiclases representado na matriz de códigos em um conjunto de validação. Classificações desconhecidas, que ocorrem caso mais de uma linha da matriz de códigos apresente distância mínima em relação à cadeia de previsões, encontram-se imbutidas nesse erro. As classificações desconhecidas também incluem o caso em que duas ou mais linhas da matriz são iguais, ou seja, os códigos de classes diferentes são idênticos. Nessa situação, há empate entre as

linhas iguais, ocasionando a classificação desconhecida e, conseqüentemente, um erro. A forma de definição do conjunto de validação para um conjunto de dados é discutida na Seção 6.2.

Na operação do AG, uma matriz de códigos pode acabar apresentando colunas equivalentes. Para inibir esse tipo de ocorrência, cada variante de AG multi-objetivo utiliza uma estratégia distinta. Enquanto a versão lexicográfica penaliza a aptidão de indivíduos que apresentem essa característica, o algoritmo SPEA2 considera a proporção de colunas equivalentes nos indivíduos como um terceiro objetivo a ser minimizado. Esses pontos serão discutidos novamente na Seção 4.2.4.

Como segundo objetivo em ambos os AGs codificados tem-se a minimização do número de classificadores binários nas matrizes. A versão lexicográfica privilegia a minimização do erro na Equação 4.6, colocando a redução do número de classificadores em segunda ordem de importância. No SPEA2, este valor foi considerado como um segundo objetivo a ser minimizado utilizando as relações de dominância de Pareto. Novamente, as descrições desses procedimentos são detalhadas na Seção 4.2.4.

4.2.3 Operadores Genéticos

Os operadores genéticos de cruzamento e mutação foram adaptados a partir da representação de indivíduos considerada e também das características do problema de busca de matriz de códigos formulado. Para a realização de cruzamentos, dois operadores foram definidos neste trabalho:

- Trocas de colunas entre indivíduos. Esta operação corresponde a uma troca de classificadores binários, motivado pelo fato de que um preditor binário pode ser mais eficiente em uma combinação multiclases alternativa. Esse operador é ilustrado na Figura 4.5a.
- Trocas de grupos de colunas entre indivíduos. Neste caso, dados dois indivíduos, seus descendentes são produzidos permutando-se todas colunas dos pais a partir de posições selecionadas aleatoriamente. Esse operador é ilustrado na Figura 4.5b. Pode-se verificar que a aplicação deste procedimento possibilita a produção de indivíduos com novos tamanhos, permitindo a exploração de códigos de comprimento variado. Caso algum dos filhos gerados possua um número de colunas fora dos limites mínimo e máximo estabelecidos, ele é descartado e o seu pai correspondente é passado à nova geração.

Com relação aos operadores de mutação, foram definidos quatro tipos:

- Mudar o valor de um elemento da matriz, selecionado de forma aleatória. Este corresponde ao operador de mutação usual e é ilustrado na Figura 4.6a.
- Pode-se também escolher novos valores para todos os elementos de uma coluna, como demonstrado na Figura 4.6b.

Figura 4.5: Operadores de cruzamento para matrizes de códigos

- Dado um indivíduo, gerar uma nova coluna (classificador binário) com elementos de valores aleatórios. Na Figura 4.6c ilustra-se este caso. Esse operador pode ser aplicado a um indivíduo somente se este possuir um número de colunas inferior ao máximo delimitado.
- Dado um indivíduo, retirar uma de suas colunas, conforme ilustrado na Figura 4.6d. Esse operador pode ser aplicado somente a indivíduos que possuam um número de colunas maior que o valor mínimo delimitado.

Figura 4.6: Operadores de mutação para matrizes de códigos

Deve-se observar que a aplicação dos operadores de mutação pode levar à geração de colunas em que não estejam definidas classes negativas e/ou positivas. Uma fase de checagem de consistência deve então corrigir esses casos, definindo novos rótulos positivos e/ou negativos.

Como há mais de um tipo de operadores de cruzamento e de mutação, uma questão adicional que deve ser considerada é a da escolha de qual deve ser empregado. Para optar por um deles a cada cruzamento e mutação, empregou-se um critério utilizado em (Martí et al., 2005). Cada possível operador é selecionado probabilisticamente de acordo com o seu desempenho em gerações prévias. Este método é efetivado com a definição de um vetor com pontuações para cada operador. Se um indivíduo gerado por cruzamento possui aptidão melhor que a de seus pais, a pontuação do operador de cruzamento que o gerou é incrementada. No caso de uma mutação, se a aptidão do novo indivíduo é melhor que a daquele que o originou, a pontuação do operador de mutação empregado é incrementada. A probabilidade de uso de um operador é então determinada de acordo com a sua pontuação registrada. Para realizar esse procedimento, após os cruzamentos e as mutações, deve ocorrer uma fase de avaliação dos indivíduos, quando as pontuações dos operadores são então atualizadas.

Por meio do uso do esquema descrito, os operadores que produzem melhores soluções em gerações anteriores têm maiores chances de serem aplicados e com isso, a determinação da importância de cada um deles é adaptada pelo AG. Além disso, esse procedimento tem a vantagem de possibilitar o uso de vários tipos de operadores em conjunto.

4.2.4 Algoritmos

Além dos pontos comuns indicados anteriormente, os AGs multi-objetivo formulados para a busca de matrizes de códigos compartilham uma estratégia para acelerar a avaliação dos indivíduos. Esse artifício foi adotado porque o treinamento e teste (no conjunto de validação) dos diferentes classificadores contidos nas matrizes de códigos pode tornar a etapa de avaliação dos indivíduos lenta. Foram criadas duas listas. A primeira armazena matrizes de códigos geradas com as suas respectivas avaliações da medida de erro da Equação 4.6. A segunda mantém classificadores binários treinados com suas informações de desempenho no conjunto de validação.

Quando a avaliação de um indivíduo é realizada, ele é inicialmente procurado na lista de matrizes. Se ele é encontrado, copia-se as informações de desempenho armazenadas. Caso contrário, antes de treinar cada um de seus classificadores binários, eles são inicialmente procurados na lista de preditores. Se o classificador é encontrando, não é necessário treiná-lo novamente. Esse procedimento foi adotado porque um mesmo preditor pode ser gerado muitas vezes no decorrer do processamento do AG. Ambas as listas implementadas são duplamente encadeadas e as suas indexações são feitas de forma a mantê-las ordenadas. Assim, as buscas podem ser realizadas de forma rápida. Para prevenir problemas com armazenamento em memória, o tamanho das listas deve ser limitado pelo usuário. Procura-se manter nas listas apenas os elementos mais acessados e com as melhores aptidões. Isso porque, no decorrer das iterações do AG, tende-se cada vez mais a gerar os mesmos indivíduos, os quais por sua vez são cada vez mais aptos.

As duas versões multi-objetivo possuem, entretando, algumas etapas e características

particulares. Seguem descrições dos aspectos pertinentes a cada variante.

► **Versão Lexicográfica**

Na implementação da versão lexicográfica, ordenou-se inicialmente os dois objetivos minimizados no problema: a medida de erro da Equação 4.6 e o número de classificadores binários contidos nas matrizes. Colocou-se em primeira ordem de importância a minimização do erro. Com isso, privilegia-se a adaptação das soluções à resolução correta do problema multiclases. Caso mais de uma matriz apresente a mesma medida de erro, o segundo objetivo age como um critério de desempate, estando então em segundo lugar na ordem estabelecida. Dessa forma, como mencionado anteriormente, segue-se o princípio de Occam. Este preceito, amplamente difundido na comunidade de AM, afirma que, entre várias hipóteses com o mesmo desempenho, deve-se dar preferência à mais simples (Mitchell, 1997).

Para efetivar essa ordenação dos objetivos, primeiramente calcula-se a aptidão dos indivíduos com a Equação 4.6. Tratando-se de um problema de minimização, os indivíduos mais aptos são aqueles que possuem menor valor de erro segundo essa equação. A seguir, modificou-se as etapas de elitismo e de seleção do Algoritmo 4.1. Com isso, emprega-se um AG básico com algumas modificações para levar em conta a ordem dos múltiplos objetivos considerados.

No elitismo (passo 6 do Algoritmo 4.1), para selecionar os melhores indivíduos em uma população, considera-se inicialmente a informação de aptidão dos mesmos. Se ocorrem empates entre diferentes indivíduos, escolhe-se aquele com o menor número de classificadores. Mantém-se assim a ordem lexicográfica dos objetivos estabelecida: inicialmente considera-se a minimização da estimativa de erro da aptidão e em seguida, do número de classificadores binários na solução.

Para a seleção de cada indivíduo a sofrer cruzamento por torneio (passo 7 do Algoritmo 4.1) também considera-se inicialmente as aptidões dos indivíduos. Caso ocorram empates, utiliza-se o critério de minimização do número de classificadores.

Para matrizes que possuam colunas equivalentes, o cálculo da aptidão foi modificado. Considerou-se evitar a presença de classificadores repetidos nas soluções como uma restrição do problema. Deve-se então privilegiar soluções que não violem a restrição estabelecida. Para tal, a aptidão dos indivíduos passa a ser calculada pela Equação 4.7 (Deb, 2000).

$$\tilde{e}(i) = \begin{cases} e(i), & \text{se } i \in F_c \\ \max_{j \in F_c} (e(j)) + p_{ce}(i), & \text{se } i \in \bar{F}_c \end{cases} \quad (4.7)$$

Nessa equação, F_c denota o conjunto de soluções factíveis, ou seja, aquelas que não violam a restrição, não possuindo classificadores binários iguais. $\bar{F}_c$, por sua vez, representa soluções não-factíveis, $p_{ce}(i)$ representa a proporção de colunas equivalentes no indivíduo i e $e(i)$ é fornecido pela Equação 4.6. Logo, soluções não-factíveis devem possuir aptidão

inferior às das factíveis e são comparadas apenas em termos da intensidade com que violam a restrição. Além disso, a seleção por torneio que envolva indivíduos não-factíveis é adaptada. Se os dois indivíduos no torneio são não-factíveis, seleciona-se aquele que viola menos a restrição. Se um deles é factível, ele é automaticamente selecionado.

Com o uso dos procedimentos descritos as populações do AG podem conter soluções não-factíveis. Embora viole as restrições, uma solução não-factível pode conter informações interessantes para serem exploradas na busca.

No passo 10 do Algoritmo 4.1, o AG é parado se um número máximo de gerações previamente estabelecido é atingido.

Para concluir as considerações sobre o AG lexicográfico, seus parâmetros são enumerados. São eles: tamanho da população, taxa de elitismo p_e , taxa de cruzamento p_c , taxa de seleção por torneio p_s , taxa de mutação p_m e número máximo de gerações.

► SPEA2

Na aplicação do SPEA2, os objetivos minimizados são, como no AG lexicográfico, a medida de erro da Equação 4.6 e o número de classificadores binários da matriz. Como se deseja que as matrizes geradas não possuam colunas equivalentes, optou-se por minimizar ainda a proporção p_{ce} de classificadores iguais nos indivíduos, por meio da consideração de um terceiro objetivo. Com isso, a penalidade a soluções não-factíveis é incorporada ao AG multi-objetivo, conforme descrito em (Michalewicz, 2000).

Logo, tem-se neste caso três objetivos conjuntos. Definidos os objetivos, aplica-se o Algoritmo 4.2 descrito na Seção 4.1.3. Para a implementação desse algoritmo, algumas funções foram adaptadas a partir do código disponível em <http://www.tk.ee.ethz.ch/pisa/selectors/spea2/spea2.html> (Bleuler et al., 2003). A avaliação dos indivíduos é realizada considerando os objetivos determinados e o conceito de dominância de Pareto. Obtém-se, a cada execução deste AG, um conjunto de soluções não dominadas. Como critério para a escolha de uma delas, considerou-se a distância a um ponto de referência. Este ponto apresenta uma taxa nula de erro, um número de classificadores igual ao mínimo necessário para diferenciar as classes e um número nulo de colunas equivalentes. Deve-se optar pela solução cujas avaliações nos objetivos possuam menor distância em relação a esse ponto.

Nas seleções por torneio (passo 16 do Algoritmo 4.2) do SPEA2 original descrito em Zitzler et al. (2002), escolhe-se dois indivíduos aleatoriamente e aquele com melhor aptidão é selecionado como pai. Modificou-se este torneio da forma reportada em (Mitchell, 1999) e descrita na Seção 4.1.1, para prover uma maior uniformidade entre os algoritmos implementados.

Para a parada do SPEA2 no passo 12 do Algoritmo 4.2, utilizou-se o critério de um número máximo de gerações.

Como parâmetros do SPEA2 formulado, tem-se: tamanho da população regular, tamanho do arquivo, taxa de cruzamento p_c , taxa de seleção por torneio p_s , taxa de

mutação p_m e número máximo de gerações.

4.3 AGs na Determinação da Estrutura de DAGs

Conforme apontado na Seção 3.1.2, alguns trabalhos apontam que os resultados de um DDAG dependem de sua estrutura, ou seja, de como os classificadores binários para pares de classes são dispostos nos nós do grafo. Embora este problema seja minimizado nos ADAGs, ainda há diferenças de desempenho entre configurações distintas de nós nesse tipo de grafo. Nesta seção é descrito o uso de AGs na busca de permutações de nós em um DAG (DDAG ou ADAG), baseado em seu desempenho na solução do problema multiclases. Com isso, também neste caso a obtenção do preditor multiclases final é orientada segundo cada aplicação.

Essa definição da ordem dos classificadores binários em um DAG pode ser reduzida a um problema de permutação, no qual se ordena uma lista de classes. Busca-se a ordem de classes que resulte em um grafo com bom desempenho na aplicação multiclases. Por esse motivo, a codificação dos indivíduos e a definição dos operadores genéticos empregados no AG foram adaptadas a partir das estratégias para a solução do problema do caixeiro viajante apresentadas na Seção 4.1.2.

Seguindo a mesma estrutura da seção anterior, a seguir apresenta-se a forma de representação dos indivíduos (Seção 4.3.1), como a avaliação de cada um deles é realizada (Seção 4.3.2), os operadores genéticos empregados (Seção 4.3.3) e o AG formulado a partir das partes descritas (Seção 4.3.4).

4.3.1 Representação dos Indivíduos

Cada possível estrutura de DDAG para um problema com k classes pode ser representada por um vetor contendo uma permutação dessas classes (Platt et al., 2000; Kijirikul & Ussivakul, 2002). Na Figura 4.7 são ilustrados os três diferentes DDAGs para um problema com três classes e as respectivas listas que os representam. Pode-se observar que a ordem apresentada em cada lista define a última camada do grafo correspondente.

Figura 4.7: Representação de DDAGs para três classes por meio de listas

A computação da classe de um novo exemplo com um DDAG particular equivale a operar a lista de classes que o representa. Iniciando pelo nó raiz, o dado é testado contra o

primeiro e o último elementos da lista. Se o valor predito é $+1$, a primeira classe é mantida na lista, enquanto a segunda classe é eliminada. Se a saída é -1 , ocorre o oposto. O nó equivalente ao primeiro e ao último elementos da nova lista obtida é então consultado. A cada passagem de nível no DDAG, a lista é reduzida de um elemento. Esse processo é repetido até que uma única classe permaneça na lista, a qual corresponde ao valor final predito. Para k classes, $k - 1$ preditores são avaliados. Na Figura 4.8 esses procedimentos são apresentados para o DDAG central da Figura 4.7.

Figura 4.8: Previsão da classe de um exemplo com um DDAG para três classes

Os ADAGs também podem ser representados por listas de classes (Kijisirikul & Ussivakul, 2002). Na Figura 4.9 são ilustrados os possíveis ADAGs para três classes e as respectivas listas que os representam. Porém, a operação da lista de classes na obtenção de predições pelo ADAG possui algumas diferenças em relação ao realizado para o DDAG.

Figura 4.9: Representação de ADAGs para três classes por meio de listas

Um exemplo de teste é inicialmente avaliado pelo classificador binário que distingue o primeiro e o último elementos da lista. A classe preferida deste nó é mantida na posição do primeiro elemento. O ADAG então testa o dado em um preditor que envolve a segunda e a penúltima classes da lista. Esse processo é repetido até que uma ou nenhuma classe permaneça sem ser avaliada na lista. Uma nova rodada é então iniciada, com a lista reduzida a $\lfloor k/2 \rfloor$ elementos. Um total de $k - 1$ rodadas são realizadas, quando uma única classe permanece na lista, sendo a saída final fornecida. Caso uma das listas operadas possua um número ímpar de elementos, a classe que se encontra na posição central da

mesma é diretamente passada para a próxima etapa, assumindo a última posição da nova lista formada. Na Figura 4.10 esses procedimentos são ilustrados para o ADAG central da Figura 4.9.

Figura 4.10: Previsão da classe de um exemplo com um ADAG para três classes

Como as abordagens DDAG e ADAG podem ser representadas por listas de classes, o problema de determinação da estrutura desses DAGs pode ser reduzido a buscar uma permutação das classes. Na aplicação dos AGs na obtenção das estruturas de ambos DAGs, optou-se então por uma representação de indivíduos por listas. Cada indivíduo corresponde a um vetor de inteiros, representando as classes. Todas as classes devem estar presentes na lista e não é permitida a ocorrência de repetições de classes. O objetivo é encontrar a permutação de classes nessa lista que leve a menores taxas de erro na operação multiclases do grafo. No caso dos ADAGs, a ordenação dessa lista leva à definição dos pares de classes separados em sua primeira camada. A representação adotada para os indivíduos é similar à representação de caminhos comumente empregada na solução do problema do caixeiro viajante por AGs, descrita na Seção 4.1.2.

Deve-se observar que, na aplicação presente, um classificador para o par de classes (i, j) pode ser considerado equivalente ao preditor para o par (j, i) , uma vez que para as SVMs uma inversão dos rótulos positivo e negativo em um conjunto de treinamento produz essencialmente o mesmo classificador.

O tamanho do espaço de busca para os DDAGs, ou seja, o número de diferentes soluções do problema de determinação da estrutura de um DDAG, é dado por $\frac{k!}{2}$. Este valor corresponde ao número de possíveis DDAGs para um problema com k classes. De forma equivalente, o tamanho do espaço de busca dos ADAGs é $\frac{k!}{2^{\lfloor k/2 \rfloor}}$. Deve-se observar que o uso de AGs na determinação da estrutura de ambos DAGs não é eficiente para aplicações com um número pequeno de classes. Nestes casos, o espaço de busca é reduzido, tornando possível a enumeração exaustiva de todas as estruturas.

4.3.2 Procedimento de Avaliação

A avaliação dos indivíduos foi realizada pela medida de aptidão dos DAGs que estes representam na solução do problema multiclases. Como no AG da Seção 4.2.4, a aptidão de cada indivíduo pode ser calculada pela Equação 4.6, a qual emprega uma estimativa de erro multiclases em um conjunto de validação. O AG deve então buscar soluções que

minimizem este erro e, com isso, os indivíduos mais aptos são aqueles com menor valor da medida calculada na Equação 4.6.

4.3.3 Operadores Genéticos

Dada a similaridade entre a presente aplicação e a do caixeiro viajante, utilizou-se os operadores genéticos da literatura relacionada à solução deste último problema com AGs.

Para a realização dos cruzamentos, foram implementados três dos operadores descritos na Seção 4.1.2. O primeiro é o PMX. Como na implementação do ADAG uma classe na posição i da lista para com a classe na posição $k - i$, tem-se a opção de gerar apenas um ponto de corte. O segundo ponto é então dado pelo seu par segundo a regra apresentada anteriormente, tal que pares de classes (os nós da primeira camada do grafo) dos pais possam ser preservados ainda mais. Entretanto, nos experimentos descritos no Capítulo 6, também escolheu-se pontos de corte aleatórios para verificar os benefícios da adaptação anterior na determinação desses pontos ao processo de busca dos ADAGs. O segundo cruzamento implementado foi o CX. Como terceiro tipo de cruzamento, empregou-se a versão PX do operador OX. Também neste caso, para escolher os índices de posições aleatórias para os ADAGs, dois procedimentos distintos podem ser adotados. O primeiro consiste em selecionar pares de classes em posições i e $k - i$ a serem preservadas, como no operador PMX. O segundo é o processo aleatório padrão. O número de posições a serem mantidas com o uso deste operador também é determinado aleatoriamente.

Como existem três tipos de operadores de cruzamento, o uso deles foi determinado utilizando a seleção de operadores probabilística descrita na Seção 4.2.3. A cada cruzamento, escolhe-se um dos tipos de acordo com o seu desempenho na obtenção de bons indivíduos em gerações prévias. Dessa forma, a importância de cada operador é automaticamente determinada durante o processamento do AG.

Empregou-se como operador de mutação a inserção, também da literatura do caixeiro viajante. Este operador possibilita grandes alterações na configuração dos nós do grafo. Para cada indivíduo sofrendo mutação, optou-se por aplicar o operador de inserção n_m vezes, em que n_m é determinado pelo usuário do algoritmo. Escolhe-se então o indivíduo com melhor aptidão entre os gerados. Com isso, incorporou-se um tipo de procedimento de busca local simplificado ao AG, que tem como objetivo explorar a vizinhança de cada indivíduo que sofre mutação.

4.3.4 Algoritmo

O AG empregado nesta aplicação é o apresentado no Algoritmo 4.1, com indivíduos codificados por meio de listas de classes. Este AG prevê o uso dos operadores de elitismo, cruzamento e mutação. O elitismo segue o padrão convencional, selecionando uma fração dos indivíduos com as melhores aptidões diretamente para a próxima geração. O cruzamento e a mutação são realizados com os operadores descritos anteriormente. A seleção de indivíduos para os cruzamentos é efetuada pelo método de torneio descrito na Seção

4.1.1. A parada do AG no passo 10 do Algoritmo 4.1 é determinada quando um número máximo de gerações é atingido.

Como nos AGs da Seção 4.2.4, foi implementada uma lista com os indivíduos mais gerados, cujas avaliações são armazenadas. Dessa forma, acelera-se a etapa de avaliação das soluções obtidas durante as iterações do AG. Se o indivíduo é localizado na lista, basta copiar a sua informação de aptidão, não sendo necessário testá-lo novamente para o cálculo de seu erro no problema multiclases.

Os parâmetros do AG empregado são: tamanho da população, taxa de elitismo p_e , taxa de cruzamento p_c , taxa de seleção por torneio p_s , taxa de mutação p_m , número máximo de gerações e n_m , o número de vezes em que a inserção é aplicada a um indivíduo sofrendo mutação.

4.4 Considerações Finais

Neste capítulo foram introduzidas propostas envolvendo o uso de AGs na solução de duas questões identificadas no Capítulo 3 relativas à obtenção de classificadores multiclases a partir de preditores binários.

Inicialmente, abordou-se a definição de matrizes de códigos a partir de seu desempenho na solução do problema multiclases. Dessa maneira, a decomposição realizada é orientada de acordo com cada aplicação. Formulou-se um problema de busca com dois objetivos principais. Procura-se a combinação de classificadores binários na matriz com baixo erro na tarefa multiclases e com um número reduzido de classificadores. Deve-se ainda evitar a presença de classificadores binários iguais na matriz de códigos.

Dois tipos de AGs multi-objetivo foram apresentados para a solução desse problema. O primeiro impõe uma ordem lexicográfica nos objetivos de minimização de erro e do número de classificadores binários, penalizando a presença de colunas equivalentes nas matrizes. Duas colunas são consideradas equivalentes se elas levam à geração de preditores binários iguais. Isso pode ocorrer caso todos os elementos das colunas sejam idênticos ou se eles são complementares. O segundo AG considera a minimização do erro multiclases, do número de preditores binários e também de colunas equivalentes de forma conjunta, com o uso do conceito de dominância de Pareto.

Alguns resultados da aplicação de uma versão prévia do AG lexicográfico a um problema de classificação estrutural de proteínas foram apresentados em (Lorena & Carvalho, 2004c). Porém, nesse artigo, os indivíduos foram avaliados empregando um procedimento de *r-fold cross-validation*¹ na obtenção de estimativas de taxa de acerto. O AG buscava então maximizar essa taxa e minimizar o número de classificadores binários empregados nas matrizes de códigos. Reformulou-se esse algoritmo considerando a minimização da taxa de erro para manter ambos os objetivos considerados no processo de otimização como minimizações. Além disso, a versão anterior não apresentava a penalização a matrizes

---

¹O procedimento de *cross-validation* encontra-se descrito no Apêndice A.

com colunas equivalentes, não limitava o tamanho das matrizes geradas pelo AG em um intervalo e não possuía a seleção de tipos de operadores genéticos probabilística.

O segundo problema abordado foi a definição da estrutura dos DDAGs e ADAGs, descritos na Seção 3.1.2. Os AGs foram empregados para encontrar a ordem dos classificadores binários contidos nesses grafos. Considerou-se também a informação de desempenho na solução da aplicação multiclasse na orientação dessa busca. Os grafos obtidos são então adequados a cada aplicação particular.

Em (Lorena & Carvalho, 2004d), esse AG foi testado sobre três conjuntos de dados multiclasse. Como em (Lorena & Carvalho, 2004c), buscava-se maximizar estimativas da taxa de acerto em um procedimento de *r-fold cross-validation*. A reformulação para uma minimização da taxa de erro em um conjunto de validação foi efetuada para promover uma maior uniformidade aos algoritmos codificados no trabalho. Essa versão anterior do algoritmo apresentava apenas o operador de cruzamento do tipo PMX.

A seguir, no próximo capítulo são propostos algoritmos para solucionar o terceiro problema identificado no Capítulo 3 relacionado às estratégias decomposicionais. Este problema se refere a como definir a estrutura de classificadores hierárquicos que possuem uma organização de árvore direcionada binária.

Capítulo 5

Árvores Multiclasses

Uma das formas de organizar vários preditores binários na solução de um problema multiclasses é dispondo-os em uma árvore direcionada binária, conforme descrito na Seção 3.1.2. Essa estrutura possui $k - 1$ classificadores binários, os quais particionam as k classes do problema recursivamente de forma binária.

A definição dessas partições constitui uma etapa importante, uma vez que a estrutura da árvore influencia o seu desempenho na solução do problema multiclasses. Neste capítulo são propostos algoritmos para auxiliar esse processo a partir de informações extraídas dos dados das classes. Os algoritmos descritos se baseiam no uso de grafos e permitem que a estrutura da árvore seja definida automaticamente de acordo com a aplicação multiclasses a ser solucionada.

Na Seção 5.1 são introduzidos conceitos da teoria de grafos. Na Seção 5.2 é apresentado um exemplo de aplicação prática de um algoritmo introduzido por essa teoria, formando uma base para a compreensão dos algoritmos para a definição de classificadores multiclasses com estrutura de árvore, os quais são descritos na Seção 5.3. Na Seção 5.4 são realizadas algumas considerações finais.

5.1 Teoria de Grafos

O uso de grafos foi introduzido por Euler em 1736, que os utilizou para solucionar um problema de caminhar por pontes conectadas a duas ilhas da cidade de Königsberg (Grimaldi, 2000). Ele foi posteriormente disseminado e estendido a diversas outras aplicações, como o desenvolvimento de circuitos elétricos e a modelagem de redes de computadores.

Nesta seção são apresentados os principais conceitos referentes aos grafos. Inicialmente, na Seção 5.1.1, são apresentadas algumas definições básicas da teoria de grafos que são

utilizadas no decorrer desta Tese. Em seguida, na Seção 5.1.2, é apresentado um problema abordado nessa teoria, cuja solução pode ser estendida a várias aplicações práticas.

5.1.1 Definições

Um grafo G é constituído por dois conjuntos. O primeiro é o conjunto de vértices ou nós V , o qual é finito e não-vazio. O segundo é o conjunto de arestas A , que define relações entre pares de vértices de V ($A \subseteq V \times V$). Outra notação para G é $G = (V, A)$ (Grimaldi, 2000).

Caso A contenha pares ordenados de vértices, as arestas assumem uma orientação e G é denominado grafo direcionado ou dígrafo. Logo, nos dígrafos uma aresta $a = (v_i, v_j)$ entre dois vértices v_i e v_j possui a direção de v_i para v_j ($v_i \rightarrow v_j$). Em contrapartida, se os pares de vértices em A não são ordenados, G é denominado grafo não direcionado e cada aresta $a = (v_i, v_j)$ representa as direções $v_i \rightarrow v_j$ e também $v_j \rightarrow v_i$. Na Figura 5.1a é apresentado um grafo não direcionado com cinco vértices, enquanto na Figura 5.1b é ilustrado um grafo direcionado. Este último grafo contém ainda um elemento nomeado laço, que é uma aresta do tipo (v_i, v_i) .

Figura 5.1: Exemplos de grafos

Um grafo $G' = (V', A')$ é um subgrafo de $G = (V, A)$ se $V' \subseteq V$ e $A' \subseteq A$. Se $V' = V$, ou seja, se G' contém todos os vértices de G , G' é um subgrafo gerador de G (Aluha et al., 1993). Na Figura 5.2 são ilustrados dois subgrafos do grafo da Figura 5.1a, sendo que na Figura 5.2b tem-se um subgrafo gerador do grafo da Figura 5.1a.

Uma sequência de vértices $v_1, v_2, \dots, v_q$ define um caminho em $G = (V, A)$ se A contém as arestas (v_i, v_{i+1}) , para todo $i = 1, \dots, q - 1$. No caso em que os vértices inicial e final coincidem, o caminho é fechado, sendo também denominado ciclo (Aluha et al., 1993). Nas Figuras 5.1a e 5.1b, a sequência v_2, v_3, v_4 forma um caminho nos grafos apresentados e a sequência v_2, v_3, v_4, v_2 compõe um ciclo. Se um grafo não contém ciclos, ele é referenciado como acíclico. As estruturas multiclasses hierárquicas DDAGs, ADAGs e de árvores apresentadas na Seção 3.1.2 são exemplos de grafos direcionados acíclicos.

Quando existe ao menos um caminho entre todos os pares de vértices distintos de um grafo não direcionado G , ele é conexo (Grimaldi, 2000). Caso contrário, ele é desconexo.

Figura 5.2: Exemplos de subgrafos do grafo ilustrado na Figura 5.1a

Se o grafo G é direcionado, obtém-se primeiramente um grafo não direcionado associado e verifica-se se este é conexo. O grafo associado é obtido ignorando as direções das arestas e também repetições de arestas que possam resultar do procedimento anterior. Se o grafo associado a G é conexo, G é considerado conexo. Os grafos apresentados nas figuras 5.1a, 5.1b e 5.2b são conexos, enquanto o grafo da Figura 5.2a é desconexo. Os DDAGs, ADAGs e as árvores da Seção 3.1.2 também são exemplos de grafos conexos.

Se $G = (V, A)$ é um grafo não direcionado conexo acíclico e sem laços, ele também é referenciado como uma árvore (Aluha et al., 1993). Um grafo direcionado G é uma árvore direcionada se o grafo não direcionado associado a G é uma árvore (Grimaldi, 2000). É importante salientar que uma árvore com q vértices possui $q - 1$ arestas.

Uma árvore direcionada pode ter uma raiz caso possua um único vértice que não tenha arestas direcionadas para si. Este vértice é então denominado raiz. Nas figuras 5.3a e 5.3b são apresentados exemplos de árvores direcionadas, sendo que na Figura 5.3b tem-se uma árvore com v_1 como vértice raiz. Os classificadores multiclasse hierárquicos com estrutura de árvore apresentados na Seção 3.1.2 também são exemplos de árvores direcionadas com raiz. Essas estruturas ainda podem ser classificadas como árvores direcionadas binárias com raiz, pois cada vértice tem no máximo duas arestas que incidem para outros vértices. Os DDAGs e ADAGs, por sua vez, não podem ser caracterizados como árvores, uma vez que os grafos não direcionados associados aos mesmos contêm ciclos.

Em uma árvore direcionada com raiz, é possível distinguir diferentes níveis de vértices. O nível de um vértice é definido como o número de arestas contidas no caminho percorrido da raiz a esse nó (Grimaldi, 2000). Na Figura 5.3b, por exemplo, o nível do nó v_4 é igual a 2.

Uma árvore T é uma árvore geradora de G se T é um subgrafo gerador de G . As árvores geradoras de um grafo G com q vértices possuem q vértices e $q - 1$ arestas de G . O subgrafo apresentado na Figura 5.2b, por exemplo, é uma árvore geradora do grafo da Figura 5.1a. A seguir, um tipo especial de árvore geradora é descrito. O procedimento de obtenção dessa árvore pode ser empregado na modelagem e solução de diversos problemas, como os que serão abordados nas seções 5.2 e 5.3.

Figura 5.3: Exemplos de árvores direcionadas

5.1.2 Árvore Geradora Mínima

Frequentemente, as arestas de um grafo possuem pesos associados. Esses pesos representam alguma informação da relação entre os vértices conectados. Se $G = (V, A)$ é um grafo conexo não direcionado com arestas ponderadas, a árvore geradora T^* que possui soma mínima dos pesos das arestas que a constituem é denominada árvore geradora mínima (AGM) de G (Wu & Chao, 2004). É importante salientar que um grafo pode conter mais de uma AGM, caso mais de uma árvore geradora de G possua arestas cuja soma dos pesos resulte no valor mínimo.

Um dos algoritmos existentes para encontrar uma AGM a partir de um grafo G é o Kruskal (Aluha et al., 1993). Esse algoritmo constrói a AGM iterativamente, conforme ilustrado no pseudo-código apresentado no Algoritmo 5.1. Ele mantém subconjuntos de vértices C_i que definem subgrafos em forma de árvore. Logo, tem-se várias árvores, que em conjunto recebem a denominação de floresta. Essas árvores são iterativamente conectadas até que, ao final do algoritmo, tem-se uma AGM. Nas Figuras 5.4b a 5.4f são apresentados os passos envolvidos na aplicação do algoritmo de Kruskal para a obtenção de uma AGM no grafo com quatro vértices da Figura 5.4a.

Inicialmente, define-se uma floresta formada por árvores com um único vértice (Figura 5.4b). Arestas são inseridas nessa floresta de acordo com o seu peso, em ordem crescente. A inserção de uma nova aresta implica na concatenação de duas árvores da floresta. Se os vértices de uma aresta encontram-se em uma mesma árvore, a sua adição à AGM resultaria na formação de um ciclo, o que deve ser evitado (Figura 5.4e). O passo 9 do Algoritmo 5.1 realiza essa checagem, impedindo a criação de ciclos. A união de dois subconjuntos no passo 10 representa a conexão de duas árvores da floresta. Esses procedimentos são iterados, implicando na adição de uma aresta por vez à AGM (passo 12). A AGM determinada para o grafo da Figura 5.4a encontra-se na Figura 5.4f.

Algoritmo 5.1 Algoritmo de Kruskal (Cormen et al., 2001)

- 1: Dado um grafo $G = (V, A)$ com $|V|$ vértices e $|A|$ arestas com pesos;
- 2: Ordenar as arestas em ordem crescente de peso;
- 3: **para todo** (vértice v_i) **faça**
- 4:   Definir um subconjunto $C_i = \{v_i\}$;
- 5: **fim-para**
- 6: Definir: $i = 0$, $j = 0$ e $T^* = \emptyset$;
- 7: **enquanto** ($i < |V| - 1$) **faça**
- 8:   Tomar a aresta a_j de acordo com a ordenação;
- 9:   **se** (os vértices de a_j estão em subconjuntos diferentes C_p e C_q) **então**
- 10:   Concatenar C_p e C_q ($C_p = C_p \cup C_q$);
- 11:   Remover C_q ($C_q = \emptyset$);
- 12:   Adicionar a aresta a_j a T^* ($T^* = T^* \cup \{a_j\}$);
- 13:   Incrementar i ;
- 14:   **fim-se**
- 15:   Incrementar j ;
- 16: **fim-enquanto**
- 17: Retornar a AGM T^* ;

Figura 5.4: Exemplo de operação do algoritmo de Kruskal

5.2 Exemplo de Aplicação de AGM: Agrupamento de Objetos

O problema de se encontrar uma AGM T^* a partir de um grafo G pode ser direta ou indiretamente utilizado para modelar e solucionar diversas aplicações práticas (Aluha et al., 1993; Wu & Chao, 2004). Um exemplo é determinar as conexões entre terminais de um circuito elétrico que reduzam gastos em cabeamento. O grafo neste caso possui como vértices os terminais elétricos e as arestas apresentam pesos que representam os comprimentos dos cabos necessários para conectar cada par de terminais.

Outro exemplo de aplicação prática é a realização de um agrupamento de objetos ou itens. O agrupamento pode ser definido como um particionamento dos objetos em subconjuntos, também denominados grupos, tal que os membros contidos em cada um deles apresentem alguma característica comum (Duda et al., 2001).

Existe uma divisão geral dos algoritmos de agrupamento em dois tipos: hierárquicos e particionais (Faceli et al., 2005). Os algoritmos hierárquicos organizam os grupos em camadas, resultando em uma seqüência aninhada de partições dos objetos. Algoritmos particionais, por sua vez, definem todos os grupos conjuntamente, obtendo como resultado uma única partição dos objetos em grupos.

Os agrupamentos realizados pelos algoritmos hierárquicos são comumente representados por um tipo especial de árvore binária denominada dendograma (Faceli et al., 2005). Exemplos de dendogramas são apresentados nas figuras 5.5f, 5.6 e 5.9. Os dendogramas possuem camadas de nós, os quais correspondem aos grupos. Linhas conectam nós de grupos que se encontram aninhados. Inicialmente tem-se um único grupo constituído por todos os itens. Esse grupo é então particionado em dois subgrupos, os quais são também sucessivamente divididos em duas partes até a formação de grupos com um único item. Cortes horizontais em um dendograma definem diferentes partições dos itens em grupos, de maior ou de menor granularidade.

Entre os algoritmos hierárquicos se distinguem ainda dois subtipos: aglomerativos e divisivos (Duda et al., 2001). Nos algoritmos aglomerativos tem-se inicialmente cada objeto em um grupo separado. Esses grupos são então sucessivamente unidos em grupos maiores. A construção da hierarquia neste caso opera de baixo para cima. Os algoritmos divisivos realizam o processo inverso. Eles iniciam com um grupo que contém todos os itens e dividem-o sucessivamente em grupos menores.

Pode-se definir os passos apresentados no Algoritmo 5.2 no processo de agrupamento hierárquico aglomerativo. Nos passos 4 e 6, a similaridade entre os grupos é normalmente medida a partir da semelhança dos objetos que eles contêm. Existem três tipos básicos de métricas de integração que podem ser empregadas no passo 6 para quantificar a similaridade entre os grupos (Faceli et al., 2005): ligação simples (*single-link*), ligação completa (*complete-link*) e ligação média (*average-link*). Elas são calculadas com base em medidas de similaridade entre pares (i, j) de objetos dos grupos sendo unidos, em que i pertence a um dos grupos, enquanto j pertence ao outro. No agrupamento por ligação simples, a

semelhança entre dois grupos é medida pela maior similaridade encontrada entre os pares de membros dos grupos considerados. Na ligação completa, emprega-se a menor similaridade entre os pares de itens dos grupos. Na ligação média, considera-se a similaridade média entre os membros dos dois grupos ligados.

Algoritmo 5.2 Agrupamento hierárquico aglomerativo (Kumar, 2000)

- 1: Dados n objetos a serem agrupados hierarquicamente;
  - 2: Atribuir cada item a um grupo;
  - 3: **repita**
  - 4:   Encontrar o par de grupos mais similar;
  - 5:   Concatená-los em um único grupo;
  - 6:   Computar a similaridade entre o novo grupo e os demais;
  - 7: **até que** (todos os objetos estejam contidos em um único grupo)
-

Em diversos casos a similaridade entre os itens é quantificada por alguma medida de distância entre eles. Neste caso, a ligação simples considera a distância entre dois grupos equivalente à distância mínima encontrada entre os pares de itens dos mesmos, uma vez que dessa forma tem-se uma maior relação de similaridade. Na ligação completa, por sua vez, emprega-se a maior distância entre os pares de itens dos dois grupos, enquanto na ligação média considera-se a média das distâncias desses pares.

O algoritmo de Kruskal pode ser diretamente empregado na realização de um agrupamento hierárquico aglomerativo por ligação simples, bastando para isso a realização de algumas adaptações, ilustradas no Algoritmo 5.3. Inicialmente, porém, os n objetos a serem agrupados devem ser organizados em um grafo ponderado com n vértices. Cada vértice representa um objeto individual. As arestas entre pares de vértices representam a distância entre eles, quantificando a sua similaridade. Na Figura 5.5 ilustra-se como o agrupamento realizado opera sobre quatro itens. O grafo ponderado que define as relações entre eles é o mesmo da Figura 5.4a.

O passo 2 do Algoritmo 5.2 é implementado nas etapas 3 a 6 do Algoritmo 5.3. Tem-se então todos os itens em grupos separados, representados pelos subconjuntos C_i (Figura 5.5b). O passo 4 do Algoritmo 5.2 é realizado pelos passos 9 e 10 do Algoritmo 5.3. Toma-se sempre uma aresta de peso baixo no grafo ponderado (passo 9). Verifica-se se esta contém como vértices itens em grupos que ainda estejam separados (passo 10). Em caso afirmativo, é possível uní-los em um grupo maior (passo 11 do Algoritmo 5.3). Como considera-se sempre arestas entre pares de itens que apresentem peso baixo para concatenar dois grupos, a métrica de integração empregada na computação da similaridade entre os grupos é a ligação simples, em que a relação de semelhança entre dois grupos é dada pela menor distância encontrada entre os pares de objetos contidos nos mesmos. O passo 6 do Algoritmo 5.2 é então implicitamente implementado pela aplicação das etapas 2 e 9 do Algoritmo 5.3. Os passos 9 a 15 do Algoritmo 5.3 são iterados até a obtenção de um único grupo C_i contendo todos os objetos. Tem-se como resultado um dendograma D , representado na Figura 5.5f para o grafo da Figura 5.5a.

Algoritmo 5.3 Modificação do algoritmo de Kruskal para o agrupamento de objetos

---

```

1: Dado um grafo  $G = (V, A)$  com  $n$  vértices e arestas com pesos;
2: Ordenar as arestas em ordem crescente de peso;
3: para todo (vértice  $v_i$ ) faça
4:   Definir um subconjunto  $C_i = \{v_i\}$ ;
5:   Adicionar  $C_i$  ao dendograma  $D$ ;
6: fim-para
7: Definir:  $j = 0$ ;
8: enquanto (existir mais de um  $C_i$ ) faça
9:   Tomar a aresta  $a_j$  de acordo com a ordenação;
10:  se (os vértices de  $a_j$  estão em subconjuntos diferentes  $C_p$  e  $C_q$ ) então
11:    Definir um novo nó no dendograma ligando  $C_p$  e  $C_q$ ;
12:     $\hookrightarrow$  O ramo esquerdo deste nó deve apontar o grupo  $C_p$  no dendograma e o ramo
13:    direito deve apontar para o grupo  $C_q$ ;
14:    Concatenar  $C_p$  e  $C_q$  ( $C_p = C_p \cup C_q$ );
15:    Remover  $C_q$  ( $C_q = \emptyset$ );
16:  fim-se
17:  Incrementar  $j$ ;
18: fim-enquanto
19: Retornar o dendograma  $D$ ;

```

Pode-se verificar pelas figuras 5.4 e 5.5 que as mesmas concatenações de subconjuntos de vértices C_i são realizadas pelos algoritmos 5.1 e 5.3. Logo, na prática a aplicação do Algoritmo 5.3 implica na ligação de vértices e arestas que formam uma AGM no grafo ponderado inicial fornecido como entrada para o algoritmo.

Pode-se visualizar assim o processo de agrupamento realizado de uma maneira alternativa. Seja T^* a AGM obtida para o grafo da Figura 5.5a, a qual encontra-se representada na Figura 5.4f. Cortando sucessivamente as arestas de T^* de maior peso, define-se as partições binárias dos itens no dendograma de cima para baixo, ou seja, do grupo contendo todos os objetos àqueles que possuem um único item. Esse processo é ilustrado na Figura 5.6.

A cada corte de arestas de T^* , obtém-se dois grupos de vértices e, conseqüentemente, de objetos. Como elimina-se as arestas de maior peso em ordem decrescente, inicialmente separa-se itens mais distantes na AGM e refina-se a divisão a objetos mais próximos. Tem-se assim uma versão divisiva do agrupamento aglomerativo por ligação simples realizado anteriormente (Kumar, 2000).

A seguir, na Seção 5.3 é descrita uma adaptação do algoritmo de Kruskal, similar à realizada nesta seção, que permite a definição de partições binárias de classes em um classificador multiclasses hierárquico com estrutura de árvore direcionada binária.

5.3 Obtenção de Classificadores Hierárquicos com AGM

Conforme discutido na Seção 5.1.1, um dos classificadores multiclasses hierárquicos apresentados na Seção 3.1.2 possui a estrutura de uma árvore direcionada binária com

Figura 5.5: Agrupamento hierárquico com adaptação do algoritmo de Kruskal

raiz. Para diferenciar as notações, esta árvore será referenciada daqui por diante como árvore multiclass.

Dado um problema com k classes, as árvores multiclass possuem $k - 1$ vértices correspondentes a classificadores binários e k vértices que representam as classes do problema. Cada classificador binário distingue dois subconjuntos distintos de classes. O nó raiz particiona todas as k classes em dois subconjuntos. Os nós internos são então definidos recursivamente, a partir das classes contidas em vértices de níveis anteriores. Dois exemplos de árvores multiclass para um problema com quatro classes foram apresentados na Figura 3.8.

Na Seção 3.1.2 também apontou-se que o desempenho de uma árvore multiclass em uma aplicação é influenciado por sua estrutura. Nesta seção é apresentada uma modificação do algoritmo de Kruskal que permite determinar a estrutura de uma árvore multiclass automaticamente, de acordo com o problema multiclass de interesse.

Inicialmente, informações coletadas dos dados de treinamento são utilizadas para construir um grafo com arestas ponderadas. Esse grafo possui k vértices, que representam as k classes do problema e $\frac{k(k-1)}{2}$ arestas entre todos os pares de vértices. Diferentes critérios podem ser utilizados para calcular os pesos das arestas, quantificando a relação entre as classes com base em medidas extraídas dos dados que as representam. A partir de um grafo obtido por algum desses critérios, o algoritmo proposto pode então ser empregado na geração de uma árvore multiclass. Com isso, a estrutura desta árvore é definida

Figura 5.6: Visão divisiva do agrupamento hierárquico realizado por uma AGM

de acordo com propriedades dos dados de cada aplicação. A seguir, são descritos alguns critérios que podem ser empregados na construção do grafo.

5.3.1 Definição do Grafo Ponderado

O primeiro passo para a aplicação do algoritmo proposto é obter um grafo com arestas ponderadas. Informações coletadas do conjunto de dados são utilizadas na definição desse grafo, que tem k vértices representando as k classes do problema multiclasses e $\frac{k(k-1)}{2}$ arestas conectando todos os pares de vértices. Para atribuir pesos às arestas, quantificando a relação entre cada par de classes, empregou-se alguns conceitos de trabalhos correlatos.

A primeira abordagem implementada para este cálculo considera as distâncias entre os centróides das classes, baseado em métodos empregados em (Takahashi & Abe, 2002; Vural & Dy, 2004). Nesse critério, cada classe i é inicialmente representada por um centróide μ_i , determinado pela Equação 5.1. Nessa equação, $\mathbf{x}_q$ é um dado e n_i constitui o número de exemplos pertencentes à classe i . O peso p_{ij} de uma aresta (i, j) entre duas classes i e j é então dado pela distância Euclidiana d_E entre os centróides dessas classes, segundo a Equação 5.2. Essa distância é apresentada na Equação 5.3, em que m representa o número de atributos dos dados. Dessa forma, os pesos das arestas representam distâncias entre centros de classes. Arestas que conectam classes com centros próximos possuem pesos menores em relação àquelas que conectam classes que apresentam centros distantes.

$$\mu_i = \frac{1}{n_i} \sum_{q=1}^{n_i} \mathbf{x}_q \quad (5.1)$$

$$p_{ij} = d_E(\mu_i, \mu_j) \quad (5.2)$$

$$d_E(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \sqrt{\sum_{q=1}^m (\mu_{iq} - \mu_{jq})^2} \quad (5.3)$$

O segundo critério é baseado no conceito de classes de confusão introduzido em (Schwenker, 2000; Schwenker & Palm, 2001). De maneira simplificada, uma classe de confusão pode ser definida como um subconjunto de classes cujos dados são similares. Segundo Schwenker (2000), uma forma de definir uma hierarquia de classes de confusão é pelo exame do conteúdo de matrizes de confusão.

Uma matriz de confusão quantifica quais classes um preditor tem maior dificuldade em distinguir. Para um conjunto de dados com k classes, ela tem dimensão $k \times k$, e cada elemento c_{ij} representa o número de exemplos da classe i que foram classificados erroneamente como pertencentes à classe j (Monard & Baranauskas, 2003a). No Apêndice A tem-se uma descrição e ilustração de uma matriz de confusão. Para obter essa matriz, deve-se empregar um classificador que solucione o problema total com k classes. Neste trabalho, Árvores de Decisão (ADs)¹ foram utilizadas na geração da matriz de confusão.

O peso p_{ij} de uma aresta (i, j) pode então ser calculado pela Equação 5.4. A variável $d_{MC}(i, j)$ é dada pela Equação 5.5, na qual cada termo representa a proporção de exemplos de uma das classes classificados incorretamente como pertencentes à outra classe. Dessa forma, este critério considera as confusões entre as classes segundo as ADs para definir os pesos das conexões entre as mesmas. Segundo a medida calculada, classes que apresentam maior confusão são mais similares e suas arestas recebem pesos pequenos, enquanto pesos maiores são atribuídos a arestas entre classes que apresentam pouca confusão.

$$p_{ij} = \frac{1}{d_{MC}(i, j)} \quad (5.4)$$

$$d_{MC}(i, j) = \frac{c_{ij}}{n_i} + \frac{c_{ji}}{n_j} \quad (5.5)$$

Na Figura 5.7 são ilustrados exemplos de grafos obtidos para um problema com três classes utilizando os dois critérios discutidos anteriormente. Os tamanhos das arestas não se encontram em escala de acordo com os seus pesos. Na Figura 5.7a encontra-se representado o critério de distâncias entre centróides, enquanto na Figura 5.7b é ilustrada uma matriz de confusão $\mathbf{C}$ e o grafo extraído a partir dessa matriz.

A aplicação dos critérios discutidos anteriormente geralmente leva a grafos distintos, uma vez que as características extraídas por cada um deles a partir dos dados para representar as relações entre as classes são diferentes. Com isso, as árvores multiclases obtidas utilizando cada um desses grafos podem ter estruturas diferentes. É apropriado então testar ambos os critérios e escolher aquele que melhor se ajuste à aplicação, ou seja, que

¹Uma conceituação a respeito das ADs pode ser encontrada em (Mitchell, 1997; Monard & Baranauskas, 2003b).

Figura 5.7: Grafos obtidos por diferentes critérios para um problema com três classes

gere o grafo capaz de sintetizar as informações das classes de maneira a produzir a árvore multiclasses mais adequada a solucionar o problema de interesse.

5.3.2 Algoritmo

Dado um grafo ponderado gerado por algum dos critérios apresentados anteriormente, uma versão adaptada do algoritmo de Kruskal, ilustrada no Algoritmo 5.4, pode ser aplicada na determinação de uma árvore multiclasses. Esse pseudo-código também foi apresentado em (Lorena & Carvalho, 2005a, 2006b). A geração da árvore multiclasses opera iterativamente de baixo para cima (das folhas para a raiz), conforme ilustrado no exemplo da Figura 5.8 para um problema com quatro classes. As modificações introduzidas são semelhantes às do Algoritmo 5.3, em que realiza-se um agrupamento hierárquico aglomerativo de objetos. Os objetos agrupados neste caso são as classes e as camadas e ramos do dendograma definem as partições binárias das classes para os classificadores binários que serão induzidos. O grafo da Figura 5.8a possui os mesmos pesos de arestas daqueles apresentados nas figuras 5.4a e 5.5a, permitindo visualizar mais claramente as modificações realizadas em relação ao algoritmo de Kruskal original e aquele empregado no processo de agrupamento hierárquico.

O algoritmo proposto é eficiente e permite que a estrutura da árvore multiclasses composta por preditores binários seja determinada de forma automática. Como nos algoritmos 5.1 e 5.3, o Algoritmo 5.4 mantém subconjuntos de vértices que são iterativamente agrupados. Inicialmente, cada subconjunto, constituído de uma única classe, é adicionado à árvore multiclasses como um nó folha. Considera-se então as arestas em ordem crescente de peso para definir as uniões a serem realizadas entre subconjuntos distintos de vértices.

A concatenação de dois subconjuntos C_p e C_q implica na inserção de um novo nó na árvore multiclasses (passo 11), o qual corresponde a um classificador binário. A inserção é realizada de forma que os vértices (classes) contidos em C_p assumam rótulo positivo e as classes em C_q possuam rótulo negativo na indução do preditor binário. Logo, a partição binária das classes em cada nó é definida pelos vértices contidos nos subconjuntos que são concatenados no passo 12.

Algoritmo 5.4 Algoritmo para a obtenção de árvore multiclass

- 1: Dado um grafo $G = (V, A)$ com k vértices e $\frac{k(k-1)}{2}$ arestas com pesos;
- 2: Ordenar as arestas em ordem crescente de peso;
- 3: **para todo** (vértice v_i) **faça**
- 4:   Definir um subconjunto $C_i = \{v_i\}$;
- 5:   Adicionar C_i à árvore multiclass A_m como folha;
- 6: **fim-para**
- 7: Definir: $i = 0$ e $j = 0$;
- 8: **enquanto** ($i < k - 1$) **faça**
- 9:   Tomar a aresta a_j de acordo com a ordenação;
- 10:   **se** (os vértices de a_j estão em subconjuntos diferentes C_p e C_q) **então**
- 11:   Adicionar um novo nó com conteúdo $C_p \times C_q$ a A_m ;
- 12: $\hookrightarrow$ Definir os vértices em C_p como a classe positiva e os vértices em C_q como a classe negativa;
- 13: $\hookrightarrow$ O ramo esquerdo deste nó deve apontar para o nó de A_m que contém os vértices em C_p e o ramo direito deve apontar para o nó com os vértices de C_q ;
- 14:   Concatenar C_p e C_q ($C_p = C_p \cup C_q$);
- 15:   Remover C_q ($C_q = \emptyset$);
- 16:   Incrementar i ;
- 17:   **fim-se**
- 18:   Incrementar j ;
- 19: **fim-enquanto**
- 20: Retornar a árvore multiclass A_m ;

Figura 5.8: Exemplo de operação do algoritmo para a geração de árvore multiclass

Caso no passo 9 seja selecionada uma aresta entre classes que se encontram em um mesmo subconjunto C_p , este fato indica que já existe um nó na árvore multiclasses que particiona essas classes. Além disso, a inserção de um nó do tipo $C_p \times C_p$ implicaria na geração de um classificador binário em que as classes em C_p assumiriam rótulos positivo e negativo simultaneamente. Este nó também estaria duplamente conectado àquele que contém os vértices de C_p , o que descaracterizaria a árvore multiclasses. Logo, quando os vértices de uma aresta encontram-se em um mesmo subconjunto, parte-se para uma nova aresta.

Os passos descritos anteriormente são iterados até a obtenção das $k - 1$ partições binárias de classes que deverão estar contidas na árvore multiclasses.

A aplicação do Algoritmo 5.4 em um grafo resulta nas mesmas concatenações de subconjuntos C_i que são realizadas no algoritmo de Kruskal básico e, conseqüentemente, também no algoritmo de agrupamento 5.3. A operação do Algoritmo 5.4 na obtenção das partições binárias de classes na árvore multiclasses pode então ser visualizada como um agrupamento hierárquico das classes. Cada concatenação de subconjuntos de vértices corresponde a um agrupamento, o qual define uma partição binária de classes, conforme ilustrado na Figura 5.9.

Figura 5.9: Agrupamento realizado pelo algoritmo na geração de uma árvore multiclasses

Os critérios de atribuição de pesos ao grafo inicial definem formas alternativas de quantificar a semelhança entre as classes no agrupamento realizado. No caso da distância entre centróides, agrupa-se classes similares de acordo com os seus centróides. No critério de classes de confusão, agrupa-se subconjuntos de classes consideradas similares a partir da confusão encontrada por ADs em sua separação. Cada um dos subconjuntos de classes formados neste processo de agrupamento constitui uma classe de confusão.

Um procedimento semelhante também foi adotado em (Lei & Govindaraju, 2005), que propôs o uso de um algoritmo de agrupamento hierárquico para definir as partições binárias das classes em uma árvore multiclasses. Esses autores empregaram como medida de similaridade entre duas classes a distância média entre os seus dados.

Como a geração da árvore multiclasses implica na ligação de vértices do grafo que definem uma AGM no mesmo, o algoritmo proposto pode ser ainda visualizado segundo

a visão de agrupamento divisivo apresentada na Seção 5.2. Seja T^* a AGM obtida para o grafo da Figura 5.8a. Cortando sucessivamente as arestas de T^* em ordem decrescente de peso, define-se as partições binárias da árvore multiclases de cima para baixo, ou seja, do nó raiz às folhas. Na Figura 5.10 esse procedimento é ilustrado. Nesta figura também é apresentado como um conjunto de dados com quatro classes é particionado por cada um dos nós definidos. Esse conjunto apresenta aproximadamente a mesma relação de distância entre as classes definida segundo os pesos do grafo.

Figura 5.10: Definição de árvore multiclases a partir de AGM

A aresta de maior peso na AGM T^* ilustrada conecta o vértice 4 ao vértice 1. Sua remoção implica na formação de dois subconjuntos, $\{4\}$ e $\{1, 2, 3\}$, os quais serão distinguidos pelo classificador binário do nó raiz. No conjunto de dados, esta operação implica na separação da classe 4 das demais, a qual é a classe mais afastada de 1 com relação à informação de distância calculada. No segundo passo, é definida a divisão do subconjunto $\{1, 2, 3\}$ em duas partes: $\{1, 2\}$ e $\{3\}$, por meio da retirada da aresta de T^* entre os vértices 2 e 3. Com isso, a classe 3 é separada das classes 1 e 2. A seguir, separa-se as classes 1 e 2, definindo o último classificador binário da árvore multiclases. Com esse procedimento, classes consideradas mais distantes na AGM são separadas no início e as partições passam a focar classes cada vez mais próximas. Logo, inicialmente realiza-se discriminações mais gerais, as quais são sucessivamente refinadas.

Como discutido anteriormente, de acordo com o critério empregado na atribuição de pesos às arestas do grafo inicial, as relações de distância entre as classes são definidas a partir de características distintas dos dados. Dessa maneira, diferentes propriedades dos dados são exploradas na organização da estrutura da árvore multiclasses. Com o uso das distâncias entre centróides, as partições binárias são refinadas com o propósito de focar a distinção de classes que possuam centros próximos. Assim, inicialmente particiona-se classes de centros mais distantes na AGM e, sucessivamente, foca-se os classificadores para classes com centros mais próximos. No critério de confusão entre classes, inicialmente distingue-se subconjuntos de classes que apresentem menor relação de confusão na AGM, ou seja, classes de confusão distintas. Refina-se então as divisões binárias dentro de cada classe de confusão determinada, particionando-as em subconjuntos de classes que apresentam cada vez mais confusão entre si.

Classes com centróides mais afastados em geral são mais separadas. Como consequência, um classificador geralmente apresenta maior facilidade na distinção de classes com centróides distantes. O mesmo ocorre entre classes que apresentam pouca relação de confusão. Na prática então o algoritmo proposto tende a posicionar partições binárias de maior dificuldade próximas às folhas, enquanto em níveis anteriores tem-se partições mais simples de serem realizadas, como também efetuado em (Takahashi & Abe, 2002).

5.3.3 Modificações no Algoritmo

Na definição da informação de distância para a formação da árvore multiclasses no Algoritmo 5.4, emprega-se relações entre pares de classes. Porém, as partições realizadas muitas vezes envolvem outras classes além do par considerado. Essa característica é agravada em níveis inferiores da árvore multiclasses, culminando no nó raiz, o qual deve dividir todas as classes do problema em duas partes. Na Figura 5.11a, por exemplo, tem-se um grafo para um problema com quatro classes, com arestas em escala de peso. A AGM extraída encontra-se na Figura 5.11b. Com a retirada da aresta de maior peso da AGM, separa-se a classe 4 das demais na raiz da árvore multiclasses. Porém, na média, pode-se verificar na Figura 5.11a que a distância da classe 3 às demais é maior.

Figura 5.11: Grafo para um problema com quatro classes e AGM extraída

Para verificar se este problema identificado acarreta prejuízos à formação da árvore

multiclasses, foi realizada uma modificação no Algoritmo 5.4, que permite considerar a informação conjunta de todas as classes que devem ser particionadas em um subconjunto. Nesta variação, quando dois subconjuntos de vértices do grafo são concatenados, a estrutura do grafo é adaptada de acordo com a alteração realizada. Os vértices equivalentes a essas classes são agrupados em um só, assim como as informações das classes extraídas dos dados. As conexões com os outros vértices são então recalculadas. Esse processo é ilustrado na Figura 5.12 para o grafo da Figura 5.11a. Tem-se na prática um novo algoritmo. Como no uso das AGMs, escolhe-se a cada passo uma aresta de peso reduzido no grafo, representando o agrupamento de subconjuntos de classes consideradas mais similares. Entretanto, este grafo é continuamente atualizado de acordo com as seleções realizadas.

O Algoritmo 5.4 é modificado da forma apresentada no Algoritmo 5.5. Cada concatenação de subconjuntos neste novo algoritmo implica na alteração do grafo (passo 12), que passa a conter um vértice apresentando as classes concatenadas. Os pesos de arestas em relação a esse novo vértice devem ser recalculados, enquanto as arestas entre os outros vértices são mantidas.

Os subconjuntos de vértices C_i possuem a utilidade de representar os vértices contidos no grafo corrente. Não é necessário testar se os vértices de uma aresta encontram-se em subconjuntos diferentes, pois a adaptação do grafo já representa a informação de quais classes foram separadas por algum nó da árvore multiclasses.

Como o grafo é continuamente alterado, toma-se no passo 9 sempre a aresta de peso mínimo. Na Figura 5.12 é apresentado um exemplo de operação desse algoritmo. Os valores de pesos recalculados são meramente ilustrativos. Pode-se observar que, a cada alteração, o grafo é reduzido.

No agrupamento das informações das classes nos vértices, estas são novamente calculadas a partir do conjunto de dados. No uso de centróides, calcula-se o centróide dos dados de todas as classes que constam no vértice concatenado. Os pesos das arestas entre o novo vértice e os demais presentes no grafo são então recalculados com base na distância desse novo centróide aos centróides que representam as classes contidas nos outros vértices. No caso do uso de uma matriz de confusão, gera-se uma nova matriz sobre um problema de aprendizado em que os dados das classes conjuntas passam a formar uma só classe. Como na obtenção da matriz inicial, emprega-se ADs na geração das novas matrizes de confusão. Os pesos das arestas no grafo são então recalculados com base nas informações da nova matriz de confusão formada, segundo a Equação 5.4.

A influência exercida por cada um dos critérios de atribuição de pesos às arestas do grafo na organização da árvore multiclasses é a mesma discutida na seção anterior, porém considera-se neste caso as informações de todas as classes contidas em um mesmo subconjunto para realizar as partições binárias e não somente de pares de classes.

A operação do novo algoritmo descrito também pode ser visualizada como um agrupamento hierárquico aglomerativo das classes. Entretanto, a cada concatenação de classes

Figura 5.12: Processo de obtenção de árvore multiclasses por meio de adaptação no algoritmo

Algoritmo 5.5 Algoritmo modificado para a obtenção de árvore multiclasses

- 1: Dado um grafo $G = (V, A)$ com k vértices e $\frac{k(k-1)}{2}$ arestas com pesos;
- 2: **para todo** (vértice v_i) **faça**
- 3:   Definir um subconjunto $C_i = \{v_i\}$;
- 4:   Adicionar C_i à árvore multiclasses A_m como folha;
- 5: **fim-para**
- 6: Definir: $i = 0$;
- 7: **enquanto** ($i < k - 1$) **faça**
- 8:   Tomar a aresta a de menor peso no grafo G ;
- 9:   Sejam C_p e C_q os subconjuntos em que os vértices de a estão;
- 10:   Adicionar um novo nó com conteúdo $C_p \times C_q$ a A_m ;
      $\hookrightarrow$ Definir os vértices em C_p como a classe positiva e os vértices em C_q como a classe negativa;
      $\hookrightarrow$ O ramo esquerdo deste nó deve apontar para o nó de A_m que contém os vértices em C_p e o ramo direito deve apontar para o nó com os vértices de C_q ;
- 11:   Atualizar o grafo G ;
      $\hookrightarrow$ Concatenar os vértices de C_p e C_q em um novo nó;
      $\hookrightarrow$ Recalcular os pesos de arestas em relação a este novo nó;
- 12:   Concatenar C_p e C_q ($C_p = C_p \cup C_q$);
- 13:   Remover C_q ($C_q = \emptyset$);
- 14:   Incrementar i ;
- 15: **fim-enquanto**
- 16: Retornar a árvore multiclasses A_m ;

realizada, o novo grupo passa a ser representado por uma medida que considera informações de todos os itens (classes) contidos no mesmo. A similaridade desse grupo em relação aos demais é então definida a partir dessa medida conjunta.

5.4 Considerações Finais

Neste capítulo foram apresentados dois algoritmos para determinar a estrutura de uma árvore multiclases a partir de informações extraídas dos dados pertencentes a cada classe. Os algoritmos propostos partem de um grafo com arestas ponderadas segundo essas informações, determinando as partições binárias das classes nos nós da árvore pela escolha de arestas de peso reduzido. A operação desses algoritmos pode ser vista como um agrupamento hierárquico das classes de acordo com a sua similaridade. Cada agrupamento realizado define as separações binárias de classes efetuadas nos nós da árvore multiclases. Como resultado, inicialmente realiza-se discriminações mais gerais de acordo com as informações de similaridade calculadas, as quais são sucessivamente refinadas. Diferentes critérios podem ser empregados na atribuição de pesos às arestas do grafo, representando a extração de características alternativas dos dados para quantificar as semelhanças entre as classes.

O primeiro algoritmo proposto é uma adaptação direta do algoritmo de Kruskal para a obtenção de uma AGM a partir de um grafo ponderado. O segundo algoritmo introduz modificações no anterior, que implicam na adaptação da estrutura do grafo a medida que a árvore multiclases é formada. Com o desenvolvimento desses algoritmos, tem-se uma proposta de solução do terceiro problema identificado no Capítulo 3, que refere-se à definição da estrutura de uma árvore multiclases de acordo com os dados de cada aplicação.

Além do uso da distância entre centróides e da confusão entre as classes segundo ADs, outros critérios também podem ser utilizados na ponderação da relação entre as classes. Alguns foram investigados em experimentos reportados em (Lorena & Carvalho, 2005a,b; Ushizima et al., 2005; Lorena & Carvalho, 2006b). Após alguns testes, optou-se por apresentar nesta Tese aqueles que demonstraram os melhores resultados práticos.

Capítulo 6

Experimentos

Neste capítulo, as estratégias para a obtenção de previsões multiclasss com SVMs apresentadas nos capítulos anteriores são avaliadas por meio de análises experimentais. As técnicas mencionadas foram empregadas na solução de aplicações reais da área de Bioinformática e de problemas multiclasss provenientes de *benchmarks*, selecionados a partir de trabalhos correlatos.

Os experimentos e análises realizados podem ser divididos basicamente em quatro grupos. No primeiro, analisou-se os resultados dos AGs para a obtenção das matrizes de códigos descritos no Capítulo 4 frente a estratégias baseadas em matrizes de códigos tradicionais. No segundo grupo, as soluções obtidas pelos AGs para a determinação da estrutura de DDAGs e ADAGs foram comparadas a estruturas definidas aleatoriamente. Em seguida, analisou-se o desempenho das estratégias para a definição de árvores multiclasss do Capítulo 5, avaliando a efetividade das heurísticas propostas na determinação das partições binárias de classes nos nós da árvore. Representantes de cada um dos grupos anteriores e duas variantes de SVMs que solucionam o problema multiclasss diretamente foram então comparados.

Este capítulo está organizado da seguinte maneira. Na Seção 6.1, os conjuntos de dados empregados no trabalho experimental realizado são descritos. Na Seção 6.2 é apresentada a metodologia utilizada na condução dos experimentos. Nas seções 6.4 a 6.6 os resultados obtidos em cada grupo de experimentos anteriormente mencionado são apresentados e analisados. Uma discussão geral dos resultados verificados e outras análises são realizadas na Seção 6.7. Finalizando este capítulo, algumas considerações finais são apontadas na Seção 6.8.

6.1 Conjuntos de Dados

As principais características dos conjuntos de dados empregados nos experimentos conduzidos neste trabalho podem ser visualizadas na Tabela 6.1. Nesta tabela são apresentados, para cada conjunto, o número de dados de treinamento ($\#$ Trein.), o número de dados de teste ($\#$ Teste), o número de classes do problema ($\#$ Classes), o número de atributos contínuos e nominais ($\#$ Atrib. cont./nom.), os números médio, mínimo e máximo de exemplos por classe ($\#$ méd./min./máx. ex./classe) e a taxa de acerto majoritária (TAM). TAM corresponde à taxa de acerto de um classificador que sempre produz como saída a classe com o maior número de dados, não apresentando, portanto, capacidade de generalização.

Conjunto	$\#$ Trein.	$\#$ Teste	$\#$ Classes	$\#$ Atrib. cont./nom.	$\#$ méd./min./máx. ex./classe	TAM
<i>iris</i>	150	-	3	4/0	50/50/50	33,3
<i>car</i>	1728	-	4	0/6	432/65/1210	70,0
<i>glass</i>	214	-	6	9/0	35,7/9/76	35,5
<i>sat-image</i>	4435	2000	6	36/0	739,2/415/1072	23,5
<i>segment</i>	2310	-	7	19/0	330/330/330	14,3
<i>opt-digits</i>	3823	1797	10	64/0	382,3/376/389	10,2
<i>pen-digits</i>	7494	3498	10	16/0	749,4/719/780	10,4
<i>vowel</i>	528	462	11	10/0	48/48/48	9,1
<i>structural</i>	313	385	4	125/0	78,3/34/115	46,3
<i>lung</i>	203	-	5	12600/0	40,6/6/139	68,5
<i>leukocyte</i>	1055	-	8	62/0	131,9/44/284	26,9
<i>fungi</i>	2355	-	9	40/0	261,7/34/814	34,6

Tabela 6.1: Principais características dos conjuntos de dados

Alguns conjuntos de dados não possuem subconjuntos de teste definidos, o que foi indicado na tabela pelo símbolo “-”. Nos conjuntos que apresentavam as divisões de treinamento e teste, os números de exemplos por classe foram calculados em relação ao subconjunto de treinamento, enquanto TAM foi definida a partir do subconjunto de teste. Na Seção B.1 do Apêndice B, as proporções de dados por classe em todos os conjuntos de dados apresentados são detalhadas.

Da primeira à oitava linha da Tabela 6.1 encontram-se conjuntos de dados provenientes do repositório UCI (Blake & Merz, 1998), o qual provê *benchmarks* para a área de AM. Decidiu-se manter o nome original desses conjuntos em Inglês, pois eles são largamente conhecidos por essas nomenclaturas. Em seguida, tem-se conjuntos de dados reais, os quais constituem aplicações multiclasse da área de Bioinformática. Apesar dos conjuntos de dados do repositório UCI representarem aplicações reais, eles foram manipulados e muitas vezes pré-processados em trabalhos anteriores. Por este motivo, é comum distinguí-los dos conjuntos de dados reais com a nomenclatura de naturais.

Tem-se assim avaliações em duas categorias de dados. Os conjuntos de *benchmarks*, os quais serão referenciados por naturais, têm a utilidade de prover análises genéricas, em

aplicações multiclasse de domínios variados. Os resultados alcançados nesses conjuntos podem ser facilmente reproduzidos e comparados frente a trabalhos correlatos. Os dados reais permitem avaliar o uso prático dos algoritmos codificados. A seguir são apresentadas descrições mais detalhadas dos conjuntos de dados empregados e das aplicações que eles representam.

6.1.1 Conjuntos de Dados Naturais

Estes conjuntos de dados foram selecionados entre os providos pelo repositório da UCI (Blake & Merz, 1998). Procurou-se escolher dados utilizados em trabalhos correlatos que avaliam a geração de predições multiclasse a partir de classificadores binários, como os empregados em (Hsu & Lin, 2002; Fürnkranz, 2002; Rifkin & Klautau, 2004). A seguir, esses conjuntos e as aplicações que eles representam são brevemente descritos. Maiores detalhes podem ser consultados em <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

► *Iris*

Este conjunto de dados é empregado em diversos trabalhos de AM, por possuir características e distribuição dos dados conhecidas. A partir de quatro atributos numéricos medindo os comprimentos e as larguras das sépalas e pétalas de flores de plantas *iris*, o objetivo é classificá-las em um entre três tipos: “*setosa*”, “*versicolor*” e “*virginica*”. Esses tipos constituem as três classes do problema. A classe “*setosa*” é linearmente separável das outras duas, as quais por sua vez não são linearmente separáveis. Cada classe possui 50 dados.

► *Car*

Este conjunto apresenta dados para a avaliação de carros. Os atributos constituem propriedades e características dos carros, como o seu preço de compra e os seus níveis de conforto e de segurança. A partir deles, um índice de aceitação é atribuído, que pode ser: “não aceitável” (*unacceptable*), “aceitável” (*acceptable*), “bom” (*good*) ou “muito bom” (*very good*). Tem-se então quatro classes, sendo que a maioria (70%) dos dados encontra-se na classe “não aceitável” e a minoria (3,8%) na classe “muito bom”. Tem-se assim um desbalanceamento no número de exemplos por classe.

► *Glass*

Deseja-se neste caso classificar vidros entre seis possíveis tipos: “vidro de edifício processado” (*building window float processed*), “vidro de edifício não processado” (*building window non float processed*), “vidro de carro processado” (*vehicle window float processed*), “recipiente” (*container*), “louça” (*tableware*) e “farol” (*headlamps*). Os atributos são características físico-químicas dos vidros, como o seu índice de refração e a porcentagem de elementos químicos contidos nos mesmos. A classe majoritária, com 35,5% dos dados, é a “vidro de edifício não processado” e a minoritária é a “louça”, com uma proporção

de 4,2% dos dados. Os números de exemplos por classe neste conjunto também podem ser considerados desbalanceados, embora em menor proporção que no conjunto de dados anterior.

► *Sat-image*

Este conjunto de dados, utilizado no projeto Statlog¹ (Michie et al., 1994), contém fragmentos de 3x3 pixels de parte de uma imagem de satélite com 82x100 pixels. Deve-se classificar o pixel central de cada um desses fragmentos em uma de seis categorias, as quais descrevem o solo ou a vegetação presente na área. São elas: “solo vermelho” (*red soil*), “colheita de algodão” (*cotton crop*), “solo cinza” (*grey soil*), “solo cinza umedecido” (*damp grey soil*), “solo com resíduos de vegetação” (*soil with vegetation stubble*) e “solo cinza bastante umedecido” (*very damp grey soil*). Para cada pixel, tem-se quatro valores de bandas espectrais. Cada fragmento é então constituído de $9 \times 4 = 36$ atributos numéricos. Os dados encontram-se divididos em subconjuntos de treinamento e de teste. O maior número de exemplos no subconjunto de treinamento possui “solo vermelho” em seu pixel central. No subconjunto de teste a maior parte dos exemplos é da classe “solo cinza bastante umedecido”. A classe minoritária nos subconjuntos de treinamento e de teste é a de “solo cinza umedecido”. As proporções de dados por classe em ambos os subconjuntos são levemente desbalanceadas, variando de 9,4% a 24,2% no subconjunto de treinamento e de 10,6% a 23,5% nos dados de teste.

► *Segment*

Este conjunto de dados possui imagens de sete tipos de ambientes ao ar livre: “paramento de tijolos” (*brickface*), “céu” (*sky*), “folhagem” (*foliage*), “cimento” (*cement*), “janela” (*window*), “estrada” (*path*) e “grama” (*grass*). O tamanho das imagens é de 3x3 e elas foram segmentadas à mão, sendo representadas por 19 atributos numéricos. Este conjunto também foi utilizado no projeto Statlog (Michie et al., 1994). Ele é balanceado, possuindo 330 dados por classe.

► *Opt-digits*

Este é um problema de classificação de dígitos manuscritos. Tem-se neste conjunto dados dígitos escritos por 43 diferentes pessoas. Os dígitos de 30 delas foram reunidos em um subconjunto de treinamento, enquanto os das 13 pessoas restantes encontram-se em um subconjunto de teste independente. Na obtenção dos atributos dos dados, imagens de 32x32 bits de cada dígito manuscrito foram divididas em blocos de 4x4 e o número de pixels com alguma informação foram contados em cada bloco. Tem-se então uma matriz de dimensão 8x8 com elementos inteiros assumindo valores de 0 a 16, constituindo assim os 64 atributos numéricos extraídos de cada dígito. Das dez classes do problema, correspondentes aos dígitos “0” a “9”, os dígitos “1” e “3” prevalecem no subconjunto de

---

¹O projeto Statlog avaliou o uso de diversas técnicas de AM em problemas de classificação variados (Michie et al., 1994).

treinamento, enquanto entre os dados de teste a classe majoritária é a do dígito “3”. No geral, porém, os números de dados nas classes são similares e esses subconjuntos encontram-se balanceados.

► *Pen-digits*

Tem-se novamente um problema de classificação de dígitos manuscritos. Os dígitos foram escritos por 44 pessoas diferentes. Separou-se para treinamento os dados referentes aos dígitos de 30 delas, enquanto os das 14 restantes foram reunidos em um subconjunto de teste independente. Os dados foram coletados a partir de um sistema de LCD para escrita integrado a um computador. Foram amostradas 8 coordenadas (x, y) de pontos do tablado empregado na escrita, formando um total de 16 atributos. As classes majoritárias no subconjunto de treinamento correspondem aos dígitos “0”, “2” e “4”. No subconjunto de teste, tem-se um maior número de exemplos nas classes “1”, “2”, “4” e “7”. Este conjunto também apresenta números semelhantes de dados por classe, sendo então balanceado.

► *Vowel*

Tem-se neste conjunto de dados sinais sonoros de vogais a serem classificados entre onze tipos de fonemas de Inglês britânico. Há sinais produzidos por 15 diferentes pessoas (8 homens e 7 mulheres), as quais pronunciaram cada vogal seis vezes. Os fonemas registrados a partir de 8 pessoas (4 homens e 4 mulheres) foram reunidos em um subconjunto de treinamento, enquanto os das 7 pessoas (4 homens e 3 mulheres) restantes encontram-se em um subconjunto de teste independente. Ambos os subconjuntos de treinamento e de teste são balanceados. Os 10 atributos dos dados foram extraídos a partir das seqüências sonoras por um procedimento denominado *log-ratio*, descrito em (Robinson, 1989).

6.1.2 Conjuntos de Dados Reais

A Bioinformática é uma área de pesquisa que vem despertando crescente interesse na comunidade científica. Esse novo campo interdisciplinar consiste na aplicação de técnicas matemáticas, estatísticas e computacionais para a resolução de problemas provenientes da Biologia (Setubal & Meidanis, 1997). Dentre as diversas áreas da Biologia, aquela em que a aplicação de técnicas computacionais tem se mostrado mais frutífera é a Biologia Molecular, principalmente sua parte relacionada à genética.

Os estudos em genética possibilitam avanços em várias áreas, entre elas a Medicina, a Farmácia e a 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.

Técnicas de AM vêm sendo empregadas na resolução de diversos problemas práticos em Bioinformática (Souto et al., 2003; Lorena & Carvalho, 2003f). Neste trabalho, foram selecionados conjuntos de dados que cobrem três diferentes aplicações multiclasses dessa área: a classificação estrutural de proteínas, o diagnóstico de câncer a partir de dados de expressão gênica e a localização de proteínas no meio celular. Essas aplicações apresentam

características que representam alguns desafios na predição multiclass. Entre elas, pode-se destacar o desbalanceamento nas proporções de dados por classe e a presença de ruídos e de números elevados de atributos.

Em geral, os diversos trabalhos da literatura que empregam SVMs na solução desses problemas utilizam as estratégias 1ct ou tct na geração de previsões multiclass. Neste trabalho diversas estratégias, incluindo as propostas neste Doutorado, foram avaliadas para esses problemas. Os resultados dos experimentos com essas aplicações vêm contribuindo também para projetos em Bioinformática em andamento no ICMC-USP.

► Classificação Estrutural de Proteínas

As proteínas são componentes essenciais de todo ser vivo, possuindo diversas funções biológicas (Lewis, 2001). Elas podem ter papel estrutural, como no caso do colágeno presente nos tendões, ou estar ligadas a atividades regulatórias, como no caso das enzimas. Essas moléculas são seqüências lineares, compostas por conjuntos de aminoácidos. Existe um total de 20 tipos de aminoácidos, que se combinam em uma seqüência característica para cada proteína. A composição desta seqüência é denominada estrutura primária da proteína (Minatti, 2003). Ações mútuas entre esses aminoácidos, por meio de interações moleculares, formam outros dois tipos de estruturas: secundária e terciária. As três conformações estruturais mencionadas são ilustradas na Figura 6.1².

Figura 6.1: Conformações estruturais das proteínas

A estrutura secundária é definida pelos ângulos formados por ligações entre os aminoácidos da proteína. Quando essas ligações fazem a proteína assumir uma forma helicoidal, tem-se uma estrutura denominada hélice alfa. Outras duas estruturas secundárias são as folhas beta, em que segmentos da cadeia de aminoácidos interagem paralelamente, e as voltas, que revertem a cadeia peptídica (Vouet et al., 2002). Dobras da cadeia proteica sobre ela mesma compõem a estrutura terciária da proteína (Minatti, 2003). Esta corresponde à estrutura tridimensional da proteína, definindo a sua forma e, conseqüentemente, a sua função.

De acordo com o tipo, a quantidade e a organização de seus elementos estruturais secundários, as proteínas podem ser genericamente categorizadas em classes estruturais (Chou & Zhang, 1995). Segundo Luo et al. (2002), essa classificação possui utilidades

²Esta figura foi extraída de http://en.wikipedia.org/wiki/Protein_structure.

teóricas e práticas, uma vez que ela provê uma descrição geral e intuitiva da forma das proteínas. Técnicas de AM vêm sendo empregadas com sucesso na solução desse problema de classificação (Cai et al., 2001, 2002; Luo et al., 2002; Isik et al., 2004).

O conjunto de dados de proteínas empregado neste trabalho foi extraído de <http://www.nersc.gov/protein>, sendo coletado e organizado por Ding & Dubchak (2001). O subconjunto de treinamento contém proteínas com identidade sequencial menor que 35% entre si. Ding & Dubchak (2001) também provêm um subconjunto de teste independente, com proteínas com menos de 40% de identidade entre si e com menos de 35% de similaridade sequencial com as proteínas do subconjunto de treinamento. Um total de 125 atributos numéricos foram extraídos das seqüências proteicas, representando informações quanto à composição e a propriedades físico-químicas das mesmas. Originalmente, Ding & Dubchak (2001) utilizaram esse conjunto para classificar tipos de dobras contidos nas proteínas.

Há quatro classes estruturais nesse conjunto de dados, o qual é indicado na Tabela 6.1 pelo nome “*structural*”: “*all- α* ”, “*all- β* ”, “ *α/β* ” e “ *$\alpha + \beta$* ”. Uma proteína é classificada como “*all- α* ” se possui predominantemente hélices alfa em sua estrutura. O mesmo raciocínio se aplica à classe “*all- β* ” para as folhas beta. A classe “ *α/β* ” é constituída de proteínas que apresentam hélices alfa e folhas beta intercaladas. Por fim, a classe “ *$\alpha + \beta$* ” reúne proteínas que contêm hélices alfa e folhas beta bastante separadas (Murzin et al., 1995).

A classe majoritária nos subconjuntos de treinamento e de teste é a “ *α/β* ”. A classe minoritária no subconjunto de treinamento é a “ *$\alpha + \beta$* ” e no subconjunto de teste é a “*all- α* ”. A proporção de exemplos por classe no subconjunto de treinamento varia de 10,9% a 36,7%, enquanto no subconjunto de teste a variação é de 15,8% a 37,7%. Logo, tem-se um leve desbalanceamento na proporção de exemplos por classe.

Em (Cai et al., 2002) e (Isik et al., 2004), SVMs integradas com as estratégias lct e tct, respectivamente, foram aplicadas na solução desse problema de classificação. Porém, os conjuntos de dados utilizados em ambos os trabalhos, assim como os atributos extraídos das proteínas, foram diferentes dos empregados nesta Tese.

► Diagnóstico de Tipos de Câncer

O tratamento de pacientes com câncer é geralmente baseado na determinação do local de origem do tumor identificado (Yeang et al., 2001). Esse tipo de diagnóstico é em geral feito por exames microscópicos, genéticos e imunológicos. Porém, além de subjetivas, as técnicas utilizadas nesses exames são limitadas quando é necessária a distinção de diversos tipos de tumores. Em alguns casos, essa tarefa torna-se impossível, devido à presença de amostras tumorais com morfologias atípicas (Ramaswamy et al., 2001). Além disso, diversas amostras de tecidos cancerígenos provenientes de tipos de tumores distintos mostram pouca diferenciação, dificultando a sua identificação laboratorial. A tecnologia para medir o nível de expressão gênica em tecidos oferece meios de se analisar múltiplos tipos de tumores, e vem se tornando uma ferramenta promissora para o diagnóstico de

câncer (Ramaswamy et al., 2001; Yeang et al., 2001; Ooi & Tan, 2003; Rifkin et al., 2003; Statnikov et al., 2005).

Os genes são porções das moléculas de DNA (Ácido Desoxirribonucleico), presentes nas células, que armazenam as informações genéticas dos organismos. Sua propriedade mais importante está no fato de que eles codificam proteínas, por um processo denominado expressão gênica. O nível de expressão de um gene constitui então uma estimativa da quantidade de proteínas por ele produzidas em um determinado intervalo de tempo. Experimentos para medir o nível de expressão gênica são usualmente realizados para monitorar um conjunto de genes em diferentes condições (de pacientes com e sem tratamento por uma nova droga, por exemplo) ou comparar a expressão de um gene em diferentes tipos de tecidos (normal e canceroso, por exemplo) (Furey et al., 2000).

Em (Statnikov et al., 2005), os algoritmos k-vizinhos mais próximos (Mitchell, 1997), RNAs do tipo MLP (Braga et al., 2000), RNAs probabilísticas e SVMs, integradas por diversas estratégias multiclases, foram empregadas na diferenciação de múltiplos tipos de tumores. Os resultados indicam que as SVMs multiclases que utilizam os algoritmos de Weston & Watkins (1998), Crammer & Singer (2002) e a estratégia 1ct podem ser consideradas mais adequadas para a solução desse tipo de problema de classificação do que as estratégias tct e DDAG.

Selecionou-se nesta Tese um dos conjuntos de dados empregados no trabalho anterior. Ele foi coletado a partir do endereço <http://www.gems-system.org>. O conjunto escolhido entre os disponíveis neste endereço foi o de tecidos de pulmão (“lung” na Tabela 6.1), o qual apresentava a maior relação entre o número total de dados e o número de classes. Ele contém medidas de expressão de genes em tecidos de pulmão, os quais podem ser normais ou categorizados entre quatro tipos de tumores: “adenocarcinoma de pulmão definido histologicamente” (*histologically defined lung adenocarcinomas*), “carcinomas de pulmão de células escamosas” (*squamous cell lung carcinomas*), “carcinóides pulmonares” (*pulmonary carcinoids*) e “carcinomas de pulmão de células pequenas” (*small-cell lung carcinomas*). A classe majoritária é a de “adenocarcinoma de pulmão definido histologicamente” e a minoritária é a de “carcinomas de pulmão de células pequenas”. A proporção de exemplos por classe é desbalanceada e varia de 3,0% a 68,5%. Deve-se observar que esse conjunto de dados apresenta um número elevado de atributos, ou seja, os tecidos analisados contêm medidas da expressão de milhares de genes.

Em cooperação com a Profa. Dra. Daniela Ushizima da Universidade Católica de Santos, foi obtido um outro conjunto de dados para o diagnóstico de câncer. Ele é constituido de imagens de leucócitos, coletadas no Hemocentro de Ribeirão Preto. Os leucócitos são células de defesa presentes no sangue. O diagnóstico de leucemias pode ser realizado por meio da análise dessas células. Em seu Doutorado, Ushizima (2004) extraiu 62 atributos numéricos descrevendo características de cor, textura e forma dessas células por meio de técnicas de segmentação e de processamento de imagens. Um sistema automatizado foi então construído para auxiliar o processo de análise de imagens de leucócitos e, com isso,

realizar o diagnóstico de leucemias (Sabino et al., 2004a,b). Empregou-se classificadores Naïve-Bayes (NB) (Mitchell, 1997) na classificação das células.

Na cooperação estabelecida, estuda-se a utilização de SVMs na solução dessa tarefa multiclass. Experimentos preliminares com uma parte desse conjunto de dados foram publicados em (Ushizima et al., 2005). O conjunto de dados empregado nesta Tese, nomeado “*leukocyte*” na Tabela 6.1, possui oito classes de células: “basófilos”, “eosinófilos”, “linfócitos”, “monócitos”, “neutrófilos”, “leucemia linfóide crônica”, “prolinfócitos” e “células cabeludas” (*hairy cell*). As cinco primeiras caracterizam tipos normais de leucócitos, enquanto as três restantes constituem tipos malignos associados à presença de leucemias. A maior parte das células presentes nesse conjunto é do tipo “linfócitos”, enquanto o tipo “basófilos” apresenta a minoria dos dados. A proporção de exemplos por classe varia de 4,2% a 26,9%.

► Localização de Proteínas

As proteínas podem estar localizadas em várias regiões da célula ou ser transportadas para o meio extracelular (Feng, 2002). Como várias funções celulares são realizadas pelas proteínas em compartimentos específicos, a identificação de sua localização é importante para o entendimento de sua função.

A determinação laboratorial da localização de uma proteína envolve técnicas custosas, subjetivas e muito instáveis (Feng, 2002). Logo, o desenvolvimento de sistemas computacionais para realizar essa tarefa possui grande relevância. Vários trabalhos aplicaram técnicas de AM na solução deste problema multiclass (Horton & Nakai, 1997; Emanuelsson et al., 2000; Hua & Sun, 2001; Cui et al., 2004; Lu et al., 2004; Garg et al., 2005). Os sistemas desenvolvidos podem ser genericamente divididos em três grupos, de acordo com as informações das proteínas que eles utilizam no processo de reconhecimento. Os grupos são (Cui et al., 2004): baseados em sinais peptídicos, baseados na composição de aminoácidos e baseados em homologia.

Os sinais peptídicos são porções das seqüências proteicas responsáveis por encaminhá-las à sua localização específica (Emanuelsson et al., 2000). Logo, a identificação desses sinais pode ser utilizada no reconhecimento de algumas localidades. Esse foi o procedimento adotado por Emanuelsson et al. (2000), que empregaram RNAs no reconhecimento de sinais associados a três localizações celulares: “mitocôndria”, “cloroplasto” e “caminho secretório”. Porém, como apontado em (Hua & Sun, 2001), métodos baseados na identificação de sinais falham quando eles encontram-se omitidos ou estão apenas parcialmente incluídos. Além disso, os sinais conhecidos não são gerais o suficiente para identificar diversas localidades.

A composição de aminoácidos em uma proteína pode ser indicada por um vetor que apresenta a proporção dos 20 possíveis aminoácidos em sua seqüência. Hua & Sun (2001) empregaram este vetor com SVMs e a estratégia 1ct na localização de proteínas em organis-

mos procariotos e eucariotos³. Porém, a representação por esse vetor tem dois problemas. Primeiro, ela ignora todo o efeito da ordem dos aminoácidos na sequência da proteína na previsão de sua localidade. Segundo, diferentes proteínas podem ser mapeadas para o mesmo vetor de composição, ou seja, elas podem ter a mesma composição, embora os seus aminoácidos encontrem-se em ordens distintas.

Em (Cui et al., 2004), uma variação simples do vetor de composição de aminoácidos foi utilizada para desenvolver um classificador capaz de distinguir oito localizações de proteínas em eucariotos. Empregou-se SVMs com a estratégia tct na geração do preditor multiclass. Na representação adotada, inicialmente a sequência da proteína é dividida em duas metades. Em seguida, um vetor de composição de aminoácidos é produzido para cada metade. Estes dois vetores são então concatenados para produzir a representação final da proteína. Esse esquema de codificação também foi empregado no presente trabalho.

Entre os sistemas que utilizam uma abordagem baseada em homologia, pode-se citar o desenvolvido por Lu et al. (2004). Neste trabalho, dada uma sequência proteica, uma busca por cadeias similares é realizada na base de proteínas SWISS-PROT (Apweiler et al., 1999). Um conjunto de informações textuais é extraído dos homólogos encontrados. Classificadores NB são então utilizados na predição da classe da proteína a partir da informação coletada. Porém, Garg et al. (2005) alertam que os métodos baseados em similaridade falham quando a proteína de interesse não possui homólogos conhecidos.

Como alternativa à utilização de uma única estratégia para representar as proteínas, alguns trabalhos combinam vários tipos de atributos coletados das sequências. Isso é realizado, por exemplo, em (Horton & Nakai, 1997) e (Garg et al., 2005).

O conjunto de dados selecionado neste trabalho para a localização de proteínas, indicado na Tabela 6.1 por “*fungi*”, foi extraído de <http://www.cs.ualberta.ca/~bioinfo/PA/Subcellular>. Ele foi pré-processado segundo o método de Cui et al. (2004) para a extração de atributos numéricos a partir das sequências proteicas. Adotou-se esta codificação por sua simplicidade e capacidade de considerar alguma informação de ordem dos aminoácidos nas sequências. Tem-se assim 40 atributos para cada proteína. Há nove possíveis localizações: “mitocôndria” (*mitochondrion*), “núcleo” (*nucleus*), “retículo endoplasmático” (*endoplasmic reticulum*), “extracelular” (*extracellular*), “citoplasma” (*cytoplasm*), “membrana plasmática” (*plasma membrane*), “golgi”, “peroxisomo” (*peroxisome*) e “vacúolo” (*vacuole*). Este conjunto também foi usado em (Lu et al., 2004), mas com uma codificação baseada em homologia na extração de atributos das proteínas. A maior parte dos exemplos nesse conjunto de dados pertencem à classe “núcleo” e a minoria encontra-se na classe “membrana plasmática”. A proporção de exemplos por classe é desbalanceada, variando de 1,4% a 34,6%.

---

³Os organismos eucariotos são aqueles que possuem o seu material genético em um núcleo, delimitado por uma membrana. Seres procariotos possuem o material genético difuso em suas células.

Conjunto	#Partições
<i>iris</i>	10
<i>car</i>	10
<i>glass</i>	5
<i>segment</i>	10
<i>lung</i>	3
<i>leukocyte</i>	10
<i>fungi</i>	10

Tabela 6.2: Números de partições de validação cruzada

6.2 Metodologia

Nesta seção são descritos os procedimentos adotados para a realização dos experimentos. São definidos uma série de critérios seguidos desde o pré-processamento dos conjuntos de dados até a maneira de se aplicar os algoritmos de aprendizado e comparar os seus resultados. A definição desse processo tem como objetivo uniformizar a apresentação de resultados nesta Tese, facilitando assim uma posterior reprodução dos experimentos apresentados.

6.2.1 Pré-Processamentos dos Dados

Foi verificado na Seção 6.1 que alguns conjuntos de dados possuem subconjuntos de treinamento T e teste S definidos, enquanto outros não apresentam essa divisão. Para o caso em que o conjunto encontra-se particionado, os subconjuntos definidos foram utilizados nos experimentos, como realizado em (Hsu & Lin, 2002; Rifkin & Klautau, 2004), por exemplo. Conjuntos sem esse tipo de separação foram divididos pelo método de validação cruzada *r-fold cross-validation* (CV) estratificado, descrito no Apêndice A. Por meio deste procedimento, gera-se r subconjuntos de treinamento T_i e de teste S_i . Os números de partições geradas no CV para cada conjunto de dados da Tabela 6.1 foram definidos a partir do número de exemplos nas classes, de forma a assegurar a presença de dados de todas as classes nos subconjuntos de teste. Os valores adotados encontram-se na Tabela 6.2.

Para avaliar os indivíduos gerados pelos AGs neste trabalho, por meio da função de avaliação da Equação 4.6, cada subconjunto de treinamento empregado nos experimentos foi ainda particionado segundo o método de *holdout*, também descrito no Apêndice A. A proporção de *holdout* adotada foi de 0,7, formando a partir de cada T um novo subconjunto de treinamento T' com 70% dos dados de T e um subconjunto de validação V com 30% dos dados de T . Novamente adotou-se um procedimento estratificado nessa divisão. As avaliações das aptidões dos classificadores manipulados pelos AGs utilizarão informações de desempenho nos subconjuntos de validação formados.

Deve-se salientar ainda que os mesmos subconjuntos de treinamento, validação e teste são utilizados em todos os experimentos com um determinado conjunto de dados, para

permitir comparações justas entre os diferentes algoritmos avaliados.

Na Figura 6.2, as divisões realizadas são ilustradas. Na Figura 6.2a tem-se um conjunto de dados com subconjuntos de treinamento e teste definidos, enquanto na Figura 6.2b tem-se uma divisão por CV. Na prática, o uso do CV implica em realizar o mesmo tipo de divisão ilustrado na Figura 6.2a r vezes.

Figura 6.2: Divisões dos conjuntos de dados para a realização dos experimentos

No caso das SVMs, todos os dados devem apresentar formato numérico. Com isso, os atributos nominais presentes no conjunto *car* devem ser codificados em números. Para tal, foi aplicada uma codificação canônica, em que se tem um bit representando cada valor que o atributo pode assumir. Um dos atributos desse conjunto, por exemplo, descreve o nível de segurança dos automóveis, que pode ser: “baixo”, “médio” e “alto”. Com o uso da codificação canônica, ele passa a ser representado por três bits. Um carro com nível de segurança alto possuirá a cadeia “0 0 1” neste atributo.

Além disso, é importante que alguns conjuntos de dados a serem utilizados com as SVMs sejam normalizados. Hsu et al. (2003) apontam que esse tipo de pré-processamento impede o domínio de atributos em intervalos numéricos maiores sobre aqueles em intervalos menores. Assim, os conjuntos de treinamento foram normalizados de forma que seus atributos possuíssem média nula e variância unitária. O pré-processamento dos subconjuntos de teste e de validação foi então realizado de acordo com os fatores de normalização extraídos dos dados de treinamento correspondentes. Este passo de normalização não foi aplicado ao conjunto de dados *fungi*, pois descaracterizaria as informações de ordem dos aminoácidos imposta pela codificação adotada para as proteínas. O conjunto de dados *car*, cujos atributos nominais foram codificados de maneira canônica, também não foi normalizado.

Com relação aos dados de expressão gênica para a classificação de tumores presentes no conjunto *lung*, uma questão importante refere-se à seleção de subconjuntos de genes para reduzir o número de atributos de entrada. Pode-se verificar na Tabela 6.1 que este conjunto de dados possui medidas de expressão de milhares de genes, representados na coluna “Atrib.”. Muitos genes, porém, são expressos de maneira similar em diferentes tecidos, se tornando irrelevantes para a discriminação dos tumores (Dettling & Bühlmann, 2002). Uma etapa de redução de dimensionalidade pode então ser aplicada a esses dados, permitindo com isso que se minimize o custo computacional envolvido na posterior geração

dos classificadores.

A técnica de seleção de genes implementada neste trabalho foi extraída de (Dudoit et al., 2000) e permite que esse procedimento seja aplicado ao problema multiclases geral. Para cada vetor de expressões de um gene ($\mathbf{e}_g$) em diferentes tecidos, define-se uma razão da soma dos quadrados das diferenças de expressão entre as classes (*between groups* - BSS) e dentro de cada classe (*within groups* - WSS), como:

$$\frac{\text{BSS}}{\text{WSS}}(\mathbf{e}_g) = \frac{\sum_{i=1}^n \sum_{j=1}^k I(y_i = j) (\mu_{\mathbf{e}_{g,j}} - \mu_{\mathbf{e}_g})^2}{\sum_{i=1}^n \sum_{j=1}^k I(y_i = j) (\mathbf{e}_{g,i} - \mu_{\mathbf{e}_{g,j}})^2} \quad (6.1)$$

em que $I(z) = 1$ se z é verdadeiro e $I(z) = 0$ se z é falso, n representa o número de dados no conjunto, k é o número de classes, y_i constitui a classe do exemplo i , $\mu_{\mathbf{e}_g}$ denota a expressão média de $\mathbf{e}_g$ em todo o conjunto de dados, $\mu_{\mathbf{e}_{g,j}}$ denota a média da expressão do gene sobre os exemplos pertencentes à classe j e $\mathbf{e}_{g,i}$ representa a i -ésima componente do vetor $\mathbf{e}_g$. Seleciona-se então os n_g genes com maior razão $\frac{\text{BSS}}{\text{WSS}}$. Empregou-se um valor de $n_g = 500$, a partir de resultados apresentados em (Statnikov et al., 2005).

Como no caso da normalização, a seleção de genes deve ser aplicada inicialmente nos subconjuntos de treinamento. Retira-se então dos subconjuntos de teste e de validação correspondentes os atributos considerados menos relevantes a partir dos dados de treinamento. Deve-se salientar que foi escolhida uma técnica relativamente simples para a seleção dos genes, uma vez que não é objetivo deste trabalho analisar a influência desse processo na previsão final.

6.2.2 Teste Estatístico

Um número considerável de técnicas multiclases são comparadas neste trabalho. Para permitir uma melhor avaliação das diferenças entre os seus resultados, testes estatísticos serão empregados nessas comparações. Optou-se por utilizar nessas análises um teste apresentado em (Feelders & Verkooijen, 1996) para a comparação de múltiplos algoritmos indutivos de AM.

Dados q classificadores f_i , deseja-se compará-los dois a dois, verificando se existe uma diferença significativa entre os seus resultados. O número total de comparações realizadas é então $J = \frac{q(q-1)}{2}$. Monta-se inicialmente a matriz apresentada na Tabela 6.3. Nesta matriz, Y_{ji} é igual a 0 se f_i classifica o exemplo j corretamente e é 1 caso contrário e n representa o número de dados de teste. A variância conjunta dos testes é calculada pela Equação 6.2.

$$\sigma_{dif}^2 = 2 \frac{\left(q \sum_{j=1}^n Y_{j\cdot} - \sum_{j=1}^n Y_{j\cdot}^2 \right)}{n^2 q (q-1)} \quad (6.2)$$

Pode-se então construir o intervalo de confiança $100(1 - \alpha)\%$ apresentado na Equação

Observações	Funções				Total
	f_1	f_2	$\dots$	f_q	
1	Y_{11}	Y_{12}	$\dots$	Y_{1q}	$Y_{1\cdot}$
2	Y_{21}	Y_{22}	$\dots$	Y_{2q}	$Y_{2\cdot}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	Y_{n1}	Y_{n2}	$\dots$	Y_{nq}	$Y_{n\cdot}$
Total	$Y_{\cdot 1}$	$Y_{\cdot 2}$	$\dots$	$Y_{\cdot q}$	
Médias	$\bar{Y}_{\cdot 1}$	$\bar{Y}_{\cdot 2}$	$\dots$	$\bar{Y}_{\cdot q}$	

Tabela 6.3: Matriz para a comparação múltipla de algoritmos

6.3 para todas as diferenças entre pares de classificadores f_i e f_m , em que $Z_{J:1-\alpha/2}^{n-1}$ representa a distribuição t de Student com $n - 1$ graus de liberdade ajustada para o número de comparações J (Feelders & Verkooijen, 1996; Dunn, 1961).

$$\left(\bar{Y}_{\cdot i} - \bar{Y}_{\cdot m} \pm Z_{J:1-\alpha/2}^{n-1} \sigma_{dif} \right) \quad (6.3)$$

Se o intervalo de confiança computado entre dois classificadores f_i e f_m contém o valor 0, não há evidência a um nível de significância α de que os resultados desses modelos diferem. Caso contrário, pode-se afirmar que os algoritmos possuem desempenho diferente com $100(1 - \alpha)\%$ de confiança.

No caso de um conjunto de dados dividido por CV em r partições de treinamento e teste, agrupou-se as previsões obtidas em todos os subconjuntos de teste S_i e montou-se matriz da Tabela 6.3 a partir de toda essa informação. Logo, na prática procedeu-se como se houvesse apenas um conjunto de teste, constituído da união de todos os r subconjuntos S_i .

6.2.3 Procedimento Geral

A partir das considerações previamente apresentadas, é possível definir o procedimento geral adotado na realização dos experimentos. Para cada conjunto de dados, pode-se definir os seguintes passos experimentais:

- **Codificação de atributos nominais:** necessário para as SVMs.
- **Divisão do conjunto de dados:** todos os conjuntos de dados devem possuir subconjuntos de treinamento T e de teste S para a aplicação dos algoritmos de AM. Se o conjunto não possui esses subconjuntos definidos, ele deve ser dividido segundo o método de validação cruzada *r-fold cross-validation* (CV) estratificado. Os subconjuntos de treinamento devem ainda ser divididos com *holdout* estratificado para a obtenção de subconjuntos de treinamento T' e de validação V . Os subconjuntos de validação serão utilizados na avaliação dos indivíduos gerados pelos AGs.
- **Seleção de atributos:** esta etapa foi aplicada sobre o conjunto de dados *lung*, que

possui medidas de expressão de milhares de genes, os quais constituem os atributos deste problema.

- **Normalização dos dados:** para as SVMs, deve-se normalizar os dados de treinamento, validação e teste. Como descrito na Seção 6.2.1, este passo não é necessário para os conjuntos de dados *fungi* e *car*.
- **Executar os algoritmos:** executar cada algoritmo utilizando o subconjunto de treinamento T e avaliar os seus resultados no subconjunto de teste S . No caso dos AGs, os classificadores contidos nos indivíduos devem ser treinados em T' e avaliados no subconjunto de validação V . Após obter uma solução, ela deve ser treinada no subconjunto de treinamento original T e testada em S , no qual os resultados finais são medidos e reportados. Se há mais de um subconjunto de treinamento e de teste (obtidos pelo CV), os procedimentos apontados devem ser realizados para cada partição individual i e o resultado final deve ser fornecido pela média dos desempenhos observados nos subconjuntos de teste S_i .
- **Comparar os algoritmos:** comparar os resultados obtidos pelos algoritmos executados com o teste estatístico descrito na Seção 6.2.2 com um nível de significância $\alpha = 0,05$.

Utilizando a metodologia previamente descrita em conjuntos divididos por CV, cada partição é tratada de forma independente pelos AGs. Logo, as soluções obtidas serão voltadas a cada partição individual, podendo ser diferentes.

A biblioteca LibSVM (Chang & Lin, 2004b) foi utilizada na indução das SVMs binárias. Todos os experimentos foram executados em um computador Pentium 4 HT de 3,4 GHz e com 2 GB de memória RAM.

Para facilitar uma comparação direta dos diferentes algoritmos, adotou-se os mesmos parâmetros em todas as SVMs induzidas. Empregou-se o Kernel Gaussiano, com $\sigma = 0,01$. O valor adotado para a constante de regularização foi $C = 100$. Estes constituem os parâmetros default do simulador SVMtorch II⁴ (Collobert & Bengio, 2001). Utilizou-se esses valores porque os parâmetros default do LibSVM ($C = 1$ e $\sigma = 1/\#\text{Atrib.}$) levaram a taxas de acertos equivalentes à TAM em alguns conjuntos de dados, não se mostrando assim adequados.

6.3 Obtenção de Matrizes de Códigos

A estratégia 1ct é geralmente a mais empregada na generalização das SVMs a problemas multiclases. Em um trabalho experimental rigoroso, Rifkin & Klautau (2004)

---

⁴O parâmetro default do Kernel Gaussiano no simulador SVMtorch II é igual a 10. Porém, nessa ferramenta, a equação deste Kernel é dada por $K(\mathbf{x}_i, \mathbf{x}_j) = -\|\mathbf{x}_i - \mathbf{x}_j\|^2 / (\sigma')^2$. Para uma equivalência com a equação da Tabela 2.1, adotada no LibSVM, deve-se utilizar um valor de σ igual a $1/(\sigma')^2$. Para $\sigma' = 10$, tem-se como resultado $\sigma = 0,01$.

demonstraram que, quando os parâmetros das SVMs binárias são ajustados de forma apropriada, a matriz de códigos 1ct alcança resultados estatisticamente similares aos de outras estratégias multiclases mais complexas. Neste trabalho, esse tipo de ajuste de parâmetros não foi realizado, pois implicaria em um volume de experimentos muito grande frente ao número de técnicas que são aqui comparadas. A decomposição 1ct é também a que utiliza o menor número de preditores binários entre as principais estratégias baseadas em matrizes de códigos da literatura.

Pelo fato da estratégia 1ct ser a mais utilizada na prática e de seus resultados serem bons para as SVMs, empregou-se os AGs descritos na Seção 4.2 na busca de matrizes de códigos com um número máximo de classificadores binários igual a k , correspondente ao número de preditores no 1ct. A matriz 1ct também foi fornecida como uma solução inicial para os AGs.

Os AGs lexicográfico e SPEA2 irão então buscar matrizes com baixo erro nos subconjuntos de validação segundo a Equação 4.6, procurando também minimizar o número de classificadores binários nas soluções e evitar a ocorrência de colunas equivalentes. Duas colunas são consideradas equivalentes se elas implicam na geração de preditores binários iguais. O número de classificadores binários será buscado no intervalo $[\lceil \log_2(k) \rceil, k]$, em que $\lceil \log_2(k) \rceil$ corresponde ao número mínimo necessário para distinguir k classes.

O objetivo principal deste estudo pode então ser resumido a determinar decomposições adequadas a cada problema multiclases mais simples que a 1ct, necessitando de menos preditores binários, e com taxas de acerto superiores ou similares. A função de decodificação utilizada na integração dos classificadores binários nas matrizes foi a baseada em margens, que em (Allwein et al., 2000) mostrou ser mais adequada às SVMs do que a função de *Hamming*.

6.3.1 Configuração dos Experimentos

Os parâmetros utilizados para os AGs foram:

- Taxa de cruzamento = 0,8;
- Taxa de seleção em torneio = 0,75;
- Taxa de mutação = 0,05;
- Tamanho de população = $5 * k$;
- Número máximo de ciclos = $50 * k$;
- Taxa de elitismo = $2/\text{tamanho da população}$;

O tamanho da população e o número máximo de ciclos foram definidos de acordo com cada conjunto de dados. Para problemas com mais classes, eles são maiores, uma vez que o espaço de busca aumenta com o crescimento do número de classes. Optou-se

então por ajustar esses parâmetros como múltiplos do número de classes k dos problemas investigados. No AG SPEA2, o tamanho de população foi ainda particionado em $2/3$ para a população geral manipulada pelo AG (P no Algoritmo 4.2) e $1/3$ para o arquivo de soluções não dominadas (A no Algoritmo 4.2). Empregou-se os mesmos parâmetros para as SVMs e os AGs em todos os conjuntos de dados para facilitar posteriores comparações.

Como os resultados dos AGs dependem da população inicial de soluções fornecida, eles foram executados diversas vezes para verificar a sua estabilidade. Dada uma partição de treinamento e teste de um conjunto de dados, as técnicas baseadas em AGs foram executadas trinta vezes, com populações iniciais diferentes. O número trinta foi selecionado por ser a quantidade mínima de dados em uma amostra com distribuição normal em estatística (Achcar & Rodrigues, 2000). Fornece-se então uma média dos resultados de todas as soluções finais obtidas. Nos conjuntos de dados divididos por CV, as trinta populações iniciais fornecidas para as execuções dos AGs nas diferentes partições dos dados foram as mesmas. Como para cada partição tem-se um conjunto de validação sobre os quais os AGs adaptam as soluções, as matrizes finais obtidas a partir de uma mesma população inicial geralmente são diferentes entre partições distintas dos dados.

Para realizar comparações estatísticas das soluções dos AGs com as das outras técnicas em um subconjunto de teste, escolheu-se uma das matrizes geradas. Optou-se por selecionar aquela com taxa de acerto no teste mais próxima à taxa média de todas as soluções. Escolhe-se assim a matriz que melhor reflete o comportamento médio das diversas soluções obtidas. Além disso, é esperado que em outras iterações do AG as novas matrizes produzidas possuam taxa de acerto próxima à média computada. Para conjuntos divididos por CV, são geradas trinta soluções para cada partição de treinamento e teste, e escolhe-se uma com desempenho próximo da média no teste. Tem-se assim matrizes de códigos distintas para cada partição do conjunto de dados.

Além da matriz 1ct, foram gerados ainda outros dois tipos de matrizes de códigos: tct e ECOC⁵. Os ECOCs foram obtidos pelo método exaustivo de Dietterich & Bariki (1995). Cada classe é representada por um código de tamanho $2^{k-1} - 1$. O código da primeira classe é composto somente por rótulos positivos. Para qualquer outra classe i , em que $i > 1$, ele é composto de 2^{k-i} seqüências alternadas de rótulos negativos e positivos. Esse ECOC possui boa capacidade de correção de erros. A distância de *Hamming* entre suas linhas é de 2^{k-2} . Com isso, ele é capaz de corrigir até $\lfloor 2^{k-3} - 0,5 \rfloor$ classificações incorretas dos $2^{k-1} - 1$ preditores binários que o constituem.

Para verificar o comportamento dos AGs codificados e a sua capacidade de encontrar soluções adequadas, também foram geradas matrizes aleatórias com números de classificadores no intervalo $[\lceil \log_2(k) \rceil, k]$. Como nos AGs, um total de trinta matrizes desse tipo foram geradas para cada conjunto de dados. Pode-se assim verificar se as soluções dos

---

⁵Diversos artigos referem-se a todas as estratégias baseadas em matrizes de códigos como ECOCs. Nesta Tese, porém, procurou-se denotar por ECOCs matrizes geradas com o objetivo de apresentar alguma propriedade de correção de erros.

AGs são mais adequadas que as obtidas com o uso de uma busca aleatória.

Na obtenção de cada uma das trinta matrizes aleatórias, empregou-se um procedimento similar ao adotado em (Allwein et al., 2000). Gera-se 10000 matrizes aleatórias, com elementos $\{-1, 0, +1\}$ e tamanhos variados. Caso alguma coluna das matrizes geradas não apresentasse ao menos um rótulo positivo e um rótulo negativo, aplicou-se um algoritmo reparador, que define aleatoriamente os rótulos positivos e/ou negativos ausentes. Escolhe-se então a matriz que apresenta a maior distância de *Hamming* mínima entre as suas linhas e sem classificadores binários idênticos, seguindo assim as diretrizes de Dietterich & Bariki (1995) para a obtenção de matrizes de códigos com boa propriedade de correção de erros. O número de colunas de cada matriz gerada foi determinado aleatoriamente dentro do intervalo estabelecido anteriormente, uma vez que os AGs também buscam esse número.

Como a obtenção de cada uma das trinta matrizes emprega algumas heurísticas de correção de erros, na realidade elas não são totalmente aleatórias. Porém, elas serão referenciadas por aleatórias nas considerações subseqüentes.

Das trinta matrizes aleatórias avaliadas em cada partição de treinamento/teste, seleciona-se para testes estatísticos aquela com taxa de acerto no teste mais próxima à média das taxas de acerto de todas as matrizes, como realizado para os AGs. Em conjuntos de dados divididos por CV, o uso desse procedimento pode implicar na escolha de matrizes diferentes para cada partição de teste.

6.3.2 Resultados

Nesta seção são apresentados os resultados dos experimentos envolvendo as estratégias baseadas em matrizes de códigos anteriormente descritas.

► Taxas de acerto

As taxas de acerto das estratégias mencionadas encontram-se na Tabela 6.4. Para os AGs, tem-se os resultados das versões lexicográfica e do SPEA2. O AG lexicográfico impõe uma ordem nos objetivos de minimização de erro no subconjunto de validação e do número de classificadores binários nas matrizes, priorizando o primeiro objetivo. Ele também penaliza matrizes que apresentem colunas equivalentes. O AG SPEA2 considera os objetivos de minimizar o erro de validação, o número de classificadores binários e a ocorrência de colunas equivalentes simultaneamente, por meio do uso do conceito de dominância de Pareto. Para os AGs e as matrizes aleatórias, são ilustradas as taxas de acerto médias de todas as soluções geradas em cada conjunto de dados. Seus desvios-padrão são indicados entre parênteses. No caso das estratégias 1ct, tct e ECOC sobre conjuntos de dados divididos por CV, tem-se também a média das taxas de acerto dessas matrizes em cada partição de teste. Seus desvios-padrão encontram-se entre parênteses.

Nas estratégias baseadas em matrizes de códigos, é possível que ocorram classificações desconhecidas, caso mais de uma linha da matriz de códigos apresente distância mínima em relação às previsões dos classificadores binários. Em diversos trabalhos, quando tal

Conjunto	1ct	tct	ECOC	Aleatório	AGs	
					Lexicográfico	SPEA2
<i>iris</i>	97,3 (4,7)	96,0 (3,4)	97,3 (4,7)	91,6 (11,7)	95,0 (3,6)	95,0 (4,0)
<i>car</i>	94,0 (2,6)	98,1 (1,3)	93,8 (2,8)	85,1 (20,1)	97,8 (1,4)	94,1 (2,5)
<i>lung</i>	82,8 (3,0)	76,4 (1,3)	82,8 (3,0)	65,9 (22,1)	85,5 (4,8)	84,0 (5,1)
<i>glass</i>	70,1 (7,7)	71,0 (5,8)	69,6 (7,5)	51,2 (14,8)	68,3 (7,2)	66,5 (6,4)
<i>segment</i>	95,4 (1,5)	96,6 (1,0)	94,9 (1,2)	67,8 (16,1)	96,2 (1,1)	88,1 (7,5)
<i>leukocyte</i>	90,0 (2,8)	90,9 (2,6)	91,2 (2,1)	62,8 (14,0)	89,2 (2,9)	78,9 (7,5)
<i>fungi</i>	56,1 (2,1)	60,4 (2,4)	60,8 (2,4)	39,8 (11,7)	62,0 (2,6)	59,4 (3,3)
<i>structural</i>	80,3	81,0	80,8	73,0 (9,6)	80,1 (0,9)	75,4 (3,9)
<i>sat-image</i>	90,5	90,1	90,1	70,4 (11,6)	90,0 (0,4)	82,8 (6,0)
<i>opt-digits</i>	97,3	96,3	97,6	72,0 (16,7)	96,7 (0,5)	84,9 (16,4)
<i>pen-digits</i>	97,9	97,3	97,8	69,5 (20,8)	97,4 (0,3)	81,4 (18,7)
<i>vowel</i>	51,7	57,6	54,6	35,6 (8,9)	50,3 (3,2)	41,6 (5,2)

Tabela 6.4: Taxas de acerto das estratégias baseadas em matrizes de códigos

empate ocorre, seleciona-se, aleatoriamente, uma das classes envolvidas. Nesta Tese, a ocorrência desse tipo de empate foi discriminada. Nos testes estatísticos realizados, empates foram considerados erros.

De todas as matrizes investigadas, apenas as aleatórias e as determinadas pelos AGs apresentaram esse tipo de classificação. Esta é, então, uma deficiência das matrizes geradas por esses três métodos, pois indica que os códigos atribuídos às classes acarretaram em empates em alguns casos, não diferenciando cada classe unicamente. Porém, a ocorrência desse tipo de classificação também pode ser minimizada com o uso de uma função de decodificação distinta na integração dos preditores binários, como a baseada em probabilidades de Passerini et al. (2004).

O AG lexicográfico apresentou as menores taxas de exemplos desconhecidos, as quais podem ser consideradas baixas. Elas ocorreram em apenas parte dos conjuntos de dados. Em seguida, tem-se as matrizes do SPEA2, com um aumento na incidência desse tipo de classificação. As matrizes aleatórias apresentaram as maiores taxas de exemplos desconhecidos. Essas taxas podem ser consultadas na Tabela B.13 do Apêndice B.

É importante observar a estabilidade das soluções determinadas pelos AGs e pelo procedimento aleatório. As taxas de acerto e também de exemplos desconhecidos do AG lexicográfico foram as mais estáveis, apresentando menores desvios-padrão. Para os conjuntos de dados divididos por CV, os valores de desvio-padrão das taxas de acerto de todas as soluções obtidas por este AG se encontram próximos aos das estratégias 1ct, tct e ECOC. Logo, as variações verificadas em grande parte são efeito do uso do CV nesses conjuntos, ou seja, das taxas de acerto terem sido calculadas em partições distintas dos dados. Nos conjuntos de dados em que a CV não foi utilizada, verifica-se maiores variações de resultados no conjunto de dados *vowel*. Este conjunto de dados representa uma tarefa de classificação difícil, considerando as baixas taxas de acerto obtidas por todas as matrizes testadas. Essa maior variação também pode ser decorrente do fato de seu

número de classes ser maior e da presença de poucos dados de treinamento. Com isso, os subconjuntos de treinamento e validação sobre os quais os AGs avaliam suas soluções tornam-se pouco representativos. Ainda assim, os valores de desvio-padrão das taxas de acerto das matrizes determinadas pelo AG lexicográfico foram mais baixos que os do método aleatório e do SPEA2.

As soluções determinadas aleatoriamente tiveram taxas de acerto e de exemplos desconhecidos bastante instáveis. As matrizes do AG SPEA2 também apresentaram taxas instáveis em alguns conjuntos de dados, sendo essa uma deficiência deste AG em relação à versão lexicográfica.

► Testes Estatísticos

Para verificar mais claramente as diferenças entre os resultados reportados na Tabela 6.4, comparou-se estatisticamente as taxas de acerto de todas as técnicas para cada conjunto de dados. Exemplos desconhecidos foram considerados erros nos testes realizados. É importante observar que, com isso, o desempenho dos algoritmos que apresentam esse tipo de classificação são um pouco subestimados, pois o usual é selecionar uma das classes para as quais houve empates aleatoriamente.

Como mencionado anteriormente, no caso das matrizes aleatórias e obtidas pelos AGs, seleciona-se para os testes aquelas com taxas de acerto mais próximas à média obtida em cada partição de teste. Quando mais de uma solução encontra-se próxima a essa média, uma delas é selecionada aleatoriamente. A escolha realizada não reflete os valores de desvio-padrão do desempenho das diferentes matrizes geradas por essas abordagens. Com isso, o AG SPEA2 e o procedimento aleatório, cujas soluções apresentaram maiores taxas de desvio-padrão, são ligeiramente favorecidos.

Na Figura 6.3 são apresentados dois quadros que resumem os resultados dos testes realizados. Na Tabela B.14 do Apêndice B tem-se mais informações sobre esses testes. Na Figura 6.4 as taxas de acerto das soluções comparadas são plotadas graficamente. As faixas horizontais em tom escuro conectam as barras das estratégias que possuem taxas de acerto estatisticamente similares. Para os AGs e o método aleatório, as taxas apresentadas correspondem às das soluções médias comparadas estatisticamente.

No quadro da Figura 6.3a são indicadas as estratégias que apresentaram as maiores e as menores taxas de acerto em cada conjunto de dados. Para um determinado conjunto, no bloco de colunas “Melhores taxas”, as células correspondentes à estratégia de maior taxa de acerto e às estratégias com desempenho estatisticamente semelhante à melhor, encontram-se preenchidas em cinza. Essas são as estratégias mais adequadas para aquele conjunto, segundo suas taxas de acerto. No bloco “Piores taxas”, apresenta-se a mesma informação, mas para as técnicas que apresentaram pior desempenho. Assim, estão preenchidas em cinza as células associadas à estratégia de menor taxa de acerto e àquelas com resultados estatisticamente similares a essa. Na última linha desse quadro tem-se então a porcentagem de vezes em que as técnicas de cada coluna figuram entre as

Conjunto	Melhores taxas						Piores taxas					
	1ct	tct	eco	ale	lex	spe	1ct	tct	eco	ale	lex	spe
<i>iris</i>												
<i>car</i>												
<i>lung</i>												
<i>glass</i>												
<i>segment</i>												
<i>leukocyte</i>												
<i>fungi</i>												
<i>structural</i>												
<i>sat-image</i>												
<i>opt-digits</i>												
<i>pen-digits</i>												
<i>vowel</i>												
%	83	92	92	8	92	25	0	8	0	92	0	17

(a)

	# Dif.	Melhor	%
1ct-tct	2	tct	100
1ct-eco	1	eco	100
1ct-ale	11	1ct	100
1ct-lex	2	lex	100
1ct-spe	8	1ct	88
tct-eco	1	tct	100
tct-ale	10	tct	100
tct-lex	2	tct,lex	50
tct-spe	8	tct	100
eco-ale	11	eco	100
eco-lex	1	lex	100
eco-spe	7	eco	100
ale-lex	11	lex	100
ale-spe	9	spe	100
lex-spe	9	lex	100
		tct,lex	33

(b)

Figura 6.3: Quadros resumo dos testes entre as estratégias baseadas em matrizes de códigos

de melhores e as de piores taxas de acerto nos testes realizados.

No quadro da Figura 6.3b são indicados os números de diferenças (“#Dif.”) estatísticas encontradas entre cada par de matrizes comparadas. Na coluna “Melhor” são indicadas as estratégias que se sobressaíram nas diferenças detectadas, apresentando maiores taxas de acerto. A coluna “%” indica a proporção de vezes em que a matriz em “Melhor” se sobressaiu nas diferenças reportadas. Logo, na comparação “1ct-spe”, por exemplo, em oito conjuntos de dados houve diferenças estatísticas de resultados e em sete deles (88%) a matriz 1ct apresentou maiores taxas de acerto que a matriz gerada pelo SPEA2. A última linha do quadro indica as estratégias que figuraram mais vezes na coluna “Melhor” e a proporção de vezes que isso ocorreu. Algumas linhas da tabela encontram-se destacadas para futuras referências.

No quadro da Figura 6.3a, verifica-se que as matrizes do ECOG e do AG lexicográfico apresentaram bom desempenho na maioria dos conjuntos de dados. Elas estiveram o maior número de vezes entre as matrizes com melhores taxas de acerto e o menor número entre as com piores taxas. Em seguida, tem-se as matrizes tct e 1ct. Embora a matriz tct tenha figurado mais vezes entre as estratégias de melhores taxas de acerto, ela apresentou um caso de baixo desempenho, o que não ocorreu com a matriz 1ct. As matrizes do SPEA2 apresentaram taxas de acerto estatisticamente similares às das estratégias de melhor desempenho somente em três conjuntos de dados. Também destaca-se o fato de que, em oito dos doze conjuntos de dados empregados, as matrizes 1ct, tct, ECOG e do AG lexicográfico demonstraram taxas de acerto similares. No quadro da Figura 6.3b esta observação pode ser confirmada, uma vez que foram encontradas poucas diferenças nos testes realizados entre esses pares de técnicas.

As matrizes do AG lexicográfico não se destacaram apenas no conjunto de dados *vowel*.

Figura 6.4: Comparações das taxas de acerto das estratégias baseadas em matrizes de códigos

Nesse caso, houve uma diferença de resultado entre a matriz média desse AG e a matriz da estratégia tct, que apresentou uma maior taxa de acerto.

A matriz aleatória em geral apresentou resultados inferiores aos das outras técnicas na maioria dos conjuntos de dados avaliados. Há alguns casos, porém, em que as matrizes tct e as matrizes determinadas pelo SPEA2 apresentaram taxas de acerto similares às das matrizes aleatórias. Logo, empregar o AG SPEA2 nesses casos levou a soluções com taxas de acerto similares às de matrizes obtidas pelo procedimento mais simples de gerar múltiplas matrizes aleatórias de tamanhos variados e selecionar aquela com maior distância de *Hamming* mínima entre linhas e sem colunas equivalentes.

A matriz tct, por sua vez, teve o desempenho deteriorado no conjunto de dados *lung*. É reportado em (Passerini et al., 2004) que o uso da decodificação de *Hamming* implica na realização de uma votação por maioria entre as previsões dos preditores binários, método

de integração que produz bons resultados para classificadores tct (Friedman, 1996). O mesmo não ocorre, por exemplo, para a matriz 1ct, em que o uso desse tipo de votação ocasiona em muitos empates, prejudicando significativamente a taxa de acerto desta estratégia. Embora o uso da decodificação de *Hamming* no conjunto de dados *lung* eleve a taxa de acerto da matriz tct de 76,4 para 77,3, ela ainda se mostra baixa em relação às das outras estratégias. O desempenho inferior do tct neste caso está mais relacionado às características desse conjunto de dados, conforme será exposto na Seção 6.7.

É interessante destacar que, no conjunto de dados *iris*, todas as técnicas testadas, incluindo as matrizes de desempenho médio determinadas aleatoriamente, tiveram taxas de acerto estatisticamente similares. Pode-se atribuir esse comportamento ao fato deste ser um conjunto de dados simples, para o qual vários tipos de decomposições são capazes de obter bons resultados. Além disso, para três classes, o espaço de busca de possíveis matrizes de códigos é bastante reduzido. Com isso, o método aleatório é capaz de encontrar soluções boas na média, embora o seu desvio-padrão seja alto.

As soluções determinadas pelo AG SPEA2, embora tenham apresentado boas taxas de acerto em alguns conjuntos de dados, como o *fungi* e o *lung*, em geral tiveram o desempenho deteriorado. Logo, a consideração dos objetivos de minimização do erro de validação, do número de preditores binários nas soluções e da quantidade de classificadores iguais simultaneamente em geral não se mostrou uma alternativa adequada à obtenção de soluções com boas taxas de acerto nos subconjuntos de teste.

Por outro lado, o AG lexicográfico, que impõe uma ordem nos objetivos, foi capaz de gerar matrizes cujas taxas de acerto encontram-se entre as maiores. Em todos os conjuntos de dados as taxas de acerto das matrizes médias do AG lexicográfico foram superiores às do SPEA2, sendo que em dez deles as diferenças de resultados foram estatisticamente significantes, como pode ser observado na última linha do quadro da Figura 6.3b, destacada em cinza escuro. Embora existam técnicas para impor preferências nos objetivos de AGs baseados na relação de Pareto (Deb, 1999), como o SPEA2, sua utilização pode levar esse algoritmo a perder sua característica mais atrativa em relação à versão lexicográfica, a de considerar todos os objetivos com a mesma ordem de importância.

Na comparação das taxas de acerto das soluções médias determinadas pelos AGs com as da estratégia 1ct, verifica-se que em geral o AG lexicográfico foi capaz de mantê-las similares às da matriz 1ct. Em dois conjuntos de dados em que houve diferenças estatísticas entre as matrizes do AG lexicográfico e a 1ct (*car* e *fungi*), as taxas de acerto do AG lexicográfico se sobressaíram, conforme destacado na linha em cinza claro do quadro da Figura 6.3b. As soluções determinadas pelo AG SPEA2, em contrapartida, em vários conjuntos de dados não foram capazes de manter taxas de acerto similares às da matriz 1ct, como também pode ser observado no quadro da Figura 6.3b, na linha em cinza intermediário.

► Número de Classificadores Binários

Como apontado no início desta seção, outro objetivo dos AGs codificados é obter de composições multiclasss que necessitem de menos classificadores binários. Na Tabela 6.5 são apresentados os números de classificadores binários das matrizes geradas neste estudo. Todas as matrizes aleatórias e determinadas pelo AG lexicográfico não apresentaram classificadores binários idênticos. O mesmo não ocorreu com o AG SPEA2, que apresentou algumas soluções com preditores iguais. Ele não foi, portanto, capaz de lidar com a restrição de encontrar matrizes sem colunas equivalentes.

Conjunto	1ct	tct	ECOC	Aleatório	AGs	
					Lexicográfico	SPEA2
<i>iris</i>	3	3	3	2,6 (0,5)	2,0 (0,1)	2,0 (0,1)
<i>car</i>	4	6	7	3,2 (0,7)	3,6 (0,5)	2,1 (0,3)
<i>lung</i>	5	10	15	4,1 (0,8)	3,6 (0,7)	3,0 (0,2)
<i>glass</i>	6	15	31	4,8 (1,0)	4,6 (0,8)	3,0 (0,1)
<i>segment</i>	7	21	63	4,9 (1,4)	6,5 (0,7)	3,1 (0,3)
<i>leukocyte</i>	8	127	30	5,3 (1,6)	7,8 (0,4)	3,1 (0,3)
<i>fungi</i>	9	36	255	6,9 (1,9)	7,7 (1,2)	4,1 (0,2)
<i>structural</i>	4	6	7	3,2 (0,7)	3,8 (0,4)	2,1 (0,4)
<i>sat-image</i>	6	15	31	4,8 (1,1)	6,0 (0,2)	3,3 (0,5)
<i>opt-digits</i>	10	45	511	6,4 (2,2)	9,8 (0,6)	4,2 (0,5)
<i>pen-digits</i>	10	45	511	6,4 (2,2)	9,9 (0,3)	4,1 (0,3)
<i>vowel</i>	11	55	1023	7,5 (2,6)	10,4 (1,1)	4,1 (0,3)

Tabela 6.5: Números de classificadores das estratégias baseadas em matrizes de códigos

Pela Tabela 6.5, pode-se observar que o AG SPEA2 realiza uma minimização grande do número de classificadores binários nas matrizes. A maioria das soluções determinadas por este AG apresentaram números de preditores próximos ao mínimo necessário para diferenciar as classes dos problemas. Esse fato indica que este AG tendeu a dar ênfase ao objetivo de minimizar o número de classificadores binários nas soluções, embora esse tipo de preferência não tenha sido diretamente incorporado ao algoritmo. A maneira de escolher uma das soluções geradas por este AG em cada iteração também pode ter exercido influência nesse resultado. Seleciona-se uma das soluções no arquivo de indivíduos não dominados que apresente menor distância a um ponto de referência, o qual contém os valores mínimos dos objetivos em suas coordenadas. Um exame geral dos indivíduos contidos nas populações finais deste algoritmo revela que a maioria deles apresenta números reduzidos de classificadores binários, concordando assim com a primeira hipótese levantada. De fato, no cálculo da aptidão dos indivíduos pelo SPEA2, soluções que apresentam o número mínimo de classificadores não são dominadas por outras. Com isso, elas tendem a apresentar melhores aptidões e assim se multiplicam no decorrer do processamento deste AG.

O AG lexicográfico, por sua vez, foi capaz de reduzir os números de classificadores binários em vários casos. Entretanto, em alguns conjuntos de dados, os valores se man-

tiveram próximos ao do 1ct.

Na Tabela 6.6 são apresentadas as porcentagens médias de redução do número de classificadores nas matrizes determinadas pelos AGs lexicográfico e SPEA2 em relação à estratégia 1ct. As técnicas e os conjuntos de dados nessa tabela são denotados por suas três letras iniciais. Na última coluna são indicadas as médias das proporções verificadas em cada AG. As médias reportadas confirmam as observações anteriores de que o AG SPEA2 opera maiores minimizações na quantidade de classificadores binários.

	iri	car	lun	gla	seg	leu	fun	str	sat	opt	pen	vow	média
lex	33,3	10,0	28,0	23,3	7,1	2,5	14,4	5,0	0,0	2,0	1,0	5,5	11,0
spe	33,3	47,5	40,0	50,0	55,7	61,3	54,4	47,5	45,0	58,0	59,0	62,7	51,2

Tabela 6.6: Redução do número de classificadores das matrizes dos AGs em relação à 1ct

Examinando as soluções do AG lexicográfico em que o número de classificadores binários foi pouco reduzido, verificou-se que as matrizes obtidas foram similares à 1ct. Em alguns conjuntos de dados, como o *structural* e o *sat-image*, algumas das matrizes produzidas foram exatamente a 1ct. Este fato concorda com a afirmação de que a decomposição 1ct é adequada à solução de problemas multiclases com SVMs. Além disso, ele indica que a redução do número de classificadores binários pode não ser adequada para esses conjuntos de dados.

É interessante também observar que, nos conjuntos de dados *car*, *fungi* e *lung*, em que a matriz do AG lexicográfico apresentou as suas melhores taxas de acerto, houve reduções grandes no número de classificadores binários. Esse é um resultado importante, uma vez que ilustra que, por meio do uso de um procedimento de busca que adapta as matrizes de códigos à solução dos problemas multiclases, foi possível obter maiores taxas de acerto com decomposições mais simples.

Entre as matrizes 1ct, tct e ECOC, aquela que requer o menor número de classificadores binários é a 1ct, com k preditores para k classes. A matriz tct, com $\frac{k(k-1)}{2}$ preditores, figura em seguida, com a ECOC apresentando a maior quantidade de classificadores ($2^{k-1} - 1$). As soluções aleatórias apresentaram números de preditores binários superiores aos das matrizes dos AGs ou intermediários entre os obtidos pelos dois tipos de AGs. Entretanto, como esperado, elas também apresentaram maiores variações nesses números.

Como conclusão, tem-se que o AG lexicográfico foi capaz de obter matrizes que cumpriram os objetivos de apresentar boas taxas de acerto com o uso de menos classificadores binários que a decomposição 1ct, sem utilizar preditores idênticos. Na comparação dessas matrizes com outras determinadas por estratégias que empregam números maiores de classificadores, a tct e a ECOC, também verificou-se em geral a manutenção das taxas de acerto, com o uso de menos classificadores binários.

► Outros resultados

O tempo dispendido pelos AGs e pelo método aleatório, em segundos, na determinação das soluções reportadas é ilustrado na Tabela 6.7. Os tempos apresentados correspondem às médias dos tempos tomados para a obtenção de cada uma das trinta soluções geradas.

Conjunto	Aleatório	AGs	
		Lexicográfico	SPEA2
<i>iris</i>	0,0 (0,0)	0,4 (0,0)	0,5 (0,0)
<i>car</i>	0,0 (0,0)	9,0 (0,2)	7,8 (0,3)
<i>lung</i>	0,0 (0,0)	25,2 (0,2)	20,3 (1,1)
<i>glass</i>	0,1 (0,0)	3,1 (0,2)	3,1 (0,2)
<i>segment</i>	0,1 (0,0)	68,7 (3,5)	58,5 (6,8)
<i>leukocyte</i>	0,1 (0,0)	71,9 (3,1)	64,5 (7,7)
<i>fungi</i>	0,1 (0,0)	402,5 (32,4)	328,7 (62,5)
<i>structural</i>	0,0 (0,0)	11,9 (0,2)	9,1 (0,4)
<i>sat-image</i>	0,1 (0,0)	204,2 (8,7)	188,1 (22,5)
<i>opt-digits</i>	0,2 (0,1)	568,3 (33,4)	520,7 (101,9)
<i>pen-digits</i>	0,2 (0,1)	573,0 (24,2)	515,3 (91,7)
<i>vowel</i>	0,2 (0,1)	57,1 (3,2)	47,7 (6,8)

Tabela 6.7: Tempos de processamento de AGs e do método aleatório, em segundos

Entre os três métodos, o aleatório, que emprega um procedimento mais simples, foi o mais rápido. O AG lexicográfico foi mais lento que o SPEA2. Isso pode ser devido ao fato das matrizes nas populações do SPEA2 tenderem a apresentar números menores de classificadores binários com o decorrer das gerações. Com isso, menos classificadores são treinados e o tempo total do AG é reduzido.

Em geral, os tempos de execução para a determinação de uma solução pelos AGs variou de acordo com o tamanho do problema solucionado. Para aplicações com mais dados e mais classes, eles foram maiores. Verificou-se que o uso de uma lista com classificadores binários já treinados e avaliados foi responsável por acelerar consideravelmente o tempo de processamento dos AGs. O tamanho máximo atribuído para essa lista foi de 1000 vezes o tamanho da população, sendo então também ajustado de acordo com o número de classes dos problemas. Para a lista de matrizes avaliadas, utilizou-se um tamanho máximo igual ao da população. O uso dessa lista não foi, porém, determinante na redução do tempo de execução dos AGs.

Como os operadores genéticos dos AGs foram aplicados probabilisticamente de acordo com o seu desempenho no decorrer de suas gerações, verificou-se quais operadores predominaram na produção das soluções finais desses algoritmos. Para o AG lexicográfico geralmente o operador de cruzamento mais aplicado foi o de trocas de colunas, enquanto para o SPEA2 foi o de trocas de partes de matrizes. No caso da mutação, houve uma predominância do primeiro tipo, que altera aleatoriamente o valor de um elemento de uma dada matriz. Estes foram então os operadores que mais contribuíram na produção das soluções dos AGs considerados. Como o AG lexicográfico apresentou melhores resultados

finais, pode-se concluir que o operador de cruzamento de trocas de colunas, que representa a troca de classificadores binários entre matrizes distintas, é mais adequado ao problema de busca de matrizes de códigos.

Executou-se também os AGs lexicográfico e SPEA2 com um tamanho de população de $10 * k$ e um número máximo de ciclos igual a $100 * k$. Embora algumas melhoras de resultados tenham sido verificadas em alguns conjuntos de dados, elas foram em geral pequenas. O maior destaque foi obtido no conjunto de dados *vowel*, em que o AG lexicográfico determinou soluções com taxa média de acerto igual a 52,6 e com desvio-padrão 2,6, sendo assim superior à reportada anteriormente e estatisticamente similar à da matriz *tct*. Entretanto, o aumento nos valores desses parâmetros dos AGs também aumentou consideravelmente o tempo de processamento desses algoritmos (de 3 a 4 vezes). Por esse motivo, optou-se por apresentar os resultados com os parâmetros listados no início desta seção.

6.3.3 Resumo

Nesta seção os AGs para a determinação de matrizes de códigos lexicográfico e SPEA2 foram aplicados na procura de matrizes de códigos que apresentassem desempenho similar ou superior à matriz *1ct*, utilizando menos classificadores binários. Isso representa a busca de decomposições mais simples para solucionar os problemas multiclases. A matriz *1ct* foi escolhida como referência por ser a mais empregada na prática e por seus resultados serem bons na generalização das SVMs a problemas multiclases (Rifkin & Klautau, 2004).

Os AGs buscaram então matrizes que minimizassem a taxa de erro em subconjuntos de validação e reduzissem o número de preditores binários empregados nas soluções. As matrizes geradas também deviam obedecer à restrição de não apresentar colunas equivalentes, pois dessa forma as decomposições fariam uso de classificadores binários iguais.

O AG lexicográfico lida com os objetivos formulados impondo uma ordem aos mesmos. Privilegia-se a minimização do erro das matrizes nos subconjuntos de validação, seguida da redução do número de classificadores binários. A produção de matrizes com colunas equivalentes foi evitada com o uso de um procedimento de penalização dessas soluções. O AG SPEA2 minimiza o erro na validação, o número de classificadores e a presença de classificadores idênticos simultaneamente, por meio do uso do conceito de dominância de Pareto.

Os resultados experimentais evidenciaram que o AG lexicográfico apresentou melhor desempenho que o SPEA2. Ele foi capaz de determinar soluções com taxas de acerto similares e em alguns casos superiores às da matriz *1ct* com o uso de menos preditores binários. Porém, em alguns conjuntos de dados, as matrizes obtidas foram semelhantes ou iguais à *1ct*. A redução do número de classificadores binários nesses conjuntos pode então não ser adequada. Também tem-se uma confirmação do bom desempenho da matriz *1ct* para as SVMs. As taxas médias de acerto das matrizes do AG lexicográfico em geral se compararam ainda às de matrizes de códigos que empregam mais classificadores binários

que a 1ct, a tct e a ECOC.

O AG SPEA2, em contrapartida, embora tenha minimizado consideravelmente o número de classificadores binários das soluções, em geral não foi capaz de manter taxas de acerto comparáveis às da matriz 1ct e também não conseguiu lidar completamente com a restrição de determinar matrizes sem colunas equivalentes. Porém, deve-se destacar que este algoritmo foi aplicado em sua forma básica. A realização de alterações, como a incorporação de preferências nas soluções e o uso de um procedimento de penalização para as matrizes com colunas equivalentes, podem vir a melhorar seus resultados.

É importante salientar ainda que os AGs codificados não têm como objetivo produzir ECOCs, uma vez que não se procura matrizes que apresentem boas propriedades de correção de erros. Sua característica é adaptar as matrizes de códigos à solução de cada aplicação individual, utilizando informações de seu desempenho.

As matrizes determinadas são também adaptadas às SVMs, pois essa foi a técnica de AM empregada na solução dos subproblemas binários contidos nas mesmas. Logo, o uso desses AGs em conjunto com outro algoritmo de AM pode levar a resultados distintos dos apresentados neste estudo.

6.4 Determinação da Estrutura de DDAGs e ADAGs

Alguns trabalhos apontam que a forma como os classificadores binários são organizados nos nós de DDAGs e ADAGs afeta o seu desempenho na solução multiclases (Kijssirikul & Ussivakul, 2002; Takahashi & Abe, 2003; Feng et al., 2005). Nesta seção avalia-se a influência dessa organização na taxa de acerto desses dois tipos de DAGs quando SVMs são utilizadas na indução dos preditores binários.

Empregou-se os AGs da Seção 4.3 para determinar essa ordem, de acordo com o desempenho das diferentes estruturas de DAGs em subconjuntos de validação. Dessa forma, os DAGs foram adaptados à solução de cada aplicação particular com SVMs. Para os ADAGs, na prática determina-se os pareamentos das classes em seu primeiro nível.

6.4.1 Configuração dos Experimentos

Os AGs foram executados para determinar a estrutura de DDAGs e ADAGs sobre os conjuntos de dados com mais de sete classes, que possuem assim um espaço de busca maior. Conforme mencionado no Capítulo 3, há $\frac{k!}{2}$ DDAGs para um problema com k classes e $\frac{k!}{2^{\lfloor k/2 \rfloor}}$ ADAGs. Os parâmetros dos AGs empregados, de forma similar ao realizado na seção anterior, foram:

- Taxa de cruzamento = 0,8;
- Taxa de seleção em torneio = 0,75;
- Taxa de mutação = 0,05;
- Tamanho de população = $5 * k$;

- Número máximo de ciclos = $10 * k$;
- Taxa de elitismo = $2/\text{tamanho da população}$;

A cada mutação realizada, o operador de inserção foi aplicado k vezes sobre o indivíduo e escolheu-se então aquele de melhor aptidão, em um procedimento simplificado de busca local. Também testou-se a proporção de $10 * k$ para a população e de $50 * k$ e $100 * k$ para o número máximo de ciclos, porém os resultados foram similares a empregar os parâmetros previamente listados, a um custo computacional maior.

Como os resultados dos AGs dependem de sua população inicial, eles foram executados com soluções iniciais aleatórias um total de trinta vezes para cada partição de treinamento/teste de um conjunto de dados, como na seção anterior. Para testes estatísticos, selecionou-se a solução com taxa de acerto média no subconjunto de teste entre as geradas. Como nos AGs da Seção 6.3, em conjuntos de dados divididos por CV, tem-se soluções voltadas a cada partição de teste individual.

Foram criadas ainda estruturas aleatórias, para observar o desempenho dos AGs em relação a um procedimento de busca aleatório. Foram geradas trinta dessas estruturas para cada conjunto de dados. Cada solução aleatória foi determinada como uma lista representando uma possível permutação das classes. Procurou-se sempre gerar trinta permutações distintas. Embora esse procedimento não garanta que os DAGs obtidos serão diferentes, pois classificadores binários para os pares (i, j) e (j, i) são equivalentes, ele pode minimizar a produção de estruturas idênticas. Como evita-se a geração de estruturas iguais, o método adotado na realidade não é completamente aleatório. Entretanto, ele será referenciado como aleatório nas considerações subseqüentes.

6.4.2 Resultados

Nesta seção são apresentados os resultados da aplicação dos AGs e do método aleatório na determinação da ordem dos preditores binários em DDAGs e ADAGs.

► Taxas de acerto

Na Tabela 6.8 são apresentadas as taxas de acerto das soluções determinadas pelos AGs e aleatoriamente. Os desvios-padrão obtidos são indicados entre parênteses. Para o ADAG, tem-se o resultado de dois tipos de AGs. No primeiro, referenciado por AG₁, os dois pontos de corte nos operadores de cruzamento PMX e PX (Seção 4.1.2), são escolhidos de forma aleatória. No AG₂, um ponto de corte é selecionado aleatoriamente e o outro é determinado pela posição da classe que pareia com a do primeiro ponto no ADAG, conforme descrito na Seção 4.3.3.

Na Tabela 6.8 também encontram-se ilustradas as taxas de acerto médias do método de busca aleatório nos conjuntos de dados com menos de oito classes. Para o conjunto *iris*, que tem três classes, foram geradas as seis possíveis permutações e para os conjuntos *car* e *structural*, com quatro classes, produziu-se as 24 possíveis permutações.

Conjunto	DDAG		ADAG		
	Aleatório	AG	Aleatório	AG ₁	AG ₂
<i>iris</i>	96,7 (3,5)	-	96,7 (3,5)	-	-
<i>car</i>	97,8 (1,4)	-	97,8 (1,4)	-	-
<i>lung</i>	77,3 (0,6)	-	77,3 (0,6)	-	-
<i>glass</i>	70,4 (4,2)	-	70,3 (4,3)	-	-
<i>segment</i>	96,7 (0,9)	-	96,7 (0,9)	-	-
<i>leukocyte</i>	90,1 (2,3)	90,0 (2,3)	90,1 (2,3)	90,1 (2,1)	90,1 (2,1)
<i>fungi</i>	60,9 (2,4)	60,8 (2,2)	60,9 (2,4)	60,8 (2,2)	60,8 (2,2)
<i>structural</i>	81,1 (0,1)	-	81,1 (0,1)	-	-
<i>sat-image</i>	89,6 (0,1)	-	89,7 (0,1)	-	-
<i>opt-digits</i>	97,1 (0,1)	97,1 (0,1)	97,1 (0,1)	97,1 (0,1)	97,1 (0,1)
<i>pen-digits</i>	97,7 (0,1)	97,7 (0,1)	97,7 (0,1)	97,7 (0,0)	97,7 (0,1)
<i>vowel</i>	55,1 (0,8)	55,2 (0,6)	55,0 (0,7)	54,8 (0,5)	54,8 (0,6)

Tabela 6.8: Taxas de acerto de DDAGs e ADAGs

As soluções médias da Tabela 6.8 foram comparadas estatisticamente e nenhuma diferença foi detectada a 95% de confiança. De fato, é possível observar que todas as taxas de acerto obtidas em cada conjunto de dados foram bastante similares. Inclusive, o desempenho dos DDAGs e ADAGs foi semelhante em todos os conjuntos de dados. Isso ocorreu até mesmo para números maiores de classes, em que, segundo Kijisirikul & Ussivakul (2002), os ADAGs apresentam a vantagem de reduzir a ocorrência de erros cumulativos presente no DDAG. Os AGs encontraram estruturas com taxas de acerto similares às do procedimento aleatório, que se mostrou assim suficientemente adequado a esse processo de busca.

Os valores dos desvios-padrão das taxas de acerto das soluções determinadas aleatoriamente foram baixos, principalmente se comparados ao método aleatório de construção de matrizes de códigos da seção anterior. Isso indica que não há muita variação de resultados entre DAGs com estruturas distintas. Pode-se assim concluir que este não é um problema crítico no uso dos DAGs com SVMs, contrariando as observações de (Kijisirikul & Ussivakul, 2002).

Nos conjuntos de dados *iris*, *car* e *structural*, em que todas as possíveis estruturas aleatórias foram geradas, pode-se confirmar que há poucas variações entre os desempenhos das soluções. Os valores de desvio-padrão nos conjuntos *iris* e *car*, embora mais altos, são semelhantes aos das matrizes 1ct e tct (Tabela 6.4), que são determinísticas. Logo, eles podem ser atribuídos ao uso do CV. O mesmo tipo de observação se aplica aos outros conjuntos de dados divididos por CV. Entre os conjuntos de dados não particionados por CV, houve uma variação de resultados um pouco maior no conjunto *vowel*. É interessante observar que, nos conjuntos de dados *lung* e *glass*, as taxas de acerto das soluções dos DAGs apresentaram valores menores de desvio-padrão nas partições de teste do que os das matrizes de códigos da Seção 6.3.

O desempenho das soluções dos AGs na Tabela 6.8 também foi estável, porém se manteve similar ao das estruturas aleatórias. Como há poucas diferenças de resultados

entre DAGs distintas, o problema de busca de DAGs com baixo erro no conjunto de validação apresentou pouca variabilidade. Os AGs podem então não ser adequados a esse tipo de solução, em que uma busca aleatória alcança resultados satisfatórios. Pode ser interessante, contudo, aplicar os AGs na busca da estrutura dos DAGs juntamente com os parâmetros das SVMs binárias contidos nos mesmos. Dessa forma, o espaço de busca e a sua variabilidade aumenta e, com isso, o método aleatório pode ser incapaz de encontrar soluções adequadas.

Não houve também diferenças no uso de pontos de corte aleatórios ou em pares nos cruzamentos dos AGs para os ADAGs.

► Outros resultados

O tempo de processamento (em segundos) dos AGs na busca das soluções reportadas é apresentado na Tabela 6.9. Novamente, os tempos foram dependentes do número de classes dos problemas e também dos tamanhos dos conjuntos de dados. Na lista de permutações já avaliadas, utilizada para acelerar o tempo de execução desses algoritmos, manteve-se um número máximo de soluções de 10 vezes o tamanho da população.

	DDAG	ADAG	
Conjunto	AG	AG ₁	AG ₂
<i>leukocyte</i>	0,8 (0,1)	0,9 (0,2)	0,9 (0,2)
<i>fungi</i>	2,5 (0,1)	3,3 (0,3)	3,1 (0,4)
<i>opt-digits</i>	9,4 (0,5)	10,7 (1,6)	11,4 (1,2)
<i>pen-digits</i>	17,5 (1,8)	23,0 (3,5)	23,2 (2,8)
<i>vowel</i>	2,6 (0,3)	2,8 (0,4)	2,8 (0,4)

Tabela 6.9: Tempos de processamento dos AGs para a obtenção de DAGs, em segundos

A procura de DDAGs foi um pouco mais rápida que a de ADAGs. Isso pode ser decorrente do fato dos DDAGs terem consultado classificadores binários mais rápidos na predição da classe dos dados contidos nos conjuntos de validação. Como o método aleatório é mais simples, seus tempos foram inferiores, podendo ser considerados nulos. Uma vez que o método aleatório produziu resultados satisfatórios a um menor custo, ele pode ser considerado uma melhor alternativa que o uso dos AGs neste problema.

Também nestes AGs os operadores genéticos de cruzamento foram aplicados de forma probabilística, dependendo de seu desempenho na produção de boas soluções com o decorrer das gerações. Verificou-se que em geral o operador PMX se destacou na produção das soluções finais dos AGs.

Gerou-se ainda um número maior de estruturas de DDAGs e ADAGs aleatórias nos conjuntos de dados. O objetivo foi o de obter uma confirmação do fato de que existem poucas diferenças entre os resultados de DAGs distintas quando as SVMs consideradas neste estudo são aplicadas na geração dos preditores binários. Um total de 10000 permutações diferentes foram investigadas em cada conjunto de dados. No caso em que o número de possíveis permutações $k!$ do problema era menor que 10000, gerou-se esse número total

de estruturas. Os resultados obtidos foram iguais ou bastante similares aos da Tabela 6.8, confirmando a pouca variação de desempenho entre diferentes estruturas de DAGs anteriormente observada. Manteve-se os resultados anteriores com trinta estruturas aleatórias para uma maior uniformidade com os AGs, que foram executados trinta vezes.

6.4.3 Resumo

Nesta seção investigou-se como a organização dos classificadores binários nos DDAGs e ADAGs afeta o seu desempenho na solução multiclasse quando SVMs são utilizadas na indução dos preditores. Utilizando os parâmetros $C = 100$ e um Kernel Gaussiano com $\sigma = 0,01$, foram encontradas poucas variações entre as taxas de acertos de estruturas de DAGs distintas.

Devido à baixa variabilidade dos diferentes DAGs, a aplicação de AGs nesse processo de busca levou a soluções com taxas de acerto estatisticamente similares às de estruturas determinadas aleatoriamente. De fato, mesmo explorando o espaço de busca de maneira limitada, com apenas trinta soluções, as taxas de acerto das estruturas aleatórias foram semelhantes às das principais estratégias baseadas em matrizes de códigos da Seção 6.3.

Esse resultado pode também ser efeito do bom desempenho das SVMs. Essa técnica de AM é bastante estável e apresenta boa capacidade de generalização em aplicações variadas (Hearst et al., 1998). Ele está ainda em conformidade com o limite de generalização de (Platt et al., 2000), que afirma que as SVMs, as quais maximizam as margens de separação das classes em cada nó dos DAGs, são adequadas a esse tipo de estrutura hierárquica. O uso de outras técnicas de AM na geração dos preditores binários dos DAGs pode então levar a resultados distintos.

O uso de AGs com SVMs neste problema não deve, porém, ser completamente descartado. Eles podem ser utilizados na determinação conjunta dos parâmetros das SVMs nos diferentes DAGs. Entretanto, deve-se salientar que, com isso, a complexidade do problema aumenta e também o custo envolvido em sua solução.

Os AGs também podem ser mais viáveis na busca de estruturas de DAGs para problemas com números elevados de classes. Em espaços de busca maiores, pode haver maiores variações entre soluções distintas, tornando mais difícil realizar uma amostragem aleatória de DAGs com bom desempenho.

Contrariando as observações de (Kijssirikul & Ussivakul, 2002), também verificou-se que os DDAGs, quando utilizados com SVMs, apresentaram resultados similares aos dos ADAGs, mesmo em aplicações com números maiores de classes. Porém, são necessários experimentos em conjuntos de dados com mais classes para confirmar essa observação.

Como alternativa a determinar uma única estrutura de DAG em um conjunto de dados, a combinação de diferentes soluções pode ser explorada. As pequenas variações verificadas entre as estruturas distintas podem assim se complementar, produzindo melhores resultados.

6.5 Determinação da Estrutura de Árvores Multiclasses

Como apontado em (Frank & Kramer, 2004), existem $\prod_{i=3}^k 2i - 3$ árvores multiclasses distintas para solucionar um problema com $k \geq 3$ classes. Nesta seção investiga-se como a organização dos preditores binários nesse tipo de estrutura influencia o seu desempenho na solução multiclasses. Aplicando os algoritmos e heurísticas do Capítulo 5, avalia-se como o uso de informações extraídas de cada aplicação pode auxiliar nessa organização.

6.5.1 Configuração dos Experimentos

Os algoritmos 5.4 e 5.5 do Capítulo 5 foram aplicados na determinação das partições binárias nos nós de árvores multiclasses para os doze conjuntos de dados investigados nesta Tese. Para facilitar as referências a esses algoritmos, o Algoritmo 5.4 será denotado por “Alg1” e o Algoritmo 5.5, por “Alg2”.

Na construção dos grafos com informações de cada aplicação sobre os quais Alg1 e Alg2 são aplicados, empregou-se o critério de distâncias entre centróides, que podera as relações entre as classes dos problemas por meio das distâncias entre os seus centros, e o critério de classes de confusão, que define essas relações por meio da confusão encontrada em sua separação por ADs. Os centróides foram calculados a partir dos dados de treinamento. As matrizes de confusão para o critério de classes de confusão foram obtidas a partir do teste das ADs, geradas sobre os subconjuntos de treinamento T' , nos dados de validação. As ADs foram induzidas com o uso do algoritmo C4.5 (Quilan, 1988). Para conjuntos de dados divididos por CV, foram geradas árvores multiclasses a partir dos dados de cada partição T_i (T'_i e V_i no caso do critério de classes de confusão), as quais foram então avaliadas em seus subconjuntos de teste correspondentes S_i .

Como nas seções anteriores, gerou-se também trinta estruturas aleatórias para cada conjunto de dados. Essas estruturas foram criadas definindo recursivamente partições binárias aleatórias das classes nos nós das árvores. Para testes estatísticos, selecionou-se, em cada partição, a árvore que apresentasse taxa de acerto mais próxima à taxa média de todas as estruturas originadas. Procurou-se ainda minimizar a produção de árvores aleatórias idênticas. Para o conjunto de dados *iris*, com três classes, todas as possíveis estruturas de árvores foram geradas.

6.5.2 Resultados

Nesta seção são apresentados os resultados verificados na determinação da estrutura de árvores multiclasses para os doze conjuntos de dados considerados nesta Tese.

► Taxas de acerto

Na Tabela 6.10 são apresentadas as taxas de acerto das árvores multiclasses obtidas com o uso dos algoritmos Alg1 e Alg2, utilizando os critérios de distância entre centróides (colunas “Centróide”) e de classes de confusão (colunas “Cl. confusão”). Tem-se também

as taxas de acerto médias das árvores cujas partições binárias nos nós foram definidas de maneira aleatória. Valores de desvio-padrão são indicados entre parênteses.

Conjunto	Alg1		Alg2		Aleatório
	Centróide	Cl. confusão	Centróide	Cl. confusão	
<i>iris</i>	96,7 (3,5)	96,7 (3,5)	96,7 (3,5)	96,7 (3,5)	96,9 (3,3)
<i>car</i>	98,0 (1,3)	97,9 (1,3)	98,0 (1,3)	97,9 (1,3)	94,6 (3,3)
<i>lung</i>	76,8 (1,0)	76,9 (1,6)	76,8 (1,0)	77,3 (0,8)	80,9 (4,3)
<i>glass</i>	68,7 (5,8)	66,4 (7,4)	68,2 (6,7)	69,6 (5,7)	68,8 (6,6)
<i>segment</i>	96,0 (1,3)	96,3 (1,0)	96,0 (1,3)	96,4 (1,0)	95,0 (1,8)
<i>leukocyte</i>	88,8 (2,0)	89,0 (1,6)	89,2 (2,7)	89,8 (1,8)	88,6 (2,7)
<i>fungi</i>	52,3 (2,9)	51,1 (2,5)	52,1 (3,0)	50,0 (3,1)	54,2 (4,6)
<i>structural</i>	76,6	79,5	76,6	79,5	79,4 (2,1)
<i>sat-image</i>	89,6	89,9	90,1	89,7	89,2 (0,4)
<i>opt-digits</i>	96,8	96,8	97,2	97,4	96,4 (0,4)
<i>pen-digits</i>	97,6	97,3	97,6	96,9	97,2 (0,2)
<i>vowel</i>	54,8	55,6	56,5	57,6	49,3 (4,6)

Tabela 6.10: Taxas de acerto das árvores multiclases

É interessante observar que os valores de desvio-padrão do método aleatório novamente não foram altos se comparados aos das matrizes aleatórias da Seção 6.3. Isso indica que as soluções determinadas não apresentaram uma grande variabilidade. Porém, os valores verificados são maiores que os dos DAGs aleatórios na Tabela 6.8. Logo, estruturas de árvores distintas apresentam alterações de desempenho maiores que no caso dos DDAGs e ADAGs.

► Testes estatísticos

Comparando estatisticamente as taxas apresentadas, verificou-se em geral que todas as árvores geradas tiveram desempenho similar. Os resultados dos testes realizados são resumidos nos quadros da Figura 6.5 e nos gráficos da Figura 6.6, como realizado na Seção 6.3 para as estratégias baseadas em matrizes de códigos. Nessas figuras, “*ci*” e “*cci*” indicam, respectivamente, os critérios de distância entre centróides e de classes de confusão aplicados com o algoritmo “Alg i ”. Uma tabela com os resultados dos testes realizados também é apresentada na Seção B.3 do Apêndice B.

Na obtenção do quadro da Figura 6.5a, para cada conjunto de dados, indicou-se no bloco “Melhores Taxas” a árvore de maior taxa de acerto e aquelas que apresentaram resultados estatisticamente similares à ela. Como os resultados de grande parte das estruturas geradas nesse estudo foram próximos, em alguns casos, uma parcela das árvores com taxa de acerto similar à melhor também obteve desempenho semelhante à pior taxa. Essa afirmação pode ser verificada pela grande quantidade de sobreposições entre as faixas horizontais que conectam as árvores com taxas de acerto estatisticamente similares na Figura 6.6. Isso ocorreu porque essas estruturas apresentaram um desempenho intermediário entre os dois extremos, sendo assim próximos a ambos. Decidiu-se então indicá-las apenas entre as melhores, uma vez que elas são tão aptas a solucionar os problemas quanto as

Conjunto	Melhores taxas					Piores taxas				
	c1	cc1	c2	cc2	ale	c1	cc1	c2	cc2	ale
<i>iris</i>										
<i>car</i>										
<i>lung</i>										
<i>glass</i>										
<i>segment</i>										
<i>leukocyte</i>										
<i>fungi</i>										
<i>structural</i>										
<i>sat-image</i>										
<i>opt-digits</i>										
<i>pen-digits</i>										
<i>vowel</i>										
%	92	92	92	83	67	8	8	8	17	33

(a)

	# Dif.	Melhor	%
c1-cc1	1	cc1	100
c1-c2	0	-	-
c1-cc2	2	c1-cc2	50
c1-ale	3	c1	67
cc1-c2	1	cc1	100
cc1-cc2	0	-	-
cc1-ale	3	cc1	67
c2-cc2	2	c2-cc2	50
c2-ale	4	c2	75
cc2-ale	5	cc2	80
		cc1-cc2	30

(b)

Figura 6.5: Quadros resumo dos testes entre árvores multiclasses

estruturas de melhor desempenho. Em “Piores taxas” indicou-se somente as árvores com piores taxas de acerto que as da melhor solução encontrada em cada conjunto de dados.

O que se destaca nos testes é o fato de grande parte das árvores terem apresentado taxas de acerto similares. Isso é comprovado pela grande quantidade de células preenchidas em cinza no bloco “Melhores taxas” do quadro da Figura 6.5a. Ou seja, várias das árvores geradas em cada conjunto multiclasses apresentaram a mesma capacidade de classificação daquela de maior taxa de acerto. Há um destaque um pouco maior na maioria dos conjuntos de dados das árvores “c1”, “cc1” e “c2”, que estiveram entre as de melhor taxas de acerto em onze dos doze conjuntos de dados.

Nesse quadro também averigua-se, para cada aplicação, que as árvores aleatórias médias figuram mais vezes entre as soluções de piores taxas de acerto que as de melhor taxa. Nos conjuntos de dados *car* e *segment*, as árvores aleatórias de desempenho médio tiveram taxas de acerto estatisticamente diferentes e inferiores às de todas as outras árvores criadas.

Nas linhas em cinza claro no quadro da Figura 6.5b tem-se também que, na maioria das comparações estatísticas em que houve diferenças entre as árvores determinadas por Alg1 e Alg2 e as aleatórias médias, as árvores obtidas com o uso das heurísticas propostas se sobressaíram, obtendo maiores taxas de acerto. Somente no conjunto de dados *fungi*, as árvores de Alg2 geradas com o critério de classes de confusão foram inferiores às árvores aleatórias médias. No conjunto de dados *structural* ocorreu o mesmo para as árvores obtidas por Alg1 e Alg2 geradas com o critério de distâncias entre centróides.

Na maioria dos conjuntos de dados, porém, as árvores geradas pelos algoritmos propostos obtiveram taxas de acerto similares às das árvores aleatórias de desempenho médio. Com isso, apesar das árvores aleatórias médias terem apresentado pior desempenho em 33% dos conjuntos de dados, na maioria (67%) deles suas taxas de acerto estiveram entre

Figura 6.6: Comparações das taxas de acerto das árvores multiclases

as melhores e foram semelhantes às de árvores determinadas pelos algoritmos propostos.

No conjunto de dados *structural*, o uso das confusões entre as classes na definição das partições binárias levou a árvores de desempenho superior ao das árvores obtidas com o critério de distâncias entre centróides. Também observou-se anteriormente que, nesse conjunto, as árvores determinadas por este critério tiveram desempenho inferior ao das árvores aleatórias médias. O oposto ocorreu no conjunto de dados *fungi*, em que separar as classes com base na distância entre seus centróides foi melhor que com o uso da informação de confusão. Também no conjunto de dados *pen-digits*, o desempenho de Alg2 com o critério de classes de confusão foi inferior ao de Alg1 e Alg2 com as distâncias de centróides. Esses resultados podem ser consequência das características desses conjuntos de dados, que podem então ter tornado um critério mais adequado que o outro na definição

das partições binárias de classes nos nós das árvores multiclasses.

Também observa-se nos resultados reportados que as modificações introduzidas por Alg2 em relação a Alg1 no processo de agrupamento das classes não implicaram na geração de árvores multiclasses com taxas de acerto estatisticamente diferentes. Essa informação pode ser visualizada nas linhas destacadas em cinza escuro no quadro da Figura 6.5b, em que indica-se que não houve diferenças estatísticas nas comparações “c1-c2” e “cc1-cc2”.

► Outros resultados

Calculou-se também os tempos, em segundos, tomados na obtenção das diversas árvores multiclasses geradas neste trabalho. Eles podem ser considerados nulos no caso das árvores aleatórias e do critério de distância entre centróides. No uso do critério de classes de confusão, que requer o treinamento e teste de ADs, há um custo computacional maior. Na Tabela 6.11 são ilustrados então os tempos médios de Alg1 e Alg2 quando o critério de classes de confusão foi empregado.

	Alg1	Alg2
Conjunto	Cl. confusão	Cl. confusão
<i>iris</i>	0,0 (0,0)	0,0 (0,0)
<i>car</i>	0,0 (0,0)	0,1 (0,0)
<i>lung</i>	0,3 (0,1)	2,0 (0,1)
<i>glass</i>	0,0 (0,0)	0,1 (0,0)
<i>segment</i>	0,2 (0,0)	1,7 (0,0)
<i>leukocyte</i>	0,8 (0,0)	5,2 (0,1)
<i>fungi</i>	1,7 (0,1)	13,5 (0,3)
<i>structural</i>	0,2	0,7
<i>sat-image</i>	0,6	4,1
<i>opt-digits</i>	0,5	7,8
<i>pen-digits</i>	0,4	7,3
<i>vowel</i>	0,1	0,6

Tabela 6.11: Tempos de execução de algoritmos para a obtenção de árvores multiclasses

O tempo de Alg2 foi superior ao de Alg1 em todos os casos. Isso ocorre devido às adaptações realizadas nos grafos a partir dos quais as árvores multiclasses são extraídas, que requer assim a realização de um número maior de treinamentos e testes de ADs. Como as taxas de acerto das árvores definidas por ambos os algoritmos foram similares, Alg1 pode ser considerada uma melhor alternativa devido a seu menor custo computacional.

Como no caso dos DAGs da Seção 6.4, gerou-se também um número maior de estruturas de árvores aleatórias, para investigar se maiores variações de desempenho poderiam ser verificadas entre árvores multiclasses diferentes. Investigou-se um total de 10000 estruturas aleatórias para cada problema. Quando o número de possíveis estruturas para um problema era menor que 10000, esse número total foi gerado.

Os resultados observados foram bastante similares aos da Tabela 6.10. As maiores diferenças em relação aos resultados anteriores foram observadas nos conjuntos de dados *fungi* e *vowel*. No conjunto *fungi*, a taxa de acerto média das estruturas aleatórias passou

para 53,8 com um desvio-padrão de 4,4. No caso do conjunto *vowel*, o novo valor foi de 48,2 com um desvio-padrão de 4,2. Com exceção dessas pequenas diferenças nas taxas médias de acerto, todos os valores de desvio-padrão encontrados foram semelhantes aos verificados na Tabela 6.10. Tem-se assim uma confirmação de que as variações de desempenho entre estruturas de árvores multiclases distintas não foram muito altas quando as SVMs consideradas neste estudo foram aplicadas na indução dos preditores binários.

6.5.3 Resumo

Nesta seção foi investigado o problema de determinação da estrutura de árvores multiclases. Foram então aplicados os dois algoritmos do Capítulo 5 que, por meio de um agrupamento hierárquico das classes dos problemas, definem as partições binárias das classes nos nós das árvores multiclases. O agrupamento é realizado a partir de informações da relação entre as classes, as quais são extraídas dos dados de cada aplicação.

Foram testados dois critérios na definição dessas relações: de distância entre centróides e de classes de confusão. No primeiro, separa-se as classes nos nós da árvore de acordo com a distância entre os seus centróides, os quais são calculados a partir dos dados de treinamento. No critério de classes de confusão, utiliza-se uma medida da confusão entre as classes, que é determinada a partir de matrizes de confusão obtidas por ADs na solução do problema multiclases total.

Nos experimentos realizados não houve diferenças significativas entre as soluções obtidas pelos algoritmos 5.4 e 5.5, que mostraram taxas de acerto estatisticamente similares. Ou seja, o desempenho das árvores obtidas por agrupamentos baseados na relação de pares de classes foi semelhante ao daquelas em que calculou-se a relação entre os subconjuntos de classes agrupados. O Algoritmo 5.4 apresenta a vantagem de ser mais simples e, com isso, menos custoso.

Entre os critérios utilizados para extrair as informações das classes de cada aplicação, também houve semelhança de resultados. Em apenas três dos doze conjuntos de dados empregados um dos critérios se sobressaiu em relação ao outro. Este fato indica, porém, que, dependendo das características dos dados, pode ser mais apropriado aplicar uma das abordagens de ponderação da relação entre as classes.

De fato, o uso de centróides não é adequado quando os dados das classes se apresentam muito dispersos ou quando eles formam subgrupos. Neste caso, os centróides não refletem bem a distribuição dos dados. A distância Euclidiana também é limitada no sentido de assumir grupos esféricos. Uma alternativa pode ser utilizar a distância de Mahalanobis. No critério de classes de confusão, mediu-se a confusão entre as classes por meio de ADs. Deve ser observado no entanto que as ADs possuem um viés (*bias*) próprio na separação dos dados, diferente do viés das SVMs. Com isso, classes que apresentam confusão para as ADs não necessariamente demonstram essa característica para as SVMs. Como o tempo de execução dos algoritmos propostos é pequeno, principalmente se comparado aos dos AGs das seções anteriores, é recomendável então testar ambos os critérios e verificar qual

é mais adequado à aplicação sendo solucionada.

Verificou-se ainda que árvores cujas partições binárias foram definidas de maneira aleatória em vários casos tiveram desempenho similar às geradas pelas heurísticas anteriores. Além disso, elas demonstraram poucas variações de desempenho, embora sejam maiores que as encontradas entre estruturas distintas de DDAGs e ADAGs. A boa capacidade de generalização e a estabilidade das SVMs pode ter contribuído para esses resultados. Como no caso dos DAGs, existe um limite de generalização para as árvores multiclases que afirma que as SVMs, as quais maximizam as margens de separação entre as classes nos nós, são adequadas à solução dos subproblemas binários contidos nesse tipo de estrutura hierárquica (Vural & Dy, 2004).

Como realizado em (Frank & Kramer, 2004), uma melhor alternativa a escolher uma única árvore pode ser combinar várias estruturas distintas, explorando assim complementariedades que estejam presentes em seus resultados.

6.6 Comparações de Diferentes Estratégias

Anteriormente, as estratégias baseadas em matrizes de códigos e hierárquicas com estrutura de DAGs e de árvores multiclases foram analisadas isoladamente. Nesta seção, selecionou-se representantes de cada um desses grupos de técnicas para serem comparados.

6.6.1 Configuração dos Experimentos

No caso das matrizes de códigos, escolheu-se aquelas geradas pelo AG lexicográfico na Seção 6.3, as quais apresentaram em geral taxas de acerto comparáveis às das outras estratégias baseadas em matrizes de códigos em todos os conjuntos de dados.

Uma vez que os ADAGs tiveram desempenho similar ao dos DDAGs na Seção 6.4, o representante dos DAGs selecionado foi o DDAG, o qual é mais conhecido na literatura.

No caso das árvores multiclases, escolheu-se em cada conjunto de dados a árvore de melhor desempenho gerada pelo Algoritmo 5.4 (Alg1 na Seção 6.5). Ou seja, nos conjuntos de dados *iris*, *car*, *glass*, *fungi*, *opt-digits* e *pen-digits* utilizou-se as árvores geradas com o critério de distância entre centróides. Nos demais conjuntos de dados, empregou-se as árvores do critério de classes de confusão. Nos casos em que as taxas de acerto das árvores de ambos os critérios foram iguais, selecionou-se o de distância entre centróides.

Também examinou-se o desempenho das técnicas de Weston & Watkins (1998) (WW) e de Crammer & Singer (2002) (CS), que reformulam os algoritmos de treinamento das SVMs, criando versões multiclases. Utilizou-se o simulador BSVM (Chang & Lin, 2004a) na indução desses modelos.

6.6.2 Resultados

Esta seção apresenta os resultados das diferentes estratégias multiclases anteriormente mencionadas.

► Taxas de Acerto

As taxas de acerto das estratégias comparadas encontram-se indicadas na Tabela 6.12. Os resultados das matrizes do AG lexicográfico (coluna “AG lex”), dos DDAGs e das árvores foram reproduzidos das tabelas 6.4, 6.8 e 6.10, respectivamente. Os DDAGs correspondem aos criados aleatoriamente.

Conjunto	AG lex	DDAG	Árvore	WW	CS
<i>iris</i>	95,0 (3,6)	96,7 (3,4)	96,7 (3,5)	95,3 (4,5)	96,7 (3,5)
<i>car</i>	97,8 (1,4)	97,8 (1,3)	98,0 (1,3)	96,6 (2,1)	97,6 (1,7)
<i>lung</i>	85,5 (4,8)	77,3 (0,6)	76,9 (1,6)	82,8 (3,0)	92,6 (5,4)
<i>glass</i>	68,3 (7,2)	70,4 (4,2)	68,7 (5,8)	71,0 (6,8)	68,2 (9,3)
<i>segment</i>	96,2 (1,1)	96,7 (0,9)	96,3 (1,0)	96,0 (1,3)	96,7 (1,4)
<i>leukocyte</i>	89,2 (2,9)	90,1 (2,3)	89,0 (1,6)	89,4 (2,9)	89,7 (2,4)
<i>fungi</i>	62,0 (2,6)	60,9 (2,4)	52,3 (2,9)	61,5 (2,5)	61,7 (2,6)
<i>structural</i>	80,1 (0,9)	81,1 (0,1)	79,5	81,0	81,0
<i>sat-image</i>	90,0 (0,4)	89,6 (0,1)	89,9	90,2	90,4
<i>opt-digits</i>	96,7 (0,5)	97,1 (0,1)	96,8	97,3	97,1
<i>pen-digits</i>	97,4 (0,3)	97,7 (0,1)	97,6	97,7	97,6
<i>vowel</i>	50,3 (3,2)	55,1 (0,8)	55,6	53,9	51,3

Tabela 6.12: Taxas de acerto de diferentes estratégias multiclases

É interessante observar que as taxas de acerto de diferentes DDAGs geradas em um mesmo conjunto de dados foram mais estáveis do que as de matrizes de códigos distintas, uma vez que os seus valores de desvio-padrão foram inferiores.

► Testes Estatísticos

As taxas de acerto das estratégias apresentadas foram comparadas estatisticamente, para verificar mais claramente as diferenças entre as diversas técnicas. Para as matrizes do AG lexicográfico e os DDAGs, selecionou-se as soluções de desempenho médio em cada partição de treinamento/teste de um conjunto de dados como representante para os testes estatísticos.

Como nas seções anteriores, os resultados dos testes realizados são resumidos nos quadros da Figura 6.7 e nos gráficos da Figura 6.8. Eles foram construídos como nas seções 6.3 e 6.5. Uma tabela com as comparações também pode ser encontrada no Apêndice B (Tabela B.16).

A primeira observação sobre os testes realizados é que as taxas de acerto de todas as estratégias foram em geral similares. Isso pode ser comprovado pelo grande número de células preenchidas em cinza no bloco “Melhores taxas” do quadro da Figura 6.7a. Porém, algumas exceções e aspectos interessantes podem ser apontados.

As matrizes de códigos do AG lexicográfico e o algoritmo CS estiveram entre as técnicas de maiores taxas de acerto em todos os conjuntos de dados. Além disso, em nenhum dos conjuntos suas taxas de acerto se encontraram entre as piores. Esses fatos podem ser observados nas porcentagens apresentadas na última linha do quadro da Figura 6.7a.

Conjunto	Melhores taxas					Piores taxas				
	lex	dda	arv	ww	cs	lex	dda	arv	ww	cs
<i>iris</i>										
<i>car</i>										
<i>lung</i>										
<i>glass</i>										
<i>segment</i>										
<i>leukocyte</i>										
<i>fungi</i>										
<i>structural</i>										
<i>sat-image</i>										
<i>opt-digits</i>										
<i>pen-digits</i>										
<i>vowel</i>										
%	100	92	83	75	100	0	8	17	25	0

(a)

	# Dif.	Melhor	%
lex-dda	1	lex	100
lex-arv	2	lex	100
lex-ww	1	lex	100
lex-cs	0	-	-
dda-arv	1	dda	100
dda-ww	2	dda	100
dda-cs	1	cs	100
arv-ww	2	ww, arv	50
arv-cs	2	cs	100
ww-cs	2	cs	100
		lex-cs	30

(b)

Figura 6.7: Quadros resumo dos testes entre diferentes estratégias multiclases

No quadro da Figura 6.7b tem-se ainda que as matrizes do AG lexicográfico (“lex”) e o algoritmo CS se sobressaíram mais vezes nas diferenças estatísticas detectadas entre todos os pares de técnicas. Ainda nesse quadro, a linha em cinza claro evidencia que não houve nenhum caso de diferenças estatísticas entre essas duas estratégias.

Na Seção 3.2.2, mostrou-se que a técnica CS realiza modificações no algoritmo de treinamento das SVMs de forma a determinar uma matriz de códigos com elementos contínuos adequada a cada aplicação multiclases. O AG lexicográfico, que também adequa matrizes de códigos a cada problema, porém com elementos discretos em $\{-1, 0, +1\}$, foi então capaz de encontrar soluções com taxas de acerto semelhantes às do algoritmo CS.

O algoritmo WW, por sua vez, foi a estratégia que esteve mais vezes entre as de piores taxas de acerto em relação à maior taxa nos conjuntos de dados, embora na maioria deles (75%) as suas taxas de acerto tenham se mantido entre as melhores. Essas observações são ilustradas no quadro da Figura 6.7a.

As árvores multiclases figuram em seguida na parte “Piores taxas”. No conjunto de dados *lung*, em particular, o DDAG também teve desempenho deteriorado. Logo, ambas as estruturas hierárquicas investigadas não se mostraram adequadas a solucionar esse problema multiclases frente às outras estratégias. Na Tabela 6.10 e na Figura 6.6, observa-se que a maior taxa de acerto obtida por uma árvore multiclases no conjunto de dados *fungi* foi a do método aleatório. Comparando as taxas de acerto das árvores multiclases aleatórias de desempenho médio nas partições com as das técnicas consideradas nesta seção, observou-se que as árvores multiclases continuaram apresentando resultados inferiores. Esse tipo de estrutura hierárquica não se mostrou assim apropriado a solucionar o problema multiclases considerado se comparado às outras estratégias.

Apesar desses casos, em grande parte dos conjuntos de dados (dez dos doze), as árvores multiclases apresentaram desempenho comparável ao das estratégias de maior taxa de

Figura 6.8: Comparações das taxas de acerto de diferentes estratégias multiclases

acerto. Nos conjuntos de dados *car* e *vowel*, particularmente, as suas taxas de acerto foram as maiores.

Na comparação das técnicas WW e CS, que reformulam o algoritmo de treinamento das SVMs em versões multiclases, a CS se sobressaiu. Segundo a linha em cinza escuro no quadro da Figura 6.7b, nas duas diferenças identificadas entre esses algoritmos, o CS apresentou as maiores taxas de acerto.

Comparando ainda as estruturas hierárquicas DDAG e árvore, verifica-se uma pequena vantagem do DDAG. Isso é ilustrado na linha em cinza intermediário do quadro da Figura 6.7b, em que o DDAG se destacou na diferença estatística detectada entre essas estruturas.

► Número de Classificadores Binários em Decomposições

Na Tabela 6.13 são apresentados os números de classificadores binários induzidos nas estratégias que realizam decomposições dos problemas multiclases: as matrizes de códigos do AG lexicográfico, os DDAGs e as árvores multiclases. Os resultados do AG lexicográfico foram reproduzidos da Tabela 6.5.

Conjunto	AG lexic	DDAG	Árvore
<i>iris</i>	2,0 (0,1)	3	2
<i>car</i>	3,6 (0,5)	6	3
<i>lung</i>	3,6 (0,7)	10	4
<i>glass</i>	4,6 (0,8)	15	5
<i>segment</i>	6,5 (0,7)	21	6
<i>leukocyte</i>	7,8 (0,4)	28	7
<i>fungi</i>	7,7 (1,2)	36	8
<i>structural</i>	3,8 (0,4)	6	3
<i>sat-image</i>	6,0 (0,2)	10	5
<i>opt-digits</i>	9,8 (0,6)	45	9
<i>pen-digits</i>	9,9 (0,3)	45	9
<i>vowel</i>	10,4 (1,1)	55	10

Tabela 6.13: Números de classificadores em diferentes estratégias decomposicionais

Nessa tabela é possível verificar que as árvores multiclases geralmente apresentam os menores números de classificadores binários, sendo $k - 1$ para um problema com k classes. Em seguida figuram as matrizes do AG lexicográfico, que em alguns casos (nos conjuntos de dados *lung*, *glass* e *fungi*) chegaram a apresentar um número médio de classificadores binários inferior ao das árvores. O DDAG, que integra classificadores gerados segundo a decomposição tct, ou seja, para todos os pares de classes, apresenta os maiores números de preditores ($\frac{k(k-1)}{2}$).

6.6.3 Resumo

Nesta seção comparou-se o desempenho de estratégias que empregam abordagens distintas na generalização das SVMs a problemas multiclases. As baseadas em matrizes de códigos foram representadas pelas soluções do AG lexicográfico. Entre as hierárquicas, analisou-se os DDAGs e as árvores multiclases. Foram ainda testados dois algoritmos diretos (WW e CS), que reformulam o treinamento das SVMs para solucionar o problema multiclases total.

As matrizes do AG lexicográfico e o algoritmo CS estiveram entre as técnicas de maiores taxas de acerto em todas as comparações realizadas. Ambos realizam adaptações de matrizes de códigos a cada problema multiclases, embora os métodos empregados por cada um deles nesse processo sejam diferentes. O algoritmo CS reformula o problema de otimização envolvido no treinamento das SVMs, determinando neste procedimento matrizes de códigos de norma máxima com elementos contínuos e com baixo erro no conjunto de treinamento (Seção 3.2.2). No caso do AG lexicográfico, busca-se matrizes com ele-

mentos discretos que apresentem baixo erro em subconjuntos de validação, procurando ainda reduzir o número de classificadores binários empregados na decomposição.

Embora algumas técnicas (a WW, as árvores multiclases e os DDAGs) tenham apresentado desvantagem em termos de taxa de acerto em alguns conjuntos de dados, em geral os resultados das estratégias multiclases consideradas, as quais usam diferentes abordagens na generalização das SVMs a problemas multiclases, foram semelhantes. Outros critérios devem então ser explorados na escolha de uma estratégia particular para solucionar uma dada aplicação. Uma discussão geral a esse respeito é realizada na próxima seção.

Com relação às árvores multiclases, atestou-se que as estruturas determinadas pelo Algoritmo 5.4 foram geralmente plausíveis, apresentando taxas de acerto estatisticamente similares às melhores em dez dos doze conjuntos de dados empregados. Entre as estratégias decomposicionais, elas apresentam a vantagem de utilizar menos preditores binários.

6.7 Discussões

Nesta seção são realizadas algumas observações em relação aos resultados e análises experimentais previamente apresentados. São também discutidos outros pontos pertinentes à pesquisa realizada.

6.7.1 Desempenho nos Conjuntos de Dados Reais

Embora o objetivo dos experimentos realizados anteriormente não tenha sido obter as melhores taxas de acerto possíveis em cada conjunto de dados, analisou-se o comportamento das técnicas testadas na solução dos problemas reais da área de Bioinformática. Eles são representados pelos conjuntos de dados *lung*, *leukocyte*, *fungi* e *structural*.

No conjunto de dados *lung*, as estratégias tct, DDAG, ADAG e as árvores multiclases obtiveram taxas de acerto inferiores às das outras técnicas. O mesmo tipo de resultado foi verificado por Statnikov et al. (2005), em que as estratégias 1ct e CS se sobressaíram sobre a tct e a DDAG, embora em menor proporção que a observada nas tabelas 6.4 e 6.12. No presente trabalho, as matrizes do AG lexicográfico também estiveram entre as de maiores taxas de acerto, sendo capazes inclusive de melhorar o desempenho da decomposição 1ct.

Segundo Yeang et al. (2001), os conjuntos de dados para a classificação de tumores a partir de medidas de expressão gênica geralmente contêm poucos exemplos. O número de casos disponíveis para o treinamento de cada classificador binário utilizado na decomposição tct torna-se então muito reduzido e eles são, como consequência, mais sujeitos ao efeito de super-ajustamento (*overfitting*). Essa é a razão provável do desempenho deteriorado das estratégias tct, DDAG e ADAG nesse conjunto de dados. A pouca quantidade de exemplos também pode ter prejudicado a extração de informações das classes para a definição da estrutura das árvores multiclases por meio dos algoritmos do Capítulo 5. Curiosamente, mesmo frente a um número reduzido de dados, o AG lexicográfico foi capaz de adaptar matrizes de códigos a esse problema, as quais apresentaram bom desempenho

em sua solução.

As taxas de acerto das SVMs induzidas em (Statnikov et al., 2005) para esse mesmo conjunto de dados foram superiores às do presente estudo. Porém, elas não são diretamente comparáveis, uma vez que Statnikov et al. (2005) realizaram um procedimento de seleção de parâmetros para as SVMs, ajustando melhor os classificadores à aplicação considerada.

No conjunto de dados *leukocyte*, as diferentes estratégias multiclases testadas nesta Tese mostraram taxas de acerto semelhantes. Em seu trabalho de Doutorado, Ushizima (2004) empregou classificadores NB, ADs e RNAs Perceptron Multicamadas na solução desse mesmo problema. Embora as partições dos dados tenham sido diferentes, dificultando uma comparação direta, foi observado que somente as RNAs obtiveram resultados semelhantes aos das SVMs, enquanto os classificadores NB e as ADs demonstraram pior desempenho.

As matrizes do AG lexicográfico alcançaram as maiores taxas de acerto no conjunto de dados *fungi*. As estratégias tct, ECOC, DDAG, ADAG, WW e CS também obtiveram bom desempenho. As árvores multiclases e a matriz 1ct tiveram resultados inferiores aos dessas estratégias. O prejuízo observado pode ser devido a semelhanças existentes entre os dados de diferentes classes nesse domínio. Com isso, os classificadores binários para algumas das classes na decomposição 1ct apresentaram dificuldades em sua diferenciação. No caso das árvores multiclases, essa característica pode ter causado prejuízos à extração de informações sobre as classes para a determinação da estrutura dessas árvores.

De forma geral, porém, as taxas de acerto de todas as SVMs induzidas nesse conjunto de dados foram bastante inferiores às alcançadas por classificadores NB em (Lu et al., 2004). Essas diferenças se mantiveram altas mesmo após a realização de um ajuste de parâmetros para as SVMs em (Lorena & Carvalho, 2006b). Deve-se observar que a representação das proteínas adotada no presente trabalho difere daquela de (Lu et al., 2004), em que utilizou-se informações textuais extraídas a partir de uma busca por proteínas homólogas. Em (Lu et al., 2004), SVMs induzidas utilizando os atributos baseados em homologia mostraram taxas de acerto superiores às dos classificadores NB, contrariando os resultados inferiores das SVMs nesta Tese e em (Lorena & Carvalho, 2006b). Embora os experimentos desses trabalhos não sejam diretamente comparáveis, uma vez que as partições dos dados empregadas não foram as mesmas, essa diferença indica que a representação de proteínas adotada nesta Tese pode ser muito simples comparada à usada em (Lu et al., 2004).

No conjunto de dados *structural*, as diferentes estratégias multiclases também tiveram desempenho similar. As taxas de acerto reportadas na literatura para a predição da classe estrutural de proteínas são próximas (Bahar et al., 1997; Luo et al., 2002; Isik et al., 2004) ou superiores (Cai et al., 2001, 2002) às obtidas nesta Tese. Porém, não foram encontrados trabalhos utilizando o mesmo conjunto de dados aqui empregado, não permitindo assim uma comparação direta.

Outros experimentos realizados no período de Doutorado sobre conjuntos de dados multiclases da área de Bioinformática podem ser consultados em (Lorena et al., 2002a,b; Lorena & Carvalho, 2003a,b, 2004b,c,d, 2005a,b; Ushizima et al., 2005; Lorena & Carvalho, 2006a,b).

6.7.2 Outras Técnicas da Literatura

Como apontado no Capítulo 3, existem outras técnicas na literatura para a adaptação de matrizes de códigos aos problemas multiclases, bem como para a definição da estrutura de DDAGs, ADAGs e de árvores multiclases. Nesta seção esses estudos são brevemente comentados. Como na seção anterior, é difícil comparar os resultados desses trabalhos com os das seções 6.3 a 6.6. Isso porque alguns deles apresentam apenas resultados parciais, enquanto em outros, o protocolo experimental adotado foi diferente.

► Obtenção de Matrizes de Códigos

O primeiro algoritmo para a construção de matrizes de códigos revisado na Seção 3.1.1 foi o de Mayoraz & Moreira (1996). Ele adiciona classificadores binários à matriz de códigos considerados pertinentes de acordo com a distribuição dos dados das classes. Procura ainda maximizar a distância entre as linhas e as colunas dessa matriz, segundo os princípios para a obtenção de códigos com propriedade de correção de erros.

Mayoraz & Moreira (1996) realizaram experimentos com quatro conjuntos de dados do repositório UCI (Blake & Merz, 1998), com o objetivo demonstrar que o algoritmo proposto é capaz de obter matrizes que realizam decomposições mais simples que o ECOC, mantendo a sua capacidade de generalização. O algoritmo C4.5 (Quilan, 1988) foi utilizado na indução dos classificadores binários. Observou-se que as matrizes de códigos obtidas mantiveram as taxas de acerto do ECOC e, salvo uma exceção, também do algoritmo C4.5 aplicado diretamente à solução do problema multiclases. As matrizes construídas utilizaram menos preditores binários que o ECOC. A complexidade dos modelos finais, medida em termos do número de nós das ADs induzidas, também foi reduzida em relação à do ECOC.

Há um conjunto de dados comum nos experimentos desses autores com os desta Tese: o *glass*. Entretanto, enquanto Mayoraz & Moreira (1996) empregaram *3-fold cross-validation* 25 vezes na avaliação dos resultados, neste trabalho utilizou-se uma única replicação de *5-fold cross-validation*. Além disso, a técnica de aprendizado empregada também foi diferente. De qualquer maneira, as taxas de acerto obtidas pelas decomposições desses dois estudos foram similares. As matrizes determinadas nesta Tese, porém, utilizaram menos classificadores binários, se mostrando assim ainda mais simplificadas.

Em (Alpaydin & Mayoraz, 1999) é apresentado um algoritmo que determina matrizes de códigos com a utilização de uma RNA do tipo MLP (Braga et al., 2000; Haykin, 1999). Essa RNA também define os parâmetros de classificadores lineares que são utilizados na solução dos subproblemas binários contidos na matriz. Por meio do treinamento da RNA,

otimiza-se conjuntamente os parâmetros dos classificadores lineares e a matriz de códigos utilizada na decomposição do problema multiclases.

Nos experimentos realizados por Alpaydin & Mayoraz (1999), procurou-se ajustar matrizes a oito conjuntos de dados do repositório UCI. A arquitetura e os pesos da RNA foram iniciados de acordo com o ECOC. As matrizes obtidas demonstraram resultados superiores aos dos ECOCs iniciais e, em alguns casos, também melhores que os das matrizes 1ct e tct.

Os conjuntos de dados empregados comuns aos do presente trabalho foram o *glass*, o *vowel*, o *sat-image*, o *opt-digits* e o *pen-digits*. Alpaydin & Mayoraz (1999) dividiram esses conjuntos com o método *2-fold cross-validation* cinco vezes. Embora não sejam diretamente comparáveis, as taxas de acerto verificadas foram similares ou inferiores às desta Tese, com o uso de um número maior de preditores binários que os encontrados pelos AGs na Seção 6.3. O desempenho inferior em relação às taxas de acerto pode ser devido ao fato de que a utilização de classificadores binários lineares na resolução dos subproblemas binários por Alpaydin & Mayoraz (1999) representa uma desvantagem em relação ao emprego de SVMs não lineares com um Kernel Gaussiano.

Dekel & Singer (2003) introduziram um algoritmo denominado *Bunching*, que adapta uma matriz de códigos para cada problema multiclases em seu processo de aprendizado. Esse algoritmo mapeia os dados de treinamento e os seus rótulos para um novo espaço. A matriz de mapeamento dos rótulos corresponde à matriz de códigos do problema. No novo espaço, procura-se minimizar a distância entre os dados e os seus rótulos. Como no trabalho de Alpaydin & Mayoraz (1999), o algoritmo de *Bunching* parte de uma matriz de códigos inicial, que é iterativamente adaptada ao problema.

Partindo de matrizes 1ct e aleatórias e utilizando classificadores de regressão logística (Collins et al., 2002), esses autores verificaram em geral ganhos de desempenho do *Bunching* quando aplicado sobre cinco conjuntos de dados do repositório UCI e um conjunto de dados denominado “*mnist*”. Embora hajam conjuntos de dados comuns aos aqui empregados, como o *glass*, o *sat-image* e o *vowel*, é difícil estabelecer uma comparação, uma vez que em (Dekel & Singer, 2003) são apresentadas apenas barras com a diferença relativa dos erros do *Bunching* em relação aos das matrizes 1ct e aleatórias.

Em (Rätsch et al., 2003) a adaptação de matrizes de códigos a cada aplicação multiclases foi formulada como um problema de otimização no qual os códigos, e também pesos para as funções binárias utilizadas no aprendizado das decomposições, são determinados em conjunto. Maximiza-se a diferença entre a distância do vetor de previsões dos classificadores binários em relação ao código da classe correta e a distância desse mesmo vetor ao segundo código mais próximo.

Em experimentos iniciais sobre os conjuntos de dados *iris* e *glass*, empregando ADs na geração dos preditores binários, verificou-se ganhos na taxa de acerto comparados aos obtidos com o uso das ADs diretamente na solução multiclases. Os conjuntos de dados foram divididos cinco vezes com *holdout* em partições de treinamento e de teste. As taxas

de acerto verificadas também foram próximas às desta Tese, apesar das partições dos dados e das técnicas de aprendizado utilizadas na indução dos preditores binários serem diferentes. Novamente, um número maior de classificadores binários foram utilizados nas soluções que os das matrizes determinadas pelo AG.

O algoritmo de Crammer & Singer (2002) (Seção 3.2.2), que reformula o treinamento das SVMs em uma versão multiclases, surgiu de um formalismo para a adaptação de matrizes de códigos aos problemas multiclases. As matrizes utilizadas possuem elementos contínuos, uma vez que determinar matrizes de códigos com elementos discretos foi considerado um problema NP-completo. Na Seção 6.6, as taxas de acerto médias das matrizes de códigos determinadas pelo AG lexicográfico foram comparadas às obtidas por esse algoritmo. Verificou-se que ambos obtiveram resultados estatisticamente similares nos doze conjuntos de dados testados.

Outro trabalho interessante na construção de matrizes de códigos é apresentado em (Pimenta, 2005). Nele é proposta uma função para a avaliação de ECOCs, considerando suas propriedades de correção de erros segundo as diretrizes de Dietterich & Bariki (1995). Pimenta (2005) introduz um algoritmo iterativo de perseguição, que adiciona ou retira colunas de um ECOc inicial, procurando maximizar a sua qualidade de acordo com a função definida. Também propõe um método para a determinação do número de preditores binários nos ECOCs, por meio do exame dos pontos de máximo local de uma função que considera o número de erros corrigíveis por ECOCs de diferentes tamanhos.

As matrizes obtidas por esses algoritmos foram avaliadas em seis conjuntos de dados do repositório UCI. Utilizando ADs na geração dos preditores binários, verificou-se ganhos em relação ao seu uso direto na solução multiclases e também sobre matrizes tct. Empregando SVMs na geração dos preditores, a matriz 1ct demonstrou os piores resultados, enquanto os ECOCs obtidos e a matriz tct tiveram desempenhos similares.

Os conjuntos de dados comuns aos desta Tese foram o *car*, o *glass*, o *sat-image*, o *opt-digits* e o *pen-digits*. Eles foram particionados pelos autores utilizando *10-fold cross-validation*. Nos conjuntos de dados *car* e *glass*, as taxas de acerto das matrizes do AG lexicográfico foram superiores. No conjunto *sat-image*, elas foram similares. Nos conjuntos *opt-digits* e *pen-digits*, os resultados de Pimenta (2005) foram melhores. Deve-se salientar, porém, que, além das partições dos dados, os parâmetros empregados para as SVMs nesses dois estudos também foram distintos, dificultando ainda mais a realização de uma comparação direta. Os tamanhos de matrizes determinados pelo método de investigação dos máximos locais da função de erros corrigíveis dos ECOCs foram maiores que os das matrizes do AG lexicográfico. Porém, em alguns casos, o algoritmo de perseguição foi capaz de determinar matrizes de bom desempenho mesmo com o uso do número mínimo de classificadores necessários para distinguir as classes dos problemas.

Entretanto, é importante destacar que os objetivos do algoritmo de perseguição e do AG lexicográfico são distintos. Enquanto o primeiro busca matrizes com boa propriedade de correção de erros, o segundo procura adaptar as matrizes de acordo com o seu de-

sempenho da solução multiclases quando SVMs são utilizadas na geração dos preditores binários. Pode ser interessante incorporar a função de avaliação de Pimenta (2005) aos AGs propostos nesta Tese, determinando assim matrizes com boa propriedade de correção de erros e adaptando-as também aos problemas de acordo com o desempenho obtido na solução multiclases.

► Determinação da Estrutura de DDAGs e ADAGs

Na Seção 3.1.2 foram descritos alguns algoritmos e heurísticas encontrados na literatura para a determinação da estrutura de DDAGs e ADAGs.

O primeiro realiza uma modificação no ADAG, levando ao RADAG (*Reordering ADAG*) (Phetkaew et al., 2003a,b). Em (Phetkaew et al., 2003b), os classificadores binários de cada nível do RADAG são definidos ordenando os pares de classes (i, j) tal que a soma dos $\|\mathbf{w}_{ij}\|$ seja minimizada. A cada classificação de um exemplo, os preditores binários nos níveis do RADAG vão sendo determinados de acordo com essa regra. Logo, o RADAG mantém a característica do ADAG de adaptar a estrutura do grafo de acordo com cada dado de teste, porém utiliza-se a informação dos $\|\mathbf{w}_{ij}\|$ nesse processo. Em (Phetkaew et al., 2003a) parecia-se as classes de acordo com a capacidade de generalização dos classificadores binários, estimada pelo limite de erro da Equação 2.6. Procura-se parear as classes de forma a minimizar a soma desse erro.

Em experimentos com sete conjuntos de dados do repositório UCI, os autores compararam os resultados do RADAG a ADAGs geradas aleatoriamente e ao tct com integração de votação por maioria. Realizou-se também um ajuste de parâmetros para as SVMs utilizando os Kernels Polinomial e Gaussiano. Embora o RADAG tenha apresentado em geral as maiores taxas de acerto, os resultados dos algoritmos testados foram próximos e não foram comparados estatisticamente.

Entre os conjuntos de dados empregados pelos autores, encontram-se o *glass*, o *sat-image*, o *segment* e o *vowel*. Os conjuntos *glass* e *segment* foram divididos por *5-fold cross-validation*. As taxas de acerto reportadas em (Phetkaew et al., 2003b) e (Phetkaew et al., 2003a) foram similares ou superiores às deste trabalho. Uma característica positiva do RADAG é a sua rapidez, uma vantagem grande sobre os AGs da Seção 6.4. A heurística também é aplicável a conjuntos de dados com quaisquer números de classes, enquanto o uso de AGs sobre números reduzidos de classes não é indicado, pois o espaço de busca nesses casos é muito pequeno.

Outro trabalho que propõe uma heurística para a determinação da estrutura dos DAGs é (Takahashi & Abe, 2003). Os autores sugerem que os classificadores binários com maior capacidade de generalização sejam colocados em níveis superiores do DDAG, segundo um limite de LOO de (Vapnik, 1995). Em (Feng et al., 2005), o mesmo tipo de heurística é utilizado na definição da estrutura de DDAGs, porém com o emprego do limite de erro de LOO de Jaakkola & Haussler (1999) na avaliação da capacidade de generalização dos classificadores binários.

Nos experimentos reportados nesses artigos, as taxas de acerto dos DDAGs determinados foram próximas às taxas de acerto médias de várias estruturas geradas aleatoriamente⁶, como nos resultados dos AGs na Seção 6.4. Porém, novamente, a heurística anterior tem a vantagem de ser menos custosa e de ser aplicável a problemas com números de classes arbitrários.

► Determinação da Estrutura de Árvores Multiclasses

Na determinação da estrutura de árvores multiclasses, Takahashi & Abe (2002) propõem que nós próximos à raiz particionem classes mais facilmente separáveis. Para medir a separabilidade, foram utilizadas a distância Euclidiana entre os centros das classes e informações dos dados incorretamente classificados por um preditor que os separa de acordo com a sua distância de Mahalanobis aos centros das classes. Em seus experimentos sobre dois conjuntos de dados, utilizando um Kernel Polinomial para as SVMs, os métodos propostos demonstraram menores taxas de erro do que a estratégia 1ct. Porém, não foi realizada uma comparação estatística entre eles.

Em (Schwenker, 2000) e (Schwenker & Palm, 2001), as partições binárias das classes nos nós das árvores foram definidas a partir da confusão entre elas. Para quantificar a confusão entre as classes, empregou-se o algoritmo de agrupamento k-médias. Em ambos os artigos são apresentados experimentos em apenas um conjunto de dados de dígitos, extraído do projeto Statlog (Michie et al., 1994). Os resultados da árvore multiclasses determinada utilizando SVMs na geração dos preditores binários foram similares aos das estratégias 1ct e tct.

Em (Vural & Dy, 2004) são sugeridos três métodos para a partição binária das classes nos nós de árvores multiclasses. O primeiro agrupa classes com centróides similares utilizando o algoritmo k-médias. O segundo divide as classes de acordo com a distância média de seus dados à origem. O terceiro divide as classes em subconjuntos tal que a diferença entre o número de dados neles seja mínima.

Nos experimentos realizados, Vural & Dy (2004) empregaram o terceiro método na obtenção das árvores. Elas foram avaliadas sobre conjuntos de dados reais e do repositório UCI (*glass*, *vowel* e *segment*). Os conjuntos de dados maiores foram divididos em três partes: uma para treinamento, uma para validação e outra para teste. Nos conjuntos de dados menores, utilizou-se o *10-fold cross-validation*. Realizou-se um procedimento de seleção de parâmetros para as SVMs. Testou-se o uso dos mesmos parâmetros para todas as SVMs binárias em uma estratégia multiclasses e também o efeito de escolher parâmetros diferenciados para cada preditor binário. A segunda alternativa mostrou os melhores resultados.

As taxas de acerto das árvores obtidas nos conjuntos de dados *glass* e *vowel* foram superiores às desta Tese, enquanto no conjunto de dados *segment* elas foram similares.

---

⁶Utilizando uma medida *fuzzy* na integração dos classificadores do DDAG, Feng et al. (2005) relataram ganhos de taxa de acerto.

Isso pode ser efeito do processo de seleção de parâmetros realizado. No caso do conjunto de dados *vowel*, o CV também influi no melhor desempenho. Em experimentos anteriores encontrados na literatura, como em (Fürnkranz, 2002), foram reportadas diferenças grandes entre a avaliação das taxas de acerto nesse conjunto de dados por meio do uso de CV daquelas calculadas em seu subconjunto de teste independente original.

Em (Lei & Govindaraju, 2005), as partições binárias das classes foram definidas por um algoritmo de agrupamento hierárquico. Utilizou-se a distância média entre os dados das classes no processo de agrupamento. Na avaliação experimental sobre quatro conjuntos de dados do repositório UCI, as taxas de acerto obtidas foram semelhantes às das estratégias 1ct, tct com votação por maioria e DDAG. Porém, verificou-se vantagens nos tempos de teste e em reduções no tamanho final dos classificadores, medido em KBytes.

Dois dos conjuntos de dados empregados foram o *iris* e o *pen-digits*. O conjunto de dados *iris* foi dividido em 70% para treinamento e 30% para teste. Realizou-se ainda um procedimento de seleção de parâmetros para as SVMs, em um subconjunto de validação definido a partir dos dados de treinamento. No conjunto *iris*, todas as estratégias obtiveram uma taxa de acerto de 100%. Porém, o subconjunto de teste formado é muito pequeno para permitir uma avaliação rigorosa. As taxas de acerto no conjunto de dados *pen-digits* foram um pouco superiores às desta Tese, o que pode ser efeito do ajuste de parâmetros para as SVMs.

Frank & Kramer (2004) propõem que diversas árvores multiclases sejam combinadas em um comitê, como alternativa a determinar uma única estrutura. Em experimentos sobre 21 conjuntos de dados do repositório UCI, como o *iris*, o *glass*, o *segment*, o *opt-digits*, o *pen-digits* e o *vowel*, os autores demonstram a viabilidade de sua proposta em relação ao uso das estratégias 1ct, tct, ECOC e da solução direta do problema multiclases. Empregou-se ADs e classificadores de regressão logística na indução dos preditores binários. Adotou-se uma abordagem probabilística na integração das saídas dos classificadores binários nas árvores multiclases. Logo, o processo de reconstrução é probabilístico, sendo diferente do empregado nesta Tese. Os conjuntos de dados foram divididos com *holdout* 50 vezes, em 66% para treinamento e o restante para teste. Embora não sejam diretamente comparáveis, os resultados verificados foram em alguns casos semelhantes e em outros superiores ou inferiores aos das árvores determinadas nesta Tese.

6.7.3 Efeitos de Variações nos Parâmetros das SVMs

A representação dos indivíduos nos AGs para a determinação de matrizes de códigos da Seção 4.2.1 pode ser estendida para incorporar a busca de parâmetros diferenciados para cada SVM binária induzida. Ajusta-se assim parâmetros individuais para as SVMs contidas em uma decomposição.

Limitou-se essa seleção ao Kernel do tipo Gaussiano. Este é o Kernel mais complexo entre os apresentados na Tabela 2.1, sendo capaz de representar espaços de características de dimensão infinita (Herbrich, 2001). Buscou-se então os parâmetros C de regularização

das SVMs e σ do Kernel Gaussiano.

Para implementar esse procedimento de busca, adicionou-se a cada coluna dos indivíduos bits que codificam os parâmetros considerados. Para decodificar a informação nesses bits, optou-se pelo uso de um esquema de potências, como sugerido em (Fröhlich et al., 2003). Os possíveis valores decimais representados por b bits fornecem potências, que elevadas a uma base especificada pelo usuário, provêm os valores de parâmetros a serem testados. Pode-se também somar uma constante aos números decimais codificados pelos bits. Assim, dado um dos parâmetros, seja d o valor decimal dos bits que o representam. Soma-se uma constante a a este número, que se torna $d + a$. O resultado dessa soma é usado como expoente de uma base b , dando origem ao valor real do parâmetro, ou seja, b^{d+a} . Considerando, por exemplo, dois bits para C , pode-se representar os valores $C = [2^0, 2^1, 2^2, 2^3]$ ou $C = [10^0, 10^1, 10^2, 10^3]$ ou mesmo $C = [10^{-3}, 10^{-2}, 10^{-1}, 10^0]$, sendo que no último caso soma-se -3 aos números decimais codificados pelos dois bits. Na Figura 6.9 é apresentado um exemplo de indivíduo com essa extensão, em que dois bits são utilizados para representar cada parâmetro.

Figura 6.9: Indivíduo de AG com extensão para determinar os parâmetros das SVMs

Seguindo esse esquema de representação de indivíduos, cada classificador binário de uma matriz pode ter valores de parâmetros diferenciados. Para não dificultar a implementação do algoritmo, optou-se por manter os mesmos operadores genéticos descritos na Seção 4.2.3. Deve-se salientar que a inserção dos parâmetros das SVMs aos indivíduos aumenta ainda mais o espaço de busca do AG. Com isso, ele se torna mais oneroso computacionalmente.

Testou-se o efeito da aplicação dessa modificação no AG lexicográfico sobre quatro dos conjuntos de dados da Tabela 6.1: *car*, *lung*, *segment* e *fungi*. Esses foram os conjuntos em que o AG lexicográfico na Seção 6.3 encontrou matrizes de códigos com taxas médias de acerto maiores que as da estratégia 1ct. O objetivo é avaliar se a busca conjunta das matrizes de códigos e dos parâmetros das SVMs pode melhorar ainda mais os resultados alcançados.

Os parâmetros procurados, a constante de regularização C e o valor de σ no Kernel

Conjunto	Taxa de acerto		#Classificadores	
	SSel	CSel	SSel	CSel
<i>car</i>	97,8 (1,4)	99,3 (0,9)	3,6 (0,5)	3,6 (0,5)
<i>lung</i>	85,5 (4,8)	90,2 (6,1)	3,6 (0,7)	3,5 (0,6)
<i>segment</i>	96,2 (1,1)	96,7 (1,1)	6,5 (0,7)	6,8 (0,4)
<i>fungi</i>	62,0 (2,6)	65,1 (2,7)	7,7 (1,2)	7,7 (1,2)

Tabela 6.14: Resultados de seleção de parâmetros para as SVMs em AG lexicográfico

Gaussiano, foram representados por dois bits cada. Tem-se então quatro bits adicionais por coluna nas matrizes manipuladas pelos AGs. Com isso, os indivíduos passam a ser representados por matrizes de dimensão $(k + 4) \times l$, em que l é variado. Para decodificar a informação nesses bits, empregou-se o procedimento de potências descrito anteriormente. Para a determinação de C , adotou-se $a = 0$ e $b = 10$. Logo, os possíveis valores para C são: $C = [10^0, 10^1, 10^2, 10^3]$. No caso de σ , empregou-se $a = -3$ e $b = 10$, fornecendo o seguinte conjunto de valores: $\sigma = [10^{-3}, 10^{-2}, 10^{-1}, 10^0]$. Na prática há então 16 combinações de (C, σ) para cada classificador binário em uma matriz.

Para simplificar os experimentos, os mesmos parâmetros da Seção 6.3 foram utilizados na execução dos AGs. Porém, seria adequado aumentar o tamanho da população e o número máximo de ciclos do AG, uma vez que o espaço de busca foi ampliado com a introdução desses bits adicionais aos indivíduos.

Na Tabela 6.14 são apresentadas as taxas de acerto obtidas após a introdução da seleção de parâmetros para as SVMs ao AG lexicográfico. Também encontram-se ilustrados os números de classificadores binários das matrizes finais obtidas. Para facilitar as comparações de resultados com aqueles da Seção 6.3, em que esse ajuste de parâmetros não foi realizado, esta tabela também reproduz os resultados do AG anterior. As colunas “SSel” indicam as soluções sem o procedimento de ajuste de parâmetros para as SVMs, enquanto a coluna “CSel” denota o uso da seleção de parâmetros. Como na Seção 6.3, os AGs foram executados 30 vezes para cada partição de treinamento/teste dos conjuntos de dados e fornece-se então a média de seus resultados.

Em todos os casos foi possível obter melhoras nas taxas de acerto. No conjunto de dados *lung*, porém, o desvio-padrão dessa taxa aumentou, indicando maiores variações entre as diferentes soluções do AG. Os números de classificadores binários nas matrizes determinadas por ambas as versões de AG mantiveram-se próximos. Embora os experimentos realizados tenham sido simplificados, os resultados são encorajadores, demonstrando que os AGs também podem ser adequados à busca dos parâmetros nas SVMs binárias contidas nas decomposições.

Esse tipo de seleção de parâmetros também pode ser incorporado aos AGs para a determinação da estrutura de DAGs. Entretanto, deve-se realizar modificações na forma de representação dos indivíduos e nos operadores genéticos. Na Figura 6.10 ilustra-se como a mudança de parâmetros das SVMs afeta o desempenho dos DDAGs e ADAGs, criados aleatoriamente, nos cinco conjuntos de dados em que os AGs foram empregados

na Seção 6.4.

Figura 6.10: Efeito de variação de parâmetros das SVMs em DDAGs e ADAGs

Manteve-se a constante de regularização com um valor $C = 100$. Variou-se apenas o valor de σ no Kernel Gaussiano, entre $[10^{-3}, 10^{-2}, 10^{-1}, 10^0]$. Utilizou-se os mesmos parâmetros em todas as SVMs binárias nos DAGs. Tem-se então um cenário mais simples que o apresentado anteriormente, em que determinava-se parâmetros diferenciados para cada SVM binária.

Nos gráficos, tem-se as taxas de acerto dos DDAGs e ADAGs para cada valor de σ testado. Foram geradas 30 estruturas aleatórias para cada conjunto de dados. Plotou-se então as taxas médias de acerto de todas as soluções criadas. Para cada conjunto de dados, circulou-se os valores de σ que levaram às maiores taxas de acerto.

Os gráficos demonstram que a variação dos parâmetros das SVMs afeta as taxas de acerto dos DDAGs e dos ADAGs. Para cada conjunto de dados, há valores de parâmetros ideais distintos. Logo, a busca dos parâmetros das SVMs em conjunto com as permutações das classes nos AGs da Seção 6.4 pode introduzir maiores variabilidades entre diferentes soluções, beneficiando o uso dos AGs.

Outra informação interessante nos gráficos da Figura 6.10 é que, mesmo para valores de parâmetros distintos, as taxas de acerto dos DDAGs e dos ADAGs se mantiveram muito próximas, sendo em diversos casos idênticas.

É importante salientar que a incorporação dos parâmetros das SVMs aos AGs onera ainda mais estes algoritmos. O espaço de busca aumenta e, conseqüentemente, torna-se adequado empregar tamanhos maiores de população e também um número maior de ciclos.

Deve-se ainda destacar que o ajuste de parâmetros para as SVMs melhora o desempenho de todas as estratégias decomposicionais, conforme observado em (Rifkin & Klau-

tau, 2004; Passerini et al., 2004; Passerini, 2004; Vural & Dy, 2004). Apenas em (Vural & Dy, 2004) selecionou-se valores de parâmetros diferenciados para as SVMs binárias, o que foi realizado empiricamente. Para cada SVM binária, escolheu-se a combinação de parâmetros com menor estimativa de erro de LOO. Logo, no procedimento adotado por esses autores, os parâmetros das SVMs foram ajustados de acordo com o seus erros nos subproblemas binários, enquanto no presente trabalho procurou-se minimizar o erro conjunto de todas elas na solução multiclases.

6.7.4 Outros Resultados

Conforme discutido na Seção 6.3, as matrizes de códigos determinadas pelo AG lexicográfico no conjunto de dados *structural* foram próximas à 1ct. Investigou-se então o efeito de aumentar o número máximo de classificadores permitidos nas matrizes de códigos manipuladas por esse AG. Variou-se este número de 4 a 10 e, para cada valor, os AGs foram executados 30 vezes. Por simplicidade, empregou-se novamente os mesmos parâmetros listados na Seção 6.3 para os AGs, embora com o crescimento do tamanho dos indivíduos seja aconselhável aumentar o tamanho da população e o número máximo de ciclos.

Os resultados foram plotados graficamente na Figura 6.11. No gráfico da Figura 6.11a tem-se as taxas de acerto médias das soluções obtidas para cada valor de máximo de classificadores permitidos testado. Na Figura 6.11b são ilustrados os números médios de classificadores das matrizes obtidas pelo AG em cada caso. Circulou-se nos dois gráficos o ponto em que as maiores taxas de acerto foram verificadas.

Figura 6.11: Efeito de variação do número máximo de classificadores no AG lexicográfico

Com o aumento do número máximo de classificadores binários permitidos nas matrizes manipuladas pelo AG lexicográfico, houve pequenas melhoras nas taxas de acerto médias das matrizes obtidas. Esse crescimento se manteve até um valor máximo de nove classificadores binários. Nesse ponto, as matrizes determinadas possuem na média 5,8 classificadores binários. Para o conjunto de dados *structural* pode então ser adequado procurar soluções que utilizem mais classificadores binários que a matriz 1ct. Esse fato também foi comprovado pelo desempenho um pouco inferior das árvores multiclases nesse

conjunto de dados, as quais empregam três preditores binários.

Realizou-se o mesmo procedimento anterior no conjunto de dados *sat-image*, em que a matriz 1ct também se sobressaiu. Porém, não foram verificados aumentos da taxa de acerto neste caso, que se mantiveram iguais.

Embora nos experimentos apresentados nas seções 6.3 a 6.6 as taxas de acerto totais obtidas por diferentes estratégias multiclases tenham apresentado poucas diferenças, houve variações no desempenho observado no reconhecimento das classes. Uma exceção pode ser apontada entre as estratégias DDAG e ADAG, em que o desempenho total e também nas classes se manteve similar.

De maneira geral, cada estratégia tende a enfatizar algumas classes. Essa característica pode ser explorada na escolha de uma técnica particular em domínios em que algumas classes sejam consideradas mais importantes do que outras. Esses resultados também demonstram um potencial para a combinação de diferentes estratégias multiclases em comitês. Dessa forma, é possível explorar complementariedades encontradas no reconhecimento das classes.

As tabelas contendo as taxas de acerto obtidas nas classes dos conjuntos de dados da Seção 6.1 pelas principais estratégias multiclases testadas neste trabalho são apresentadas na Seção B.5 do Apêndice B. Apesar de haver diferenças entre as diversas taxas reportadas, é difícil estabelecer uma tendência geral nos resultados observados.

Em uma análise simplificada, verificou-se que as estratégias que mais se sobressaíram foram a 1ct e as árvores multiclases. Porém, essas árvores também obtiveram vários resultados negativos. Logo, elas tendem a favorecer algumas classes mais fortemente, em prejuízo de outras. Curiosamente, o AG lexicográfico em alguns conjuntos de dados desbalanceados tendeu a aumentar as taxas de acerto em classes com menos dados, reduzindo o desempenho em classes maiores. No conjunto de dados *lung*, por exemplo, as matrizes obtidas diminuíram as taxas de acerto no reconhecimento da classe majoritária, porém aumentaram significativamente as taxas de acerto nas outras classes. Fato semelhante ocorreu no conjunto de dados *structural* em relação à classe $\alpha + \beta$.

Há alguns pontos interessantes no desempenho nas classes em alguns dos conjuntos de dados. No conjunto *fungi* (Tabela B.23), muitas das estratégias testadas apresentaram taxas de acerto nulas em várias classes. As matrizes do AG lexicográfico foram capazes de apresentar taxas não nulas em diversos desses casos, embora elas tenham se mantido pequenas.

No conjunto de dados *vowel* (Tabela B.28), as matrizes do AG lexicográfico manifestaram grandes variações de desempenho nas classes, apresentando valores altos de desvio-padrão. As taxas de acerto totais das soluções obtidas por este AG nesse conjunto de dados também apresentaram maiores variações que as verificadas em outros casos, conforme discutido na Seção 6.3.

No conjunto de dados *lung*, as estratégias tct, DDAG e ADAG apresentaram taxas de acerto nulas em duas classes com poucos dados. Como mencionado na Seção 6.7.1,

classificadores do tipo tct podem encontrar dificuldades no caso em que as classes apresentam poucos exemplos. As árvores multiclases também apresentaram essa deficiência neste conjunto de dados.

Também houve taxas de acerto nulas em uma das classes do conjunto de dados *glass*, por parte das estratégias 1ct e ECOC. A classe em questão apresenta uma proporção pequena dos dados em relação às classes majoritárias. Essa característica pode ter afetado negativamente a indução do classificador que separa a classe considerada das demais na decomposição 1ct.

É possível apontar diferenças entre as estratégias multiclases também em termos do tempo necessário para o treinamento de seus classificadores e em seus tempos de teste, tomados na previsão da classe de novos exemplos. Uma discussão a esse respeito é realizada em (Hsu & Lin, 2002).

No presente trabalho, em algumas análises simplificadas verificou-se que, na etapa de teste, as estratégias decomposicionais hierárquicas foram as mais rápidas. Os algoritmos WW e CS também foram rápidos nessa etapa. A técnica mais lenta foi a ECOC, que apresenta o maior número de preditores binários entre as estratégias decomposicionais testadas. No treinamento, o algoritmo CS e o ECOC apresentaram os maiores tempos. Em contrapartida, a decomposição tct, que inclui o DDAG, foi rápida nessa etapa. Como ela envolve classificadores para pares de classes, emprega-se na indução de cada um deles apenas os dados de duas classes. Com isso, o seu treinamento é acelerado.

Diversos outros experimentos foram realizados no decorrer desta Tese. Optou-se por apresentar apenas aqueles considerados mais relevantes para a demonstração das técnicas propostas. Outros resultados podem ser consultados em artigos publicados no período, como (Lorena & Carvalho, 2003c, 2004a,c,d, 2005a,b; Ushizima et al., 2005; Lorena & Carvalho, 2006a,b).

6.7.5 Resumo e Análise Crítica

Os algoritmos e técnicas propostos nesta Tese tiveram por objetivo investigar a decomposição de problemas multiclases em subproblemas binários de acordo com cada aplicação particular, a partir de informações de seus dados ou do desempenho de SVMs em sua solução.

No início, investigou-se a determinação dos classificadores binários em matrizes de códigos. Utilizou-se AGs na resolução desse problema, buscando decomposições que minimizassem o erro calculado em subconjuntos de validação e o número de classificadores binários necessários. Com isso, procura-se matrizes adaptadas à solução dos problemas multiclases e também privilegia-se a sua simplicidade, por meio do controle do número de preditores binários utilizados. Este último quesito está em conformidade com o princípio de Occam, que afirma que, entre diversas hipóteses corretas, a mais simples deve ser selecionada. As matrizes procuradas também não deveriam apresentar colunas equivalentes, o que representaria o uso de classificadores binários iguais em uma mesma decomposição.

Com base nos objetivos anteriormente descritos, foram implementados dois tipos de AGs. O primeiro, denominado AG lexicográfico, impõe uma ordem nos objetivos, privilegiando a minimização do erro de validação em relação à minimização do número de classificadores binários e penalizando a ocorrência de colunas equivalentes em uma matriz. O segundo, SPEA2 (Zitzler et al., 2002), considera todos os objetivos em conjunto, por meio do uso do conceito de dominância de Pareto.

Os resultados experimentais evidenciaram um melhor desempenho do AG lexicográfico, que foi capaz de determinar matrizes de códigos com taxas de acerto similares ou superiores às da estratégia 1ct, largamente empregada na prática, com menos classificadores binários. As matrizes obtidas em geral também demonstraram resultados similares aos de outras estratégias baseadas em matrizes de códigos que empregam números maiores de preditores binários, a tct e a ECOC.

O AG lexicográfico ainda obteve bons resultados na adaptação dos parâmetros das SVMs binárias contidas nas decomposições geradas em testes preliminares realizados na Seção 6.7.3.

O segundo problema abordado neste trabalho foi a determinação da estrutura de DDAGs e ADAGs. Novamente, procurou-se adaptar essas estruturas aos problemas multiclases, minimizando o seu erro em subconjuntos de validação. Na prática, porém, verificou-se poucas variações de desempenho entre estruturas distintas quando os parâmetros de todas as SVMs envolvidas são iguais. Como um dos princípios para o bom desempenho e adequabilidade dos AGs é a presença de variabilidade entre as soluções (Seção 4.1), seu uso, nesse caso, pode não ser a melhor alternativa.

Uma vez que os parâmetros das SVMs exercem influência em seu desempenho, além de serem dependentes de cada aplicação, o uso dos AGs na busca conjunta das estruturas dos DAGs e dos parâmetros das SVMs, como realizado no caso das matrizes de códigos, pode ser uma alternativa mais interessante. Os AGs também podem ser mais vantajosos na determinação da estrutura de DAGs para aplicações com mais classes que as investigadas neste trabalho.

Estudou-se ainda como informações dos dados poderiam ser utilizadas na definição da estrutura de árvores multiclases. Para tal, os algoritmos do Capítulo 5, que agrupam hierarquicamente as classes, foram utilizados. As informações das classes utilizadas na realização desse agrupamento foram extraídas por dois critérios: o primeiro considera a distância entre os centróides das classes e o segundo utiliza o grau de confusão entre elas, segundo o desempenho de ADs em sua separação.

Os resultados obtidos indicam que a escolha de um dos critérios pode ser dependente das características do conjunto de dados considerado. De forma geral também observou-se pouca variação nas taxas de acerto de árvores multiclases distintas quando as SVMs são utilizadas na indução dos preditores binários. Porém, ela foi maior que a encontrada nos experimentos com os DDAGs e ADAGs.

Comparou-se ainda as taxas de acerto das matrizes do AG lexicográfico, dos DDAGs,

das árvores multiclases e dos algoritmos de Weston & Watkins (1998) e de Crammer & Singer (2002), que realizam reformulações no treinamento das SVMs binárias em versões multiclases. Os resultados, embora geralmente similares, foram um pouco mais favoráveis às matrizes do AG lexicográfico e ao algoritmo de Crammer & Singer (2002). É interessante observar que este algoritmo também parte do princípio de adequar matrizes de códigos às soluções multiclases, porém elas possuem elementos contínuos. Logo, as adaptações realizadas pelo AG lexicográfico em matrizes discretas foram compatíveis às realizadas por essa abordagem.

Os estudos com o AG lexicográfico e com as árvores multiclases demonstraram a viabilidade de se adaptar as decomposições à solução de cada problema multiclases particular quando as SVMs são utilizadas como preditores binários. Entretanto, no caso dos AGs para a obtenção de matrizes de códigos, claramente tem-se um custo computacional maior nessa adaptação, principalmente para problemas com números maiores de classes. Essa deficiência deve ser levada em consideração, uma vez que o desempenho das matrizes obtidas foi similar ao de outras estratégias mais simples. Como os AGs são facilmente paralelizáveis, essa dificuldade pode ser minimizada.

Outro ponto que deve ser levado em consideração decorre da influência dos valores assumidos pelos parâmetros livres dos AGs no seu desempenho. Nos experimentos realizados, diversos parâmetros apresentaram valores comuns para os diferentes conjuntos de dados. Variações destes valores podem produzir melhores resultados.

Da mesma forma, os parâmetros das SVMs binárias influem no desempenho final das estratégias multiclases testadas. Ajustar esses parâmetros para cada classificador binário é um processo custoso, principalmente com o crescimento do número de classes. Uma alternativa promissora pode ser utilizar os AGs na realização dessa tarefa, como investigado preliminarmente na Seção 6.7.3 e em (Xu & Chan, 2003).

Deve-se salientar também que o objetivo principal deste estudo não foi comparar as diversas estratégias existentes para a generalização das SVMs a problemas multiclases para então recomendar o uso de alguma delas. Um estudo desse tipo necessita de experimentos e análises mais criteriosos, com a realização de ajustes de parâmetros para as SVMs em cada estratégia, por exemplo. Existe na literatura trabalhos que realizam esse tipo de análise, como (Hsu & Lin, 2002; Rifkin & Klautau, 2004; Duan & Keerthi, 2005). Nenhum deles, porém, ajustou parâmetros diferenciados para cada SVM binária, por se tratar de um processo custoso.

Apesar dessas limitações, algumas observações interessantes podem ser apontadas quanto aos resultados das diferentes estratégias apresentados nesse capítulo:

- A estratégia 1ct apresentou boas taxas de acerto, principalmente nos conjuntos de dados *sat-image*, *opt-digits* e *pen-digits*. Todos eles se caracterizam por serem balanceados. Esses foram também os conjuntos de dados em que o AG lexicográfico apresentou as menores reduções nas matrizes de códigos. Nos conjuntos desbalan-

ceados *car*, *lung* e *fungi*, porém, a adaptação das matrizes de códigos com o AG lexicográfico foi capaz de aumentar as taxas de acerto obtidas em relação às da matriz 1ct. De fato, o desbalanceamento na proporção de dados nas classes pode ser um problema para a estratégia 1ct. Como na indução de cada preditor utiliza-se os dados de apenas uma das classes contra os de todas as outras, pode-se aumentar o efeito negativo do desbalanceamento, prejudicando o reconhecimento de classes minoritárias.

- A estratégia tct pode enfrentar dificuldades quando confrontada com conjuntos de dados que possuam poucos exemplos, como o de diagnóstico de câncer de pulmão. Seus preditores nesse caso passam a ser mais sujeitos a super-ajustamentos (*overfitting*). Outra deficiência da decomposição tct é que, na classificação de um exemplo, muitos dos classificadores induzidos são irrelevantes. Isso ocorre porque muitos dos preditores não envolvem a classe correta do exemplo. Eles então realizam previsões errôneas. Na prática, isso representa a adição de ruídos ao processo de classificação.
- O ECOC apresenta boas taxas de acerto, devido à sua capacidade de correção de erros cometidos por alguns classificadores binários. Porém, além dessa estratégia em geral requerer o uso de mais classificadores binários, as partições binárias definidas podem ser muito complexas, implicando em tempos maiores de treinamento e de teste.
- As taxas de acerto dos DDAGs e dos ADAGs nos experimentos realizados podem ser consideradas idênticas. Este resultado se manteve mesmo para o conjunto de dados *vowel*, com o maior número de classes entre os testados. Não se verificou então o benefício do ADAG na redução de erros cumulativos do DDAG. Entretanto, é necessário testar essas observações em conjuntos de dados com mais classes. Também não foram observadas diferenças grandes de taxa de acerto entre estruturas distintas desses dois grafos. Logo, esse pode não ser um problema crítico em seu uso com SVMs.
- Também no caso das árvores multiclases não houve variações muito grandes entre as taxas de acerto de estruturas distintas quando as SVMs são utilizadas na indução dos preditores binários. Entretanto, elas foram maiores que as encontradas nos experimentos com os DDAGs e os ADAGs.
- As estratégias diretas de Weston & Watkins (1998) e de Crammer & Singer (2002) também demonstraram bom desempenho na solução de diversas aplicações multiclases testadas neste trabalho. No entanto, o primeiro algoritmo apresentou quedas de taxa de acerto em relação a outras estratégias em alguns conjuntos de dados, enquanto o segundo apresentou um desempenho mais estável.

- Com relação às estratégias decomposicionais, as árvores multiclases apresentaram, em geral, o menor número de preditores binários ($k - 1$ para um problema com k classes). Em seguida tem-se as matrizes de códigos do AG lexicográfico, que em alguns conjuntos de dados apresentaram números médios de classificadores menores até mesmo que os das árvores anteriormente mencionadas. A matriz 1ct figura em seqüência, com k preditores, embora algumas das matrizes do AG lexicográfico também tenham apresentado esse número de classificadores. Em seguida na ordem crescente, tem-se a decomposição tct, que inclui o DDAG e o ADAG, com $\frac{k(k-1)}{2}$ preditores, e a matriz ECOC. Esta última foi construída por um método exaustivo e possui o maior número de classificadores binários, $2^{k-1} - 1$.

6.8 Considerações Finais

Neste capítulo analisou-se experimentalmente as técnicas e algoritmos propostos nesta Tese. O ponto comum entre elas é a adaptação das decomposições binárias dos problemas multiclases, por meio da consideração de seu desempenho ou com o uso de informações extraídas dos dados.

Inicialmente investigou-se o uso de AGs na determinação de matrizes de códigos para cada problema multiclases. Procurou-se matrizes com taxas de acerto similares ou superiores às da estratégia 1ct, mais empregada na prática, que necessitassem de menos classificadores binários. A versão de AG lexicográfica foi capaz de solucionar esse problema, a partir da adaptação das matrizes à solução de cada aplicação particular com SVMs. Pode ser interessante futuramente estender esse AG para considerar propriedades de correção de erros na definição dos classificadores binários nas matrizes de códigos, com o intuito de aumentar as suas taxas de acerto. Uma alternativa para tal seria incorporar a função de avaliação de ECOCs de Pimenta (2005) aos AGs.

Investigou-se ainda a influência da organização dos preditores binários nas estruturas hierárquicas DDAG, ADAG e de árvores multiclases em seu desempenho. Verificou-se poucas variações de taxa de acerto quando as SVMs são empregadas na indução dos preditores binários, principalmente no caso dos DDAGs e dos ADAGs.

O uso de AGs na determinação da estrutura dos DDAGs e ADAGs pode então não ser a alternativa mais adequada. São necessárias, no entanto, análises em conjuntos de dados com mais classes. Os AGs também podem ser estendidos de forma a buscar os parâmetros das SVMs binárias nessas estruturas, situação em que a variabilidade das soluções aumenta.

Os nós das árvores multiclases foram definidos por meio de um procedimento de agrupamento hierárquico das classes. Testou-se o uso da distância entre os centróides das classes e da confusão entre elas segundo ADs na quantificação de sua relação. Em alguns conjuntos de dados foi mais adequado empregar um critério do que o outro na extração das informações dos dados. Um estudo interessante nesse caso pode ser verificar

quais características e informações dos dados podem ser consideradas mais adequadas na definição dessas partições das classes em cada conjunto de dados, por meio do uso de meta-aprendizado (Giraud-Carrier et al., 2004).

Os experimentos realizados também podem ser aprimorados por meio do ajuste de parâmetros para as SVMs e para os AGs. Com isso, podem ser verificadas melhoras de resultados para todas as estratégias multiclases testadas. Os algoritmos codificados também não foram testados em toda a sua extensão. O estudo realizado abre assim diversas possibilidades de testes e extensões, que não foram realizados devido a limitações de tempo e serão propostos como trabalhos futuros.

Deve-se destacar ainda que as análises realizadas neste capítulo se aplicam às SVMs. Isso ocorre principalmente em relação aos resultados com os AGs, uma vez que as soluções manipuladas por eles foram adaptadas de acordo com o desempenho obtido em subconjuntos de validação, utilizando SVMs na indução dos preditores binários. O uso das mesmas estratégias com outros algoritmos de AM pode então levar a resultados diferentes dos reportados.

Capítulo 7

Conclusão

Esta Tese apresentou um estudo acerca da solução de problemas de classificação multiclases com o uso de uma técnica de AM originalmente binária, as SVMs. Embora o algoritmo de treinamento das SVMs possa ser reformulado em versões multiclases, o enfoque da pesquisa conduzida foi sobre estratégias que decompõem o problema multiclases em múltiplos subproblemas binários, cujas saídas são combinadas na previsão multiclases.

Entre as estratégias decomposicionais apresentadas, três problemas foram identificados e investigados. Inicialmente verificou-se como informações do desempenho de decomposições representadas por matrizes de códigos poderiam ser utilizadas em sua adaptação a cada problema multiclases particular, por meio do uso de AGs. A seguir analisou-se a influência da organização dos preditores em DDAGs e ADAGs, dois tipos de estruturas hierárquicas, em seu desempenho na solução multiclases. Novamente os AGs foram empregados para adaptar esses grafos de acordo com o seu desempenho em cada aplicação multiclases. O terceiro problema abordado foi a definição da estrutura de árvores multiclases, outro tipo de estratégia que organiza os preditores binários hierarquicamente. Ele foi solucionado com a introdução de um algoritmo que agrupa as classes hierarquicamente de acordo com informações extraídas dos dados de cada aplicação multiclases.

O ponto comum entre as propostas mencionadas está na definição de decomposições adequadas a cada problema multiclases, utilizando para isso informações do desempenho obtido em sua solução ou coletadas a partir de seus dados.

Os algoritmos introduzidos foram avaliados experimentalmente sobre conjuntos de dados gerais da área de AM e em aplicações reais da área de Bioinformática. As análises realizadas se concentraram no desempenho obtido na solução dos problemas multiclases, medido em termos de taxas de acerto totais em dados de teste, e também no número de preditores binários empregados nas decomposições.

A seguir, na Seção 7.1, os principais resultados verificados nesses experimentos são discutidos. As contribuições do trabalho realizado são apresentadas na Seção 7.2. Mais adiante, na Seção 7.3, são discutidas limitações das estratégias e técnicas propostas e investigadas. A partir das limitações delimitadas, na Seção 7.4 são definidos possíveis trabalhos futuros advindos da pesquisa realizada. Algumas considerações finais são então proferidas na Seção 7.5.

7.1 Principais Resultados

Como primeiro resultado do trabalho realizado tem-se a proposição e a análise de novos algoritmos para a solução de problemas multiclasse com SVMs binárias. Esses algoritmos cobrem três problemas identificados entre as estratégias decompositivas comumente empregadas na prática.

O primeiro relaciona-se às estratégias baseadas em matrizes de códigos tradicionais, como a *1ct* e a *tct*, as quais decompõem o problema multiclasse a priori, sem considerar as suas características e propriedades. Como mencionado anteriormente, investigou-se o uso de AGs na solução dessa limitação, os quais adaptam matrizes de códigos a cada aplicação multiclasse por meio da consideração do desempenho obtido em sua solução.

Duas versões de AGs foram implementadas. Ambas manipulam matrizes de códigos, procurando minimizar o seu erro na solução multiclasse e, ao mesmo tempo, reduzir o número de preditores binários contidos nas mesmas. Além disso, procurou-se evitar a presença de classificadores idênticos em uma mesma matriz de códigos, fato que representaria o uso de preditores iguais em uma mesma decomposição. O primeiro AG, denominado lexicográfico, impõe uma ordem nos dois primeiros objetivos, privilegiando a minimização do erro sobre a do número de preditores binários. Evitar a ocorrência de preditores binários idênticos foi considerada uma restrição do problema. O segundo AG, denominado SPEA2, lida com os três objetivos descritos simultaneamente, por meio do uso do conceito de dominância de Pareto.

Experimentalmente, o AG lexicográfico se mostrou mais adequado. Os principais benefícios verificados na aplicação desse AG se deram na obtenção de decomposições com desempenho comparável ao de estratégias tradicionais, porém mais simples que elas, por necessitar de menos classificadores binários.

Experimentos preliminares empregando esses mesmos AGs para determinar também parâmetros diferenciados para cada SVM binária em uma decomposição apresentaram resultados animadores. A busca por valores de parâmetros diferenciados para cada SVM é um problema difícil de ser solucionado empiricamente. Por esse motivo, o uso de um procedimento de busca mais sofisticado, que realiza essa procura de forma automática, pode ser uma alternativa promissora. Apesar dos bons resultados iniciais, experimentos adicionais são necessários. Além disso, seria interessante projetar um novo AG para a seleção de parâmetros, utilizando, por exemplo, uma representação por números reais para as soluções. Dessa forma a busca não fica limitada a valores discretos, como realizado

nesta Tese.

O segundo problema investigado neste trabalho foi a influência da organização dos preditores binários em DDAGs e ADAGs no desempenho obtido por eles na solução multiclases. Esses dois grafos organizam hierarquicamente preditores que diferenciam todos os pares de classes. Foi proposto e investigado um AG que adapta a ordem dos preditores binários nesses grafos de forma a minimizar o erro obtido na solução multiclases. O problema foi formulado como uma busca por permutações de classes e foi então solucionado a partir de analogias com o uso de AGs na solução de problemas de permutação, como o do caixeiro viajante.

Experimentalmente, esses AGs obtiveram soluções com desempenho similar ao obtido por estruturas determinadas aleatoriamente, as quais demonstraram poucas variações de resultados. Isso indica que a ordem dos classificadores binários nos DAGs não altera muito o seu desempenho, não sendo, portanto, um aspecto importante em seu uso. Contudo, alterações nos parâmetros das SVMs nesses grafos em geral provocam modificações maiores em seus resultados. Logo, uma alternativa promissora nesse caso pode ser utilizar os AGs para determinar parâmetros diferenciados para cada SVM binária contida nesses grafos.

Outro problema semelhante ao anteriormente abordado foi o da definição da estrutura de árvores multiclases, que também organizam preditores binários de maneira hierárquica. Entre as principais estratégias multiclases, essas árvores utilizam o menor número de classificadores binários, sendo $k - 1$ para um problema com k classes.

No estudo conduzido, realizou-se adaptações no algoritmo de Kruskal, o qual obtém árvores geradoras mínimas a partir de grafos ponderados. Essas alterações permitiram que os preditores binários a serem contidos nas árvores multiclases fossem determinados automaticamente a partir de informações coletadas dos dados de cada aplicação multiclases. Na prática, realiza-se um agrupamento hierárquico das classes, o qual é definido a partir de um grafo que pondera a relação entre elas. Dois critérios foram utilizados na atribuição de pesos a este grafo. O primeiro pondera a relação entre as classes a partir da distância entre os seus centróides. O segundo atribui os pesos de acordo com a confusão que ADs encontram na diferenciação das classes. Embora em vários casos os diferentes critérios tenham gerado estruturas com desempenho semelhante, alguns resultados sugerem que a escolha de um critério pode ser motivada pelas características dos dados das aplicações multiclases. Também não se verificou variações grandes de desempenho entre árvores multiclases distintas, embora elas tenham sido maiores que as observadas no caso dos DDAGs e ADAGs.

Os algoritmos formulados, embora possibilitem a adaptação das estratégias decomposicionais a cada aplicação multiclases, na prática tiveram desempenho similar ao das estratégias convencionais, incluindo dois algoritmos que reformulam o treinamento das SVMs em versões multiclases (Weston & Watkins, 1998; Crammer & Singer, 2002). De fato, segundo Rifkin & Klautau (2004), para as SVMs, a estratégia decomposicional mais simples e tradicional, 1ct, apresenta resultados bons frente a diversas outras estratégias.

Esse efeito pode ser atribuído ao fato das SVMs serem técnicas com boa capacidade de generalização. Por meio de um ajuste apropriado de parâmetros, todas as estratégias decomposicionais podem, na prática, alcançar bons resultados. Seria interessante então verificar se a adaptação das decomposições aos problemas multiclases pode ser benéfica para outras técnicas de AM.

Ainda com relação a esse aspecto, é importante salientar que todos os resultados obtidos e comentados anteriormente se aplicam às SVMs, técnica de aprendizado utilizada na geração dos preditores binários nesta Tese. O uso de outros algoritmos de AM pode levar a resultados distintos, principalmente no caso dos AGs para matrizes de códigos e para DDAGs e ADAGs, que utilizam informações do desempenho obtido na solução multiclases.

Entre os conjuntos de dados utilizados nos experimentos, incluiu-se algumas aplicações multiclases da área de Bioinformática. Embora o enfoque não tenha sido em obter os melhores resultados em cada aplicação, foi possível verificar que as técnicas propostas são também capazes de solucionar problemas práticos reais que apresentam alguns desafios na predição multiclases. Resultados obtidos com o objetivo de verificar quais estratégias seriam mais adequadas para solucionar algumas aplicações em Bioinformática podem ser consultados em artigos gerados no período, como (Lorena & Carvalho, 2005a,b; Ushizima et al., 2005; Lorena & Carvalho, 2006b).

7.2 Contribuições do Trabalho

Uma das contribuições principais deste trabalho se deu no estudo de como as estratégias decomposicionais para a resolução de problemas multiclases podem ser adaptadas a cada aplicação particular, por meio da consideração de seu desempenho na solução multiclases ou de informações de seus dados. A identificação de três possíveis áreas de investigação para a realização dessas adaptações, relacionadas a matrizes de códigos, a DAGs e a árvores multiclases, constitui outra contribuição.

Os algoritmos propostos são desenvolvimentos originais na área. A formulação da definição de matrizes de códigos e da estrutura de DDAGs e ADAGs como problemas de busca solucionados por AGs foram originadas neste trabalho. No caso das árvores multiclases, embora um agrupamento das classes com base em informações das classes tenha sido realizado previamente, os algoritmos propostos nesta Tese, por meio de adaptações simples no algoritmo de Kruskal, permitem que as partições binárias das classes sejam definidas automaticamente e eficientemente.

Também verificou-se em experimentos preliminares que os AGs podem ser uma boa alternativa para a escolha de parâmetros diferenciados para cada SVM binária. Este problema é difícil de ser solucionado empiricamente, principalmente para aplicações com números elevados de classes. A abordagem comumente adotada na literatura consiste em empregar os mesmos parâmetros para todas as SVMs binárias presentes em uma decomposição.

A realização dos experimentos implicou na implementação de diversas rotinas e bibliotecas para a geração de SVMs multiclases, as quais não foram testadas em toda a sua extensão e abrem assim a possibilidade de diversos trabalhos futuros. Além disso, os algoritmos codificados podem ser facilmente adaptados ao uso de técnicas alternativas de aprendizado na geração dos preditores binários.

Algumas observações advindas dos resultados experimentais também provêm diversas perspectivas de pesquisas futuras. Pode-se considerar ainda que os resultados experimentais confirmaram observações prévias da literatura (Rifkin & Klautau, 2004; Hsu & Lin, 2002) de que diversas estratégias multiclases apresentam resultados semelhantes, em termos de taxa de acerto total, quando as SVMs são utilizadas na geração dos preditores binários.

Também foram geradas contribuições para a área de Bioinformática no decorrer deste projeto, as quais foram relatadas principalmente em publicações geradas no período, como (Lorena & Carvalho, 2003c, 2004c, 2005b; Ushizima et al., 2005; Lorena & Carvalho, 2006b).

As pesquisas bibliográficas conduzidas podem ser consideradas uma outra contribuição desta Tese. Embora existam diversos outros trabalhos relacionados à geração de SVMs multiclases, uma vez que essa área de pesquisa tem se mostrado bastante ativa, procurou-se reunir os principais desenvolvimentos pertinentes a esse tema. Muitos deles podem inclusive ser estendidos a outras técnicas de AM.

7.3 Limitações

Algumas limitações podem ser apontadas nas estratégias propostas neste trabalho, bem como nos experimentos conduzidos.

Inicialmente, os AGs são computacionalmente custosos, embora sejam facilmente paralelizáveis e possam assim tirar proveito do crescente uso de tecnologias distribuídas. A formulação de busca de matrizes de códigos, em particular, é ainda mais custosa, uma vez que o tamanho dos indivíduos e o espaço de busca crescem consideravelmente com o número de classes dos problemas. Assim, seu uso pode não ser adequado a aplicações com números elevados de classes. Algoritmos de busca alternativos, como a busca Tabu (Glover & Laguna, 1993), a qual manipula apenas uma solução no lugar de múltiplas soluções em uma população, podem ser uma possibilidade promissora. O AG também pode ser reformulado de forma a utilizar uma codificação mais simples para os indivíduos no lugar das matrizes.

A função de avaliação dos AGs para a obtenção de matrizes de códigos e para a determinação da estrutura de DDAGs e ADAGs também pode ser considerada dispendiosa, pois envolve o treinamento e teste de diversos preditores binários. A minimização do erro em um subconjunto de validação pode levar ainda à ocorrência de super-ajustamentos (*overfitting*) das soluções obtidas pelos AGs, principalmente para conjuntos de dados com poucos dados. Pode ser aconselhável nesse caso investigar uma maneira alternativa de

avaliar os indivíduos desses AGs.

Os AGs para a determinação da estrutura de DDAGs e ADAGs possuem ainda a limitação de serem inadequados para problemas com poucas classes, que apresentam um espaço de busca muito reduzido para justificar o uso de um algoritmo de busca populacional.

No caso das árvores multiclases, algoritmos de agrupamento mais sofisticados poderiam ser empregados no processo de determinação de suas estruturas. A heurística seguida no agrupamento realizado pelo algoritmo de Kruskal é simples, implicando na formação de grupos de vértices tal que a soma das arestas entre pares deles seja mínima. Essa pode não ser a melhor alternativa para definir os grupos. Tem-se ainda que um grafo pode conter mais de uma árvore geradora mínima (AGM), caso várias de suas arestas possuam pesos reduzidos e iguais. Como o processo de obtenção das partições binárias das classes segue diretamente do algoritmo de Kruskal para a definição de AGMs, um mesmo grafo ponderado segundo os critérios de distância entre centróides ou de classes de confusão pode gerar árvores multiclases distintas. Quando isso ocorreu nos experimentos realizados, somente uma das árvores foi explorada.

Ambos os critérios para a ponderação da relação entre as classes na obtenção das árvores multiclases também possuem limitações. O uso de centróides, por exemplo, pode não ser adequado quando os dados das classes se apresentam distantes ou quando eles formam subgrupos. Nesse caso, os centróides não refletem bem a distribuição dos dados. A distância Euclidiana entre os centróides assume ainda que os grupos de dados são esféricos. Uma alternativa pode ser utilizar a distância de Mahalanobis. No critério de classes de confusão, mediu-se a confusão entre as classes por meio de ADs. Porém, as ADs possuem um viés (*bias*) próprio na separação dos dados, diferente do das SVMs. Dessa maneira, classes que apresentam confusão para as ADs não necessariamente mantêm essa característica para as SVMs.

Com relação aos experimentos realizados, uma das limitações refere-se à escolha de valores para os parâmetros livres. Os resultados das SVMs e também dos AGs são influenciados por ajustes de parâmetros. Em ambos os casos, porém, empregou-se valores comuns em diversos conjuntos de dados e também em estratégias multiclases distintas. Com isso, tanto os resultados dos AGs quanto os das outras estratégias multiclases podem ser subestimados. Entretanto, o uso dos mesmos parâmetros para as SVMs na comparação entre as diferentes estratégias multiclases pode ser considerado justo, uma vez que nesse caso todas elas se encontram em mesma desvantagem.

No caso das árvores multiclases e de algumas estratégias baseadas em matrizes de códigos da literatura, alguns procedimentos simplificados de seleção de parâmetros para as SVMs foram investigados em (Lorena & Carvalho, 2005a,b; Ushizima et al., 2005; Lorena & Carvalho, 2006b).

Seria interessante executar os AGs para a definição de matrizes de códigos com tamanhos maiores de indivíduos, ou seja, permitindo um número maior de classificadores binários nas matrizes do que o valor k adotado. Porém, isso aumentaria o espaço de busca,

o que traria a necessidade de um número maior de ciclos e de indivíduos nas populações.

O maior número de classes entre os conjuntos de dados empregados nos experimentos foi onze. Contudo, existem aplicações práticas com mais classes, as quais também implicam em uma maior complexidade de solução para todas as estratégias multiclases. Seria interessante então verificar o comportamento das técnicas estudadas em conjuntos de dados com números maiores de classes.

As análises realizadas no Capítulo 6 se concentraram principalmente na investigação das taxas de acerto totais obtidas pelas diferentes estratégias multiclases e do número de preditores binários empregado nas decomposições. Embora o desempenho nas classes e o tempo de treinamento e de teste dos classificadores de algumas estratégias tenha sido brevemente comentado, análises mais detalhadas podem ser realizadas nesses e em outros quesitos.

Alguns problemas podem ainda ser apontados na forma de comparar estatisticamente as taxas de acerto obtidas pelos AGs e pelos métodos aleatórios, em que várias soluções são geradas para cada partição de treinamento e teste de um conjunto de dados. O teste estatístico para múltiplas comparações empregado exigia o exame das saídas produzidas pelas técnicas comparadas para cada dado de teste. Dessa forma, entre as várias soluções geradas por esses métodos em cada partição dos dados, escolheu-se aquela que apresentasse uma taxa de acerto média. Essa escolha, embora reflita o comportamento médio das diversas soluções desses algoritmos, não representa o desvio-padrão encontrado em seus resultados.

De fato, existem poucos trabalhos e abordagens para a comparação múltipla de algoritmos de AM na literatura. Recentemente surgiu um artigo interessante sobre esse tema (Demsar, 2006), o qual pode prover meios alternativos de analisar os resultados dos algoritmos anteriormente mencionados.

7.4 Trabalhos Futuros

A partir das limitações apresentadas na seção anterior, uma série de trabalhos futuros podem ser apontados. Com relação aos experimentos, seria interessante verificar o comportamento das diferentes estratégias multiclases investigadas sobre conjuntos de dados com números maiores de classes. Também seria adequado variar os valores dos parâmetros dos AGs e das SVMs.

Uma pesquisa promissora pode ser utilizar algoritmos de busca, como os AGs, no ajuste de parâmetros diferenciados para cada SVM binária induzida em uma estratégia multiclases. AGs com indivíduos codificados como vetores de números reais poderiam ser utilizados nesse processo, permitindo assim uma maior variação nos valores dos parâmetros.

Com relação aos parâmetros dos AGs, existem atualmente diversas estratégias para adaptar os seus valores no decorrer do processamento desses algoritmos (Eiben, 2000), de maneira similar ao realizado na seleção probabilística de diferentes tipos de operadores

genéticos de cruzamento e de mutação nos AGs codificados neste trabalho. Contudo, a incorporação dessas modificações claramente aumenta a complexidade dos AGs, ampliando o seu espaço de busca ainda mais.

Além da consideração da taxa de acerto total das diferentes estratégias multiclases e do número de preditores binários contidos nas decomposições, outras análises podem ser realizadas. Entre elas, pode-se considerar o desempenho verificado nas classes, os tempos de treinamento e de teste dos classificadores induzidos e o tamanho dos mesmos, em KBytes. Também seria interessante analisar a complexidade computacional dos algoritmos codificados.

Como mencionado na seção anterior, recentemente Demsar (2006) apresentou testes estatísticos alternativos para a comparação múltipla de algoritmos de AM, os quais podem ser empregados em futuras análises do desempenho dos algoritmos testados neste trabalho.

É possível estender os algoritmos propostos de forma a utilizar outras técnicas de aprendizado na geração dos preditores binários. Há trabalhos na literatura que indicam que até mesmo técnicas diretamente extensíveis a problemas multiclases, como as ADs, podem se beneficiar do uso de uma abordagem decomposicional na solução do problema multiclases, a qual pode implicar em uma redução de sua complexidade (Pimenta, 2005; Fürnkranz, 2002; Dietterich & Bariki, 1995).

A função de avaliação utilizada nos AGs pode ser alterada ou mesmo estendida. Pode-se utilizar limites no erro de LOO na avaliação do desempenho dos indivíduos na solução multiclases. Esses limites provêm estimativas do erro obtido por uma SVM em um procedimento de *leave-one-out*, por meio de informações extraídas a partir de seu treinamento no conjunto de dados total (Duan et al., 2003).

Para as matrizes de códigos, existe um limite de LOO recente (Passerini et al., 2004) que pode ser empregado nesse processo, embora ele ainda seja limitado a valores baixos da constante de regularização C das SVMs binárias. A forma de avaliação de ECOCs introduzida em (Pimenta, 2005) também poderia ser adicionada ao AG para a obtenção de matrizes de códigos, garantindo assim que as matrizes geradas possuam boas propriedades de correção de erros, o que pode levar a maiores taxas de acerto na solução multiclases. Outra modificação relevante para esses AGs seria investigar funções de decodificação alternativas na integração dos preditores binários nas matrizes.

Como os AGs codificados são computacionalmente custosos, pode-se ainda explorar o uso de algoritmos de busca alternativos ou de codificações diferentes para os indivíduos manipulados pelos mesmos. Os AGs também podem ser paralelizados para uma maior eficiência computacional.

No caso das árvores multiclases, é possível explorar e propor diferentes critérios para ponderar o peso das relações entre as classes dos problemas. Além disso, seria interessante explorar que critérios são mais adequados às características particulares de cada aplicação.

Ainda nesse sentido, um trabalho promissor refere-se a investigar quais características dos dados de cada aplicação tornam uma estratégia multiclases mais adequada à sua

solução. Pode-se utilizar, por exemplo, estratégias de meta-aprendizado nesse processo (Giraud-Carrier et al., 2004; Brazdil et al., 1994).

Uma vez que os desempenhos de diferentes estratégias multiclases apresentaram algumas variações nas classes, a combinação de um subconjunto delas pode acarretar em benefícios, caso haja complementariedades entre os seus resultados. Resultados preliminares de combinações obtidos em (Lorena & Carvalho, 2003c) e (Lorena & Carvalho, 2006b) utilizando as SVMs como preditores binários, porém, não se mostraram satisfatórios. As combinações apresentaram desempenho semelhante ao emprego de uma única estratégia. Entretanto, experimentos adicionais são necessários para verificar o potencial dessa abordagem.

Uma outra possibilidade é atribuir pesos aos preditores binários nas decomposições, o que é realizado, por exemplo, em (Dekel & Singer, 2003). Um preditor binário considerado mais importante por algum critério pode assim possuir maior peso na decisão final.

7.5 Considerações Finais

Nesta Tese de Doutorado foi apresentado um trabalho em que explorou-se a solução de problemas multiclases com o uso de SVMs binárias. Problemas multiclases são inerentemente mais complexos que os binários, uma vez que eles envolvem a discriminação dos dados em um número maior de categorias (classes). Investigou-se então algoritmos para adaptar as decomposições binárias a cada problema multiclases particular, por meio do uso de informações de seu desempenho ou coletadas a partir dos dados das classes.

Os resultados experimentais obtidos quando as SVMs são utilizadas na geração dos preditores binários evidenciam que a adaptação das estratégias decomposicionais a cada problema multiclases leva em geral a resultados semelhantes aos de técnicas comumente empregadas na prática, em termos de taxa de acerto total. No caso dos AGs para a obtenção de matrizes de códigos e das árvores multiclases, tem-se o benefício de empregar menos preditores binários nas soluções, resultando em decomposições mais simples.

O trabalho realizado abre ainda diversas possibilidades de pesquisas futuras. O estudo de alternativas para a geração de soluções multiclases a partir de preditores binários pode assim ser considerado um campo de investigações promissor.

Referências Bibliográficas

- Achcar, J. A. & Rodrigues, J. (2000). Introdução à estatística para ciências e tecnologia. Relatório técnico, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Citado em: 117
- Allwein, E. L., Shapire, R. 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. Morgan Kaufmann. Citado em: 30, 31, 32, 36, 116, 118
- Alpaydin, E. & Mayoraz, E. (1999). Learning error-correcting output codes from data. In *Proceedings of the 9th International Conference on Neural Networks*, p. 743–748. Citado em: 35, 36, 37, 146, 147
- Aluha, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice Hall. Citado em: 82, 83, 84, 86
- Apweiler, R., Lopez, R., & Marx, B. (1999). Introduction to molecular biology databases. The EBI Online Manual on Molecular Databases. Disponível em: <http://www.ebi.ac.uk/swissprot/Publications/mbd1.html>. Acessado em: 07/2005. Citado em: 110
- Bahar, I., Atilgan, A. R., Jernigan, R. L., & Erman, B. (1997). Understanding the recognition of protein structural classes by amino acid composition. *Proteins: Structure, Function and Genetics*, 29:172–185. Citado em: 145
- Baranauskas, J. A. & 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/re1_tec/RT_102.ps.zip. Citado em: 11, 191, 192
- Bäck, T. (2000). *Binary Strings*, p. 132–135. In Bäck et al. (2000). Citado em: 55
- Bäck, T., Fogel, D. B., & Michalewicz, T. (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing. Citado em: 173, 174, 180

- Beasley, D. (2000). *Possible applications of evolutionary computation*, p. 4–18. In Bäck et al. (2000). Citado em: 54
- Blake, C. L. & Merz, C. J. (1998). UCI repository of machine learning databases. Disponível em: <http://www.ics.uci.edu/~mlearn/MLRepository.html>. Acessado em: 04/2002. Citado em: 102, 103, 146
- Bleuler, S., Laumanns, M., Thiele, L., & Zitzler, E. (2003). PISA - a platform and programming language independent interface for search algorithms. In Fonseca, C. M., Fleming, P. J., Zitzler, E., Deb, K., & Thiele, L., editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *Lecture Notes in Computer Science*, p. 494–508, Berlin. Springer-Verlag. Citado em: 74
- Boser, B. E., Guyon, I. L., & 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: 17
- Boser, R. C. & Ray-Chaudhuri, D. K. (1960). On a class of error-correcting binary group codes. *Information and Control*, 3:68–79. Citado em: 36
- Braga, A., Carvalho, A. C. P. L. F., & Ludermir, T. B. (2000). *Redes Neurais Artificiais: Teoria e Aplicações*. Editora LTC. Citado em: 9, 26, 108, 146
- Brazdil, P., Gama, J., & Henery, B. (1994). Characterizing the applicability of classification algorithms using meta level learning. In *Proceedings of the European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, p. 83–102. Springer-Verlag. Citado em: 171
- Bredensteiner, E. J. & Bennet, K. P. (1999). Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12(1-3):53–79. Citado em: 46
- 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: 13, 14, 19, 21, 22, 26
- Cai, Y.-D., Liu, X.-J., Xu, X.-B., & Chou, K.-C. (2002). Prediction of protein structural classes by support vector machines. *Computers and Chemistry*, 26(3):293–296. Citado em: 107, 145
- Cai, Y.-D., Liu, X.-J., Xu, X.-B., & Zhou, G.-P. (2001). Support vector machines for predicting protein structural class. *BMC Bioinformatics*, 2:3–7. Citado em: 107, 145
- Campbell, C. (2000). An introduction to kernel methods. In Howlett, R. J. & Jain, L. C., editors, *Radial Basis Function Networks: Design and Applications*, p. 155–192, Berlin. Springer Verlag. Citado em: 18, 19

- Cancino, W., Liang, Z., & Carvalho, A. C. P. L. F. (2003). Aplicação de algoritmos evolutivos multi-objetivo para o alinhamento de proteínas. In *Anais do IV Encontro Nacional de Inteligência Artificial*, volume VII, p. 297–306. Citado em: 62
- Carvalho, A. C. P. L. F., Braga, A. P., & Ludermir, T. B. (2003). Computação evolutiva. In Rezende (2003), p. 225–248. Citado em: xxi, 54, 55, 56, 57, 58
- Chang, C.-C. & Lin, C.-J. (2004a). BSVM. Disponível em: <http://www.csie.ntu.edu.tw/~cjlin/bsvm>. Acessado em: 07/2004. Citado em: 139
- Chang, C.-C. & Lin, C.-J. (2004b). LIBSVM: a library for support vector machines. Disponível em: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>. Acessado em: 09/2003. Citado em: xix, 26, 115
- Chou, K. C. & Zhang, C. T. (1995). Prediction of protein structural classes. *Critical Reviews in Biochemistry and Molecular Biology*, 30:275–349. Citado em: 106
- Coello, C. A. C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):129–156. Citado em: 62, 63, 64
- Collins, M., Shapire, R. E., & Singer, Y. (2002). Logistic regression, adaboost and bregman distances. *Machine Learning*, 47(2/3):253–285. Citado em: 147
- Collobert, R. & Bengio, S. (2001). SVM Torch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160. Citado em: 115
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2nd edition. Citado em: xxi, 85
- Cortes, C. & Vapnik, V. N. (1995). Support vector networks. *Machine Learning*, 20(3):273–296. Citado em: 18
- Crammer, K. & Singer, Y. (2001a). Improved output coding for classification using continuous relaxation. In Leen, T. K., Dietterich, T. G., & Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13, p. 437–443. MIT Press. Citado em: 48
- Crammer, K. & Singer, Y. (2001b). On the algorithm implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:35–46. Citado em: 49, 50
- Crammer, K. & Singer, Y. (2002). On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233. Citado em: xv, 37, 46, 48, 108, 139, 148, 159, 160, 165, 197, 198

- Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press. Citado em: 1, 22, 26
- Cui, Q., Jiang, T., Liu, B., & Ma, S. (2004). Esub8: A novel tool to predict protein subcellular localizations in eukaryotic organisms. *BMC Bioinformatics*, 5(1):1–7. Citado em: 109, 110
- Cvetkovic, D. & Parmee, I. (1998). Evolutionary design and multi-objective optimisation. In *6th European Congress on Intelligent Techniques and Soft Computing EUFIT'98*, p. 397–401, Aachen, Germany. Citado em: 63, 64, 66, 67
- Darwin, C. (1859). *On the origin of species by means of natural selection*. John Murray, London. Citado em: 54
- Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 162–164. Citado em: 59, 61
- Deb, K. (1999). Multi-objective evolutionary algorithms: introducing bias among pareto-optimal solutions. Relatório Técnico KanGAL 99002, Indian Institute of Technology, Kanpur, India. Citado em: 66, 123
- Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186:311–338. Citado em: 73
- Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, New York. Citado em: 63
- Dekel, O. & Singer, Y. (2003). Multiclass learning by probabilistic embeddings. In *Advances in Neural Information Processing Systems*, volume 15, p. 945–952. MIT Press. Citado em: 36, 37, 147, 171
- Demsar, J. (2006). Statistical comparisons of classifiers over multiple datasets. *Journal of Machine Learning Research*, 7:1–30. Citado em: 169, 170
- Dettling, M. & Bühlmann, P. (2002). How to use boosting for tumor classification with gene expression data. Research Report 103, Eidgenössische Technische Hochschule, Zürich, Switzerland. Citado em: 112
- Dietterich, T. G. & Bariki, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286. Citado em: 34, 35, 36, 67, 117, 118, 148, 170

- Ding, C. H. Q. & Dubchak, I. (2001). Multi-class protein fold recognition using Support Vector Machines and Neural Networks. *Bioinformatics*, 4(17):349–358. Citado em: 53, 107
- Duan, K. & Keerthi, S. S. (2005). Which is the best multiclass svm method? an empirical study. In *Proceedings of the 6th International Workshop on Multiple Classifier Systems, MCS 2005*, volume 3541 of *Lecture Notes in Computer Science*, p. 278–285. Citado em: 159
- Duan, K., Keerthi, S. S., & Poo, A. N. (2003). Evaluation of simple performance measures for tuning SVM hyperparameters. *Neurocomputing*, 51:41–59. Citado em: 170
- Duda, L., Hart, P., & Stork, D. (2001). *Pattern Classification*. John Wiley & Sons. Citado em: 86
- Dudoit, S., Fridlyand, J., & Speed, T. P. (2000). Comparison of discrimination methods for the classification of tumors using gene expression data. Relatório Técnico 576, Department of Statistics, University of Berkeley, Berkeley, CA. Citado em: 113
- Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64. Citado em: 114
- Eiben, A. E. (2000). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141. Citado em: 169
- Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer. Citado em: 54
- Emanuelsson, O., Nielsen, H., Brunak, S., & Heijne, G. V. (2000). Predicting subcellular localization of proteins based on their n-terminal amino acid sequence. *Journal of Molecular Biology*, 300(4):1005–1016. Citado em: 109
- Faceli, K., de Carvalho, A. C. P. L. F., & Souto, M. C. P. (2005). Algoritmos de agrupamento de dados. Relatório Técnico 249, 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_249.pdf. Citado em: 86
- Feelders, A. & Verkooijen, W. (1996). On the statistical comparison of inductive learning methods. In Fisher, D. & Lenz, H.-J., editors, *Learning from data: artificial intelligence and statistics V*, p. 272–279. Springer Verlag, New York, NY. Citado em: 113, 114
- Feng, J., Yang, Y., & Fan, J. (2005). Fuzzy multi-class SVM classifier based on optimal directed acyclic graph using in similar handwritten chinese characters recognition. In Wang, J., Liao, X., & Yi, Z., editors, *Proceedings of the International Symposium on Neural Networks*, volume 3496 of *Lecture Notes in Computer Science*, p. 875–880. Springer-Verlag. Citado em: 42, 43, 128, 149, 150

- Feng, Z.-P. (2002). An overview on predicting the subcellular location of a protein. *Silico Biology*, 2(3):291–303. Citado em: 109
- Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. Wiley. Citado em: 55
- Frank, 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. ACM Press. Citado em: xvii, 43, 44, 45, 133, 139, 151
- Fröhlich, H., Chapelle, O., & Schölkopf, B. (2003). Feature selection for support vector machines by means of genetic algorithms. In *Proceedings of the 15th IEEE International Conference on Tools with AI*, p. 142–148. Citado em: 152
- Friedman, J. (1996). Another approach to polychotomous classification. Relatório técnico, Department of Statistics, Stanford University. Citado em: 123
- Fürnkranz, J. (2002). Round robin classification. *Journal of Machine Learning Research*, 2:721–747. Citado em: 2, 103, 151, 170
- Furey, T., Cristianini, N., Duffy, N., Bednarski, D., Schummer, M., & Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914. Citado em: 108
- Garg, A., Bhasin, M., & 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: 109, 110
- Giraud-Carrier, C., Vilalta, R., & Brazdil, P. (2004). Introduction to the special issue on meta-learning. *Machine Learning*, 54:187–193. Citado em: 162, 171
- Glover, F. & Laguna, M. (1993). Tabu search. In Reeves, C., editor, *Modern Heuristic Techniques for Combinatorial Problems*, p. 70–150, Oxford, England. Blackwell Scientific Publishing. Citado em: 167
- Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, MA. Citado em: 55
- Goldberg, D. E. & Lingle, R. (1985). Alleles, loci, and the TSP. In *Proceedings of the 1st International Conference on Genetic Algorithms*, p. 154–159. Lawrence Erlbaum Associates. Citado em: 59
- Grimaldi, R. P. (2000). *Discrete and combinatorial mathematics: An applied introduction*. Addison-Wesley, 4th edition. Citado em: 42, 81, 82, 83

- Guermer, Y. (2000). Combining discriminant models with new multiclass SVMs. Relatório Técnico NC-TR-00-086, Neuro COLT, Loria Campus Scientifique. Citado em: 46
- Hastie, T. & Tibshirani, R. (1998). Classification by pairwise coupling. *The Annals of Statistics*, 2:451–471. Citado em: 34
- Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation*. Prentice-Hall, New Jersey, 2nd edition. Citado em: 9, 10, 14, 24, 26, 146
- Hearst, M. A., Schölkopf, B., Dumais, S., Osuna, E., & Platt, J. (1998). Trends and controversies - support vector machines. *IEEE Intelligent Systems*, 13(4):18–28. Citado em: xvii, 1, 19, 23, 25, 132
- Herbrich, R. (2001). *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press. Citado em: 25, 151
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press. Citado em: 55, 57
- Horton, P. & Nakai, K. (1997). Better prediction of protein cellular localization sites with k-nearest neighbor classifiers. In *Proceedings of the Conference in Intelligent Systems and Molecular Biology*, volume 5, p. 147–152. Citado em: 109, 110
- Hsu, C.-W., Chang, C.-C., & Lin, C.-J. (2003). A practical guide to support vector classification. Disponível em: <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>. Acessado em: 01/2004. Citado em: 112
- Hsu, C.-W. & 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: 2, 29, 46, 47, 50, 103, 111, 157, 159, 167
- Hua, S. & Sun, Z. (2001). Support vector machine approach for protein subcellular localization prediction. *Bioinformatics*, 17(8):721–728. Citado em: 109
- Isik, Z., Yanikoglu, B., & Sezerman, U. (2004). Protein structural class determination using support vector machines. In *Proceedings of the 19th International Symposium on Computer and Information Sciences*, volume 3280 of *Lecture Notes in Computer Science*, p. 82–89. Springer-Verlag. Citado em: 107, 145
- Jaakkola, T. & Haussler, D. (1999). Probabilistic kernel regression models. In *Proceedings of the 7th International Conference on AI and Statistics*. Citado em: 43, 149
- Joachims, T. (2002). *Learning to classify texts using support vector machines: methods, theory and algorithms*. Kluwer Academic Publishers. Citado em: 9

- Kijisirikul, B. & 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: xvii, 6, 40, 41, 53, 75, 76, 128, 130, 132
- Kim, K. I., Jung, K., Park, S. H., & Kim, H. J. (2002). Support vector machines for texture classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1542–1550. Citado em: 9
- Kinnear Jr, K. E. (2000). *Derivative methods in genetic programming*, p. 103–112. In Bäck et al. (2000). Citado em: 55
- Klautau, A., Jevtić, N., & Orlistky, 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: 39
- Knerr, S., Personnaz, L., & Dreyfus, G. (1990). *Single-layer learning revisited: a stepwise procedure for building and training a neural network*, p. 41–50. Springer-Verlag. Citado em: 34
- Knerr, S., Personnaz, L., & Dreyfus, G. (1992). Handwritten digit recognition by neural networks with single-layer training. *IEEE Transactions on Neural Networks*, 3(6):962–968. Citado em: 2
- Ko, J., Kim, E., & Byun, H. (2004). Improved n-division output coding for multiclass learning problems. In *Proceedings of the 17th International Conference on Pattern Recognition*, p. 470–473. IEEE Computer Society Press. Citado em: 36
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press. Citado em: 55
- Kreßel, U. (1999). Pairwise classification and support vector machines. In Schölkopf, B., Burges, C. J. C., & Smola, A. J., editors, *Advances in Kernel Methods - Support Vector Learning*, p. 185–208. MIT Press. Citado em: 34
- Kumar, V. (2000). An introduction to cluster analysis for data mining. Lecture Notes, Department of Computer Science & Engineering, University of Minnesota. Disponível em: http://www-users.cs.umn.edu/~han/dmclass/cluster_survey_10_02_00.pdf. Acessado em: 01/2006. Citado em: xxi, 87, 88
- Lee, Y., Lin, Y., & Wahba, G. (2001). Multicategory support vector machines. Relatório Técnico 1043, Department of Statistics, University of Wisconsin. Citado em: 46
- Lei, H. & Govindaraju, V. (2005). Half-against-half multi-class support vector machines. In Oza, N. C., Polikar, R., Kittler, J., & Roli, F., editors, *Proceedings of the 6th*

- International Workshop on Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, p. 156–164. Springer-Verlag. Citado em: 45, 94, 151
- Lewis, R. (2001). *Human Genetics - Concepts and Applications*. Mc Graw Hill, 4th edition. Citado em: 106
- Lorena, A. C., Batista, G. E. A. P. A., Carvalho, A. C. P. L. F., & Monard, M. C. (2002a). The influence of noisy patterns in the performance of learning methods in the splice junction recognition problem. In *Proceedings of the VII Brazilian Symposium on Neural Networks (SBRN 2002)*, p. 31–36. IEEE Computer Society Press. Citado em: 146
- Lorena, A. C., Batista, G. E. A. P. A., Carvalho, A. C. P. L. F., & Monard, M. C. (2002b). Splice junction recognition using machine learning techniques. In Bazzan, A. L. C. & Carvalho, A. C. P. L. F., editors, *Proceedings of the 1st Brazilian Workshop on Bioinformatics (WOB)*, p. 32–39. Citado em: 146
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2003a). Classificadores de margens largas (*Large Margin Classifiers*). Relatório Técnico 195, 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_195.ps.zip. Citado em: 26, 32, 146
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2003b). Evaluation of noise reduction techniques in the splice junction recognition problem. In 1st International Conference on Computational Biology and Bioinformatics (ICOBICOB), May 14-16, Ribeirão Preto, Brasil. Disponível em: <http://www.vision.ime.usp.br/~cesar/programa/pdf/159.pdf>. Citado em: 146
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2003c). Human splice site identification with multiclass support vector machines and bagging. In Kaynak, O., Xu, L., Alpaydin, E., & Oja, E., editors, *Joint 13th International Conference on Artificial Neural Networks (ICANN) and 10th International Conference on Neural Information Processing (ICONIP), Artificial Neural Networks and Neural Information Processing - ICANN/Iconip 2003*, volume 2714 of *Lecture Notes in Computer Science*, p. 234–241. Springer-Verlag. Citado em: 157, 167, 171
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2003d). Introdução às máquinas de vetores suporte (*Support Vector Machines*). Relatório Técnico 192, 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_192.ps.zip. Citado em: 26
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2003e). Revisão de técnicas para geração de classificadores de margens largas multiclases. Relatório Técnico 221, 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_221.ps. Citado em: 51

- Lorena, A. C. & Carvalho, A. C. P. L. F. (2003f). Utilização de técnicas inteligentes em bioinformática. Relatório Técnico 219, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Disponível em: ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/re1_tec/RT_219.ps. Citado em: 105
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2004a). Comparing techniques for multiclass classification with binary SVM predictors. In Monroy, R., Arroyo-Figueroa, G., Sucar, L. E., & Azuela, J. H. S., editors, *MICAI 2004: Advances in Artificial Intelligence, Third Mexican International Conference on Artificial Intelligence, Mexico City, Mexico*, volume 2972 of *Lecture Notes in Artificial Intelligence*, p. 272–281. Springer-Verlag. Citado em: 51, 157
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2004b). Evaluation of noise reduction techniques in the splice junction recognition problem. *Genetics and Molecular Biology*, 27(4):665–672. Citado em: 146
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2004c). A GA/SVM approach for multiclass classification applied to protein structural class prediction. In Barros, A., Araújo, A., Yehia, H. C., & Teixeira, R., editors, *Proceedings of the VIII Brazilian Symposium on Neural Networks, CD*. UFMA, IEEE Computer Society and SBC. Citado em: 79, 80, 146, 157, 167
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2004d). An hybrid GA/SVM approach for multiclass classification with directed acyclic graphs. In Bazzan, A. L. C. & Labidi, S., editors, *Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil*, volume 3171 of *Lecture Notes in Artificial Intelligence*, p. 366–375. Springer-Verlag. Citado em: 80, 146, 157
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2005a). Minimum spanning trees in hierarchical multiclass support vector machines generation. In Ali, M. & Esposito, F., editors, *Innovations in Applied Artificial Intelligence: 18th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2005, Bari, Italy*, volume 3533 of *Lecture Notes in Artificial Intelligence*, p. 422–431. Springer-Verlag. Citado em: 92, 99, 146, 157, 166, 168
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2005b). Protein cellular localization with multiclass support vector machines and decision trees. In Setubal, J. & Verjovski-Almeida, S., editors, *Advances in Bioinformatics and Computational Biology: Brazilian Symposium on Bioinformatics, BSB 2005, São Leopoldo, Brazil*, volume 3594 of *Lecture Notes in Bioinformatics*, p. 42–53. Springer-Verlag. Citado em: 99, 146, 157, 166, 167, 168

- Lorena, A. C. & Carvalho, A. C. P. L. F. (2006a). Evolutionary design of multiclass support vector machines. *Submetido a: Journal of Intelligent and Fuzzy Systems*. Em revisão. Citado em: 146, 157
- Lorena, A. C. & Carvalho, A. C. P. L. F. (2006b). Protein cellular localization prediction with multiclass support vector machines and decision trees. *Computers in Biology and Medicine*. Aceito, a ser publicado. Citado em: 92, 99, 145, 146, 157, 166, 167, 168, 171
- Lu, Z., Szafron, D., Greiner, R., Lu, P., Wishart, D. S., Anvik, B. P. J., Macdonell, C., & Eisner, R. (2004). Predicting subcellular localization of proteins using machine-learned classifiers. *Bioinformatics*, 20(4):547–556. Citado em: 109, 110, 145
- Luo, R., Feng, Z., & Liu, J. (2002). Prediction of protein structural class by amino acid and polypeptide composition. *European Journal of Biochemistry*, 269:4219–4225. Citado em: 106, 107, 145
- Martí, R., Laguna, M., & Campos, V. (2005). Scatter search vs. genetic algorithms: An experimental evaluation with permutation problems. In Rego, C. & Alidaee, B., editors, *Metaheuristic Optimization Via Adaptive Memory and Evolution: Tabu Search and Scatter Search*, p. 263–282. Kluwer Academic Publishers. Citado em: 72
- Mayoraz, E. & Alpaydim, E. (1998). Support vector machines for multi-class classification. Research Report IDIAP-RR-98-06, Dalle Molle Institute for Perceptual Artificial Intelligence, Martigny, Switzerland. Citado em: 36, 51
- Mayoraz, 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: 30, 33, 34, 36, 69, 146
- 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: 25
- Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 3rd edition. Citado em: 55, 59, 60, 61
- Michalewicz, Z. (2000). Introduction to constraint handling techniques. In Bäck, T., Fogel, D. B., & Michalewicz, T., editors, *Evolutionary Computation 2: Advanced Algorithms and Operators*, p. 38–40. Iop Institute of Physics. Citado em: 74
- Michalewicz, Z. & Fogel, D. B. (2004). *How to solve it: modern heuristics*. Springer. Citado em: 58
- Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood. Citado em: 104, 150, 191

- Minatti, E. (2003). Revista eletrônica de química, Universidade Federal de Santa Catarina. Disponível em <http://www.qmcweb.org>. Acessado em: 01/2003. Citado em: 106
- Mitchell, M. (1999). *An introduction to Genetic Algorithms*. MIT Press. Citado em: 5, 53, 54, 56, 74
- Mitchell, T. (1997). *Machine Learning*. McGraw Hill. Citado em: 1, 9, 10, 11, 45, 58, 67, 73, 91, 108, 109
- Müller, K. R., Mika, S., Rätsch, G., Tsuda, K., & Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201. Citado em: xvii, 13, 16, 17, 18, 20, 24
- Monard, M. C. & Baranauskas, J. A. (2003a). Conceitos de aprendizado de máquina. In Rezende (2003), p. 89–114. Citado em: 2, 10, 11, 91
- Monard, M. C. & Baranauskas, J. A. (2003b). Indução de regras e árvores de decisão. In Rezende (2003), p. 115–139. Citado em: 91
- Moreira, M. & 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: 2, 29
- Murzin, A. G., Brenner, S. E., Hubbard, T., & Cothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540. Citado em: 107
- Noble, W. S. (2004). Support vector machine applications in computational biology. In Schölkopf, B., Tsuda, K., & Vert, J.-P., editors, *Kernel Methods in computational biology*, p. 71–92. MIT Press. Citado em: 9
- Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators in the traveling salesman problem. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, p. 224–230. Lawrence Erlbaum Associates. Citado em: 59
- Ooi, C. H. & Tan, P. (2003). Genetic algorithms applied to multi-class prediction for the analysis of gene expression data. *Bioinformatics*, 19(1):37–44. Citado em: 108
- Pareto, V. (1896). *Cours d'economie politique*, volume I and II. F. Rouge, Lausanne. Citado em: 62
- Passerini, A. (2004). *Kernel Methods, multiclass classification and applications to computational molecular biology*. Tese de Doutorado, Università Degli Studi di Firenze. Citado em: xvii, 13, 18, 20, 22, 23, 29, 31, 48, 155

- Passerini, A., Pontil, M., & Frasconi, P. (2004). New results on error correcting output codes of kernel machines. *IEEE Transactions on Neural Networks*, 15:45–54. Citado em: 31, 32, 119, 122, 155, 170
- Phetkaew, T., Kijirikul, B., & Rivepiboon, W. (2003a). Learning multiclass support vector machines by reordering adaptive directed acyclic graph. In *Proceedings of the 7th National Computer Science and Engineering Conference at Burapha University*. Citado em: 42, 149
- Phetkaew, T., Kijirikul, B., & Rivepiboon, W. (2003b). 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. IEEE Computer Society Press. Citado em: 42, 149
- Pimenta, E. M. C. (2005). Abordagens para decomposição de problemas multiclasse: os códigos de correção de erros de saída. Dissertação de Mestrado, Departamento de Ciências de Computadores, Faculdade de Ciências da Universidade do Porto. Citado em: 2, 30, 36, 148, 149, 161, 170
- Pinto, A. K. O. (2004). Multialinhamento de seqüências biológicas utilizando algoritmos genéticos. Exame de Qualificação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Citado em: 54
- Platt, J. C., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*, volume 12, p. 547–553. The MIT Press. Citado em: xvii, 6, 39, 40, 75, 132
- Pontil, M. & 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: 9, 21, 22, 23
- Quilan, J. R. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann, CA. Citado em: 133, 146
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106. Citado em: 7
- Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C. H., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J. P., Poggio, T., Gerald, W., Loda, M., Lander, E. S., & Golub, T. R. (2001). Multiclass cancer diagnosis using tumor gene expression signatures. In *Proceedings of the National Academic Sciences of USA*, volume 98, p. 15149–15154. Citado em: 107, 108
- Rechenberg, I. (1965). Cybernetic solution path of an experimental problem. Relatório técnico, Ministry of Aviation, Royal Aircraft Establishment (UK). Citado em: 54

- Rechenberg, I. (1973). *Evolutionstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, Stuttgart. Citado em: 55
- Rezende, S. O. (2003). *Sistemas Inteligentes - Fundamentos e Aplicações*. Editora Manole. Citado em: 175, 184
- Rifkin, R. & Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:1533–7928. Citado em: 34, 36, 46, 47, 50, 103, 111, 115, 127, 154, 159, 165, 167
- Rifkin, R., Mukherjee, S., Tamayo, P., Ramaswamy, S., Yeang, C.-H., Angelo, M., Reich, M., Poggio, T., Lander, E. S., Golub, T. R., & Mesirov, J. P. (2003). An analytical method for multi-class molecular cancer classification. *Society of Industrial and Applied Mathematics (SIAM) Reviews*, 45(4):706–723. Citado em: 108
- Robinson, A. J. (1989). *Dynamic Error Propagation Networks*. Tese de Doutorado, Cambridge University Engineering Department. Citado em: 105
- Rätsch, G., Smola, A. J., & Mika, S. (2003). Adapting codes and embeddings for polytomies. In *Advances in Neural Information Processing Systems*, volume 15, p. 513–520. MIT Press. Citado em: 31, 36, 37, 147
- Sabino, D. M. U., Costa, L. F., Rizzatti, E. G., & Zago, M. A. (2004a). A texture approach to leukocyte recognition. *Real-Time Imaging*, 10(4):205–216. Citado em: 109
- Sabino, D. M. U., Costa, L. F., Rizzatti, E. G., & Zago, M. A. (2004b). Toward leukocyte recognition using morphometry, texture and color. In *Proceedings of the IEEE International Symposium on Biomedical Imaging*, volume 1, p. 121–124. Citado em: 109
- Savicky, P. & Fürnkranz, J. (2003). Combining pairwise classifiers with stacking. In Berthold, M. R., Lenz, H.-J., Bradley, E., Kruse, R., & Borgelt, C., editors, *Advances in Intelligent Data Analysis V, 5th International Symposium on Intelligent Data Analysis, IDA 2003*, p. 219–229. Citado em: 51
- Schölkopf, B., Guyon, I., & Weston, J. (2003). Statistical learning and kernel methods in bioinformatics. In Frasconi, P. & Shamir, R., editors, *Artificial Intelligence and Heuristic Methods in Bioinformatics*, p. 1–21. IOS Press. Citado em: 9
- Schwenker, F. (2000). Hierarchical support vector machines for multi-class pattern recognition. In *Proceedings of the 4th International Conference on Knowledge-Based Intelligent Systems and Allied Technologies*, p. 561–565. IEEE Computer Society Press. Citado em: 44, 91, 150

- Schwenker, F. & Palm, G. (2001). Tree-structured support vector machines for multi-class pattern recognition. In Kittler, J. & Roli, F., editors, *Proceedings of the International Workshop on Multiple Classifier Systems*, volume 2096 of *Lecture Notes in Computer Science*, p. 409–417. Springer-Verlag. Citado em: xvii, 38, 44, 45, 91, 150
- Setubal, J. & Meidanis, J. (1997). *Introduction to Computational Molecular Biology*. PWS Publishing Company. Citado em: 105
- Shawe-Taylor, J., Barlett, P. L., Williamson, R. C., & Anthony, M. (1998). Structural risk minimization over data-dependent hierarquies. *IEEE Transactions on Information Theory*, 44(5):1926–1940. Citado em: 17
- Smola, A. J., Barlett, P., Schölkopf, B., & Schuurmans, D. (1999). Introduction to large margin classifiers. In Smola, A. J., Barlett, P., Schölkopf, B., & Schuurmans, D., editors, *Advances in Large Margin Classifiers*, p. 1–28. MIT Press. Citado em: 13, 16, 17, 18, 21, 25
- Smola, A. J. & Schölkopf, B. (2002). *Learning with Kernels*. The MIT Press, Cambridge, MA. Citado em: xvii, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23
- Souto, M. C. P., Lorena, A. C., Delbem, A. C. B., & 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: 10, 58, 105
- Statnikov, A., Aliferis, C. F., Tsamardinos, I., Hardin, D., & Levy, S. (2005). A comprehensive evaluation of multicategory methods for microarray gene expression cancer diagnosis. *Bioinformatics*, 21(5):631–643. Citado em: 108, 113, 144, 145
- Syswerda, G. (1991). Schedule optimization using genetic algorithms. In *Handbook of Genetic Algorithms*, p. 332–349. Van Nostrand Reinhold. Citado em: 61
- Takahashi, F. & Abe, S. (2002). Decision-tree-based multiclass support vector machines. In *Proceedings of the 9th International Conference on Neural Information Processing*, volume 3, p. 1418–1422. Citado em: 43, 44, 90, 96, 150
- Takahashi, F. & Abe, S. (2003). Optimizing directed acyclic graph support vector machines. In *Proceedings of Artificial Neural Networks in Pattern Recognition*, p. 166–170. Citado em: 42, 43, 53, 128, 149
- Ushizima, D., Lorena, A. C., & Carvalho, A. C. P. L. F. (2005). Support vector machines applied to white blood cell recognition. In *Proceedings of the 5th International Conference on Hybrid Intelligent Systems (HIS 2005)*, p. 1–6. IEEE Computer Society Press. Citado em: 99, 109, 146, 157, 166, 167, 168

- Ushizima, D. M. (2004). *Diagnóstico de leucemia linfóide auxiliado por computador*. Tese de Doutorado, Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos. Citado em: 108, 145
- Vapnik, V. N. (1995). *The nature of Statistical learning theory*. Springer-Verlag, New York. Citado em: 2, 9, 14, 34, 43, 149
- Vapnik, V. N. (1998). *Statistical Learning Theory*. John Wiley and Sons. Citado em: 15, 46
- Vapnik, V. N. & 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: 9
- Vouet, D., Vouet, J. G., & Pratt, C. W. (2002). *Fundamentos de Bioquímica - versão traduzida*. Editora Artmed. Citado em: 106
- Vural, V. & 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, Banff, Alberta, Canada. Citado em: 45, 90, 139, 150, 155
- Wallet, B. C., Marchette, D. J., & Solka, J. L. (1996). A matrix representation for genetic algorithms. In *Automatic object recognition VI, Proceedings of the International Society for Optical Engineering*, p. 206–214. Citado em: 68
- Weston, J. & Watkins, V. (1998). Multi-class support vector machines. Relatório Técnico CSD-TR-98-04, Department of Computer Science, University of London. Citado em: xv, xvii, 46, 47, 48, 49, 108, 139, 159, 160, 165, 197, 198
- Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85. Citado em: 58
- Whitley, D. (2002). Genetic algorithms and evolutionary computing. In *Van Nostrand's Scientific Encyclopedia*. Citado em: 55
- Windeatt, T. & Ghaderi, R. (2003). Coding and decoding strategies for multi-class learning problems. *Information Fusion*, 4(1):11–21. Citado em: 32
- Wu, B. Y. & Chao, K.-M. (2004). *Spanning Trees and Optimization Problems*. Chapman & Hall. Citado em: 84, 86
- Xu, P. & Chan, A. K. (2003). An efficient algorithm on multi-class support vector machine model selection. In *Proceedings of the International Conference on Neural Networks*, p. 3229–3232. Citado em: 159

- Yeang, C. H., Ramaswamy, S., Tamayo, P., Mukherjee, S., Rifkin, R. M., Angelo, M., Reich, M., Lander, E., Mesirov, J., & Golub, T. (2001). Molecular classification of multiple tumor types. In *Intelligent Systems in Molecular Biology - Bioinformatics Discovery Note, Proceedings of the 9th International Conference on Intelligent Systems in Molecular Biology*, volume 1, p. 316–322, Copenhagen, Denmark. Citado em: 53, 107, 108, 144
- Zitzler, E. (2002). Evolutionary algorithms for multiobjective optimization. In Giannakoglou, K., Tsahalis, D., Periaux, J., Papailiou, K., & Fograty, T., editors, *Evolutionary Methods for Design, Optimisation, and Control, CIMNE, Barcelona, Spain*, p. 19–26. Citado em: 68
- Zitzler, E., Laumanns, M., & Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization. In Gandibleux, X., Sevaux, M., Sörensen, K., & T'kindt, V., editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*, p. 3–37. Springer-Verlag. Citado em: 56, 62, 63
- Zitzler, E., Laumanns, M., & Thiele, L. (2002). SPEA2: Improving the strength pareto evolutionary algorithm. In *Evolutionary Methods for Design, Optimisation, and Control, CIMNE, Barcelona, Spain*, p. 95–100. Citado em: xxi, 64, 65, 74, 158

Apêndice A

Avaliação de Algoritmos de AM

A avaliação de um algoritmo de AM supervisionado é normalmente realizada por meio da análise do desempenho do preditor gerado pelo mesmo na classificação de novos dados, não apresentados previamente em seu treinamento.

Em diversos casos tem-se apenas um conjunto com n dados, o qual deve ser empregado na indução do classificador e também em sua avaliação. Calcular a taxa de acerto do classificador nos mesmos dados utilizados em seu treinamento produz estimativas otimistas. Deve-se então utilizar métodos de amostragem alternativos para obter estimativas de acerto mais confiáveis. Entre os principais métodos existentes encontram-se o *holdout*, o *r-fold cross-validation* e o *leave-one-out*.

No caso do *holdout*, divide-se o conjunto de dados em uma proporção de p para treinamento e $(1 - p)$ para teste. Normalmente, emprega-se $p \in [0,7; 0,8]$ (Michie et al., 1994). Este procedimento pode subestimar a taxa de acerto, uma vez que um classificador produzido sobre todos os dados em geral apresentará uma taxa de acerto maior que o gerado a partir de uma parte deles (Baranauskas & Monard, 2000). Porém, segundo Michie et al. (1994), para conjuntos de dados grandes esse fato não representa um problema.

No método de validação cruzada *r-fold cross-validation*, o conjunto de exemplos é dividido em r subconjuntos de tamanho aproximadamente igual. Os dados de $r - 1$ partições são utilizados no treinamento de um classificador, o qual é então testado sobre a partição restante. Esse processo é repetido r vezes, utilizando em cada ciclo uma partição diferente para teste. A taxa de acerto final do classificador é dada pela média dos acertos observados sobre cada subconjunto de teste. Uma variação desse método é o *r-fold cross-validation* estratificado. Neste caso, ao gerar cada partição, mantém-se a proporção de exemplos de cada classe semelhante à proporção contida no conjunto de dados total.

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 classificador. Esse método produz uma estimativa mais fiel da taxa de acerto do classificador. Porém, ele é computacionalmente caro, sendo geralmente aplicado somente sobre 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 dados é partilhada (Baranauskas & Monard, 2000). Logo, embora as partições de teste sejam distintas entre si, com $r > 2$ não se tem completa independência entre os subconjuntos de treinamento.

Além da taxa de acerto, uma forma interessante de visualizar como o preditor se comporta na classificação dos dados pode ser obtida com o uso de uma matriz de confusão. Essa matriz ilustra o número de predições corretas e incorretas em cada classe. Para um determinado conjunto de dados, as linhas dessa matriz representam os valores de classes verdadeiros e as colunas, as classes previstas pelo classificador. Logo, cada elemento c_{ij} de uma matriz de confusão $\mathbf{C}$ apresenta o número de exemplos da classe i classificados como pertencentes à classe j . Para k classes, $\mathbf{C}$ tem então dimensão $k \times k$. Por meio do exame dessa matriz, tem-se medidas quantitativas de quais classes o algoritmo de aprendizado tem maior dificuldade em realizar uma diferenciação.

Na Figura A.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 prevista		
		1	2	3
classe verdadeira	1	11	1	3
	2	1	4	0
	3	2	1	6

Figura A.1: Exemplo de matriz de confusão para um problema com três classes

---

Apêndice B

Resultados Adicionais

Neste apêndice são reunidos informações e resultados adicionais dos experimentos executados nesta Tese.

B.1 Proporções de Dados por Classe em Conjuntos de Dados

Nas Tabelas B.1 a B.12 são apresentadas as proporções de dados por classe nos conjuntos de dados descritos na Seção 6.1. Alguns conjuntos de dados encontram-se divididos em subconjuntos de treinamento e de teste. Tem-se nesses casos as proporções de dados por classe em cada um desses subconjuntos (“%Treinamento” e “%Teste”). Nos casos em que esse tipo de divisão não encontrava-se definido, as proporções de dados por classe foram indicadas na coluna “%Treinamento”.

B.2 Obtenção de Matrizes de Códigos

Na Tabela B.13 encontram-se as taxas de exemplos desconhecidos das matrizes de códigos 1ct, tct, ECOC, aleatórias e dos AGs lexicográfico e SPEA2, geradas na Seção 6.3.

Na Tabela B.14 são apresentados os resultados dos testes estatísticos entre as estratégias baseadas em matrizes de códigos comparadas na Seção 6.3. Nessa tabela, cada estratégia multiclasse é representada por suas três letras iniciais, assim como os diferentes conjuntos de dados. Quando alguma diferença estatística entre duas técnicas “x” e “y” foi encontrada em um conjunto de dados “z”, a célula correspondente na linha “x-y” e coluna “z” foi preenchida com a estratégia que apresentou a maior taxa de acerto.

Classe	%Treinamento	%Teste
<i>Setosa</i>	33,3	-
<i>Versicolor</i>	33,3	-
<i>Virginica</i>	33,3	-

Tabela B.1: Proporções de dados nas classes do conjunto de dados *iris*

Classe	%Treinamento	%Teste
<i>Unacceptable</i>	70,0	-
<i>Acceptable</i>	22,2	-
<i>Good</i>	4,0	-
<i>Very good</i>	3,8	-

Tabela B.2: Proporções de dados nas classes do conjunto de dados *car*

Classe	%Treinamento	%Teste
<i>Building window float processed</i>	32,7	-
<i>Building window non float processed</i>	35,5	-
<i>Vehicle window float processed</i>	7,9	-
<i>Container</i>	6,1	-
<i>Tableware</i>	4,2	-
<i>Headlamps</i>	13,6	-

Tabela B.3: Proporções de dados nas classes do conjunto de dados *glass*

Classe	%Treinamento	%Teste
<i>Red soil</i>	24,2	23,0
<i>Cotton crop</i>	10,8	11,2
<i>Grey soil</i>	21,7	19,9
<i>Damp grey soil</i>	9,4	10,6
<i>Soil with vegetation stubble</i>	10,6	11,8
<i>Very damp grey soil</i>	23,4	23,5

Tabela B.4: Proporções de dados nas classes do conjunto de dados *sat-image*

Classe	%Treinamento	%Teste
<i>Brickface</i>	14,3	-
<i>Sky</i>	14,3	-
<i>Foliage</i>	14,3	-
<i>Cement</i>	14,3	-
<i>Window</i>	14,3	-
<i>Path</i>	14,3	-
<i>Grass</i>	14,3	-

Tabela B.5: Proporções de dados nas classes do conjunto de dados *segment*

Classe	%Treinamento	%Teste
0	9,8	9,9
1	10,2	10,1
2	9,9	9,8
3	10,2	10,2
4	10,1	10,1
5	9,8	10,1
6	9,9	10,1
7	10,1	10,0
8	9,9	9,7
9	10,0	10,0

Tabela B.6: Proporções de dados nas classes do conjunto de dados *opt-digits*

Classe	%Treinamento	%Teste
0	10,4	10,4
1	10,4	10,4
2	10,4	10,4
3	9,6	9,6
4	10,4	10,4
5	9,6	9,6
6	9,6	9,6
7	10,4	10,4
8	9,6	9,6
9	9,6	9,6

Tabela B.7: Proporções de dados nas classes do conjunto de dados *pen-digits*

Classe	%Treinamento	%Teste
i	9,1	9,1
I	9,1	9,1
E	9,1	9,1
A	9,1	9,1
a:	9,1	9,1
Y	9,1	9,1
O	9,1	9,1
C:	9,1	9,1
U	9,1	9,1
u:	9,1	9,1
3:	9,1	9,1

Tabela B.8: Proporções de dados nas classes do conjunto de dados *vowel*

Classe	%Treinamento	%Teste
$all-\alpha$	17,6	15,8
$all-\beta$	34,8	30,4
α/β	36,7	37,7
$\alpha+\beta$	10,9	16,1

Tabela B.9: Proporções de dados nas classes do conjunto de dados *structural*

Classe	%Treinamento	%Teste
<i>Histologically defined lung adenocarcinomas</i>	68,5	-
<i>Normal</i>	8,4	-
<i>Squamous cell lung carcinomas</i>	10,3	-
<i>Pulmonary carcinoids</i>	9,8	-
<i>Small-cell lung carcinomas</i>	3,0	-

Tabela B.10: Proporções de dados nas classes do conjunto de dados *lung*

Classe	%Treinamento	%Teste
Basófilos	4,2	-
Eosinófilos	10,0	-
Linfócitos	26,9	-
Monócitos	10,2	-
Neutrófilos	23,2	-
Leucemia linfóide crônica	14,3	-
Prolinfócitos	4,5	-
Células cabeludas	6,6	-

Tabela B.11: Proporções de dados nas classes do conjunto de dados *leukocyte*

Classe	%Treinamento	%Teste
<i>Mitochondrion</i>	21,9	-
<i>Nucleus</i>	34,6	-
<i>Endoplasmic reticulum</i>	4,1	-
<i>Extracellular</i>	9,9	-
<i>Cytoplasm</i>	21,1	-
<i>Plasma membrane</i>	1,4	-
<i>Golgi</i>	1,7	-
<i>Peroxisome</i>	3,1	-
<i>Vacuole</i>	2,2	-

Tabela B.12: Proporções de dados nas classes do conjunto de dados *fungi*

Conjunto						AGs	
	1ct	tct	ECOC	Aleatório		Lexicográfico	SPEA2
<i>iris</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	4,5 (11,6)		0,0 (0,0)	0,0 (0,0)
<i>car</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	2,9 (12,3)		0,0 (0,1)	0,1 (1,2)
<i>lung</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)		0,0 (0,0)	0,3 (1,3)
<i>glass</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	6,2 (15,4)		0,2 (0,7)	0,5 (1,8)
<i>segment</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	5,2 (8,9)		0,0 (0,0)	0,4 (3,5)
<i>leukocyte</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	5,0 (8,2)		0,0 (0,1)	1,4 (5,9)
<i>fungi</i>	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	1,9 (7,3)		0,0 (0,0)	0,2 (0,7)
<i>structural</i>	0,0	0,0	0,0	0,1 (0,5)		0,0 (0,1)	0,2 (0,6)
<i>sat-image</i>	0,0	0,0	0,0	2,0 (4,7)		0,0 (0,1)	1,1 (6,1)
<i>opt-digits</i>	0,0	0,0	0,0	3,1 (5,1)		0,0 (0,0)	5,2 (15,4)
<i>pen-digits</i>	0,0	0,0	0,0	7,2 (11,7)		0,0 (0,0)	6,6 (17,7)
<i>vowel</i>	0,0	0,0	0,0	2,8 (6,2)		0,0 (0,1)	0,9 (1,7)

Tabela B.13: Taxas de desconhecidos das estratégias baseadas em matrizes de códigos

	iri	car	lun	gla	seg	leu	fun	str	sat	opt	pen	vow
1ct-tct		tct					tct					
1ct-eco							eco					
1ct-ale		1ct	1ct	1ct	1ct	1ct	1ct	1ct	1ct	1ct	1ct	1ct
1ct-lex		lex					lex					
1ct-spe					1ct	1ct	spe	1ct	1ct	1ct	1ct	1ct
tct-eco		tct										
tct-ale		tct		tct	tct	tct	tct	tct	tct	tct	tct	tct
tct-lex			lex									tct
tct-spe		tct			tct	tct		tct	tct	tct	tct	tct
eco-ale		eco	eco	eco	eco	eco	eco	eco	eco	eco	eco	eco
eco-lex		lex										
eco-spe					eco	eco		eco	eco	eco	eco	eco
ale-lex		lex	lex	lex	lex	lex	lex	lex	lex	lex	lex	lex
ale-spe		spe	spe	spe	spe	spe	spe		spe	spe	spe	
lex-spe		lex			lex	lex	lex	lex	lex	lex	lex	lex

Tabela B.14: Testes estatísticos entre estratégias baseadas em matrizes de códigos

	iri	car	lun	gla	seg	leu	fun	str	sat	opt	pen	vow
c1-cc1								cc1				
c1-c2												
c1-cc2								cc2			c1	
c1-ale		c1			c1			ale				
cc1-c2								cc1				
cc1-cc2												
cc1-ale		cc1			cc1		ale					
c2-cc2								cc2			c2	
c2-ale		c2			c2			ale				c2
cc2-ale		cc2			cc2		ale			cc2		cc2

Tabela B.15: Testes estatísticos entre árvores multiclases

B.3 Determinação da Estrutura de Árvores Multiclases

Os resultados dos testes estatísticos executados entre as árvores multiclases geradas na Seção 6.5 encontram-se na Tabela B.15, construída de maneira similar à Tabela B.14. Os símbolos “c1” e “cc1” denotam o uso dos critérios de peso de distância entre centróides e de classes de confusão com o Algoritmo 5.4, enquanto “c2” e “cc2” indicam a mesma informação para o Algoritmo 5.5.

B.4 Comparações de Diferentes Estratégias

Na Tabela B.16 são apresentados os resultados dos testes estatísticos as estratégias multiclases comparadas na Seção 6.6: as matrizes de códigos do AG lexicográfico, os DDAGs aleatórios, as árvores multiclases e os algoritmos de Weston & Watkins (1998) (WW) e de Crammer & Singer (2002) (CS). A forma de obtenção dessa tabela é a mesma empregada no caso das tabelas B.14 e B.15. Os conjuntos de dados e as técnicas investigadas são representados por suas letras iniciais.

	iri	car	lun	gla	seg	leu	fun	str	sat	opt	pen	vow
lex-dda			lex									
lex-ár			lex				lex					
lex-ww		lex										
lex-cs												
dda-ár							dda					
dda-ww		dda			dda							
dda-cs			cs									
ár-ww		ár					ww					
ár-cs			cs				cs					
ww-cs		cs	cs									

Tabela B.16: Testes estatísticos entre diferentes estratégias multiclases

	Classes		
Téc.	<i>Setosa</i>	<i>Versicolor</i>	<i>Virginica</i>
1ct	100,0 (0,0)	96,0 (8,4)	96,0 (8,4)
tct	100,0 (0,0)	90,0 (10,5)	98,0 (6,3)
ECOC	100,0 (0,0)	96,0 (8,4)	96,0 (8,4)
AG lex	98,9 (3,4)	91,1 (9,6)	94,9 (8,6)
DDAG	100,0 (0,0)	90,0 (10,5)	100,0 (0,0)
ADAG	100,0 (0,0)	90,0 (10,5)	100,0 (0,0)
Árvore	100,0 (0,0)	90,0 (10,5)	100,0 (0,0)
WW	100,0 (0,0)	90,0 (10,5)	96,0 (8,4)
CS	100,0 (0,0)	90,0 (10,5)	100,0 (0,0)

Tabela B.17: Taxas de acerto nas classes do conjunto de dados *iris*

B.5 Desempenho nas Classes

Nas Tabelas B.17 a B.28 são apresentadas as taxas de acerto nas classes dos conjuntos de dados indicados na Tabela 6.1, obtidas por diferentes estratégias multiclases. Nessas tabelas, “AG lex” indica a média das taxas de acerto das matrizes do AG lexicográfico, criadas na Seção 6.3. “Árvore” corresponde à árvore multiclases determinada pelo Algoritmo 5.4 de melhor desempenho segundo a Tabela 6.10. Ou seja, nos conjuntos de dados *iris*, *car*, *glass*, *fungi*, *opt-digits* e *pen-digits* utilizou-se as árvores geradas com o critério de distância entre centróides, enquanto nos demais conjuntos de dados, empregou-se as árvores do critério de classes de confusão. Nos casos em que as taxas de acerto das árvores de ambos os critérios foram iguais, selecionou-se o de distância entre centróides. As taxas de acerto dos DDAGs e dos ADAGs correspondem à média de trinta estruturas geradas aleatoriamente, como descrito na Seção 6.4. “WW” indica o algoritmo de Weston & Watkins (1998) e “CS” denota o algoritmo de Crammer & Singer (2002). Os nomes das classes em cada conjunto de dados foram escritos por extenso ou abreviados, podendo ser consultados por completo na Seção 6.1 ou nas tabelas da Seção B.1.

Téc.	Classes			
	<i>Unacceptable</i>	<i>Acceptable</i>	<i>Good</i>	<i>Very Good</i>
1ct	97,4 (3,0)	92,7 (5,8)	48,8 (24,1)	88,1 (13,3)
tct	98,4 (1,8)	98,2 (1,8)	90,7 (15,8)	100,0 (0,0)
ECOC	97,1 (3,2)	94,3 (6,0)	40,0 (24,1)	85,0 (13,5)
AG lex	98,4 (1,7)	97,1 (2,7)	90,7 (8,6)	97,7 (3,7)
DDAG	98,4 (1,8)	96,4 (3,0)	92,1 (16,0)	100,0 (0,0)
ADAG	98,4 (1,8)	96,4 (3,0)	92,1 (16,0)	100,0 (0,0)
Árvore	98,4 (1,8)	97,4 (2,8)	92,1 (16,0)	100,0 (0,0)
WW	97,9 (2,3)	96,4 (3,3)	78,3 (15,3)	93,8 (11,2)
CS	98,5 (1,8)	96,1 (3,3)	89,8 (9,8)	98,3 (5,3)

Tabela B.18: Taxas de acerto nas classes do conjunto de dados *car*

Téc.	Classes				
	<i>HDLC</i>	<i>Normal</i>	<i>Squamous CLC</i>	<i>PC</i>	<i>Small CLC</i>
1ct	97,1 (5,0)	76,7 (8,8)	66,7 (16,5)	33,3 (57,7)	0,0 (0,0)
tct	99,3 (1,3)	52,2 (13,5)	38,1 (8,3)	0,0 (0,0)	0,0 (0,0)
ECOC	97,1 (5,0)	76,7 (8,8)	66,7 (16,5)	33,3 (57,7)	0,0 (0,0)
AG lex	90,6 (4,6)	81,4 (7,0)	74,3 (17,6)	89,3 (15,8)	3,3 (3,3)
DDAG	98,6 (2,5)	52,2 (13,5)	52,4 (8,2)	0,0 (0,0)	0,0 (0,0)
ADAG	98,6 (2,5)	52,2 (13,5)	52,4 (8,2)	0,0 (0,0)	0,0 (0,0)
Árvore	99,3 (1,3)	46,7 (17,6)	47,6 (8,3)	0,0 (0,0)	0,0 (0,0)
WW	97,1 (5,0)	76,7 (8,8)	66,7 (16,5)	33,3 (57,7)	0,0 (0,0)
CS	97,8 (2,2)	83,3 (16,7)	71,4 (24,7)	95,2 (8,3)	66,7 (57,7)

Tabela B.19: Taxas de acerto nas classes do conjunto de dados *lung*

Téc.	Classes					
	<i>Build. Proc.</i>	<i>Build. N. Proc.</i>	<i>Car. Proc.</i>	<i>Container</i>	<i>Tableware</i>	<i>Headlamp</i>
1ct	82,9 (12,0)	67,3 (9,7)	0,0 (0,0)	80,0 (18,3)	60,0 (54,8)	86,7 (7,5)
tct	80,0 (14,6)	71,2 (10,6)	5,0 (11,2)	73,3 (36,5)	90,0 (22,4)	83,3 (11,8)
ECOC	81,4 (12,0)	68,6 (12,0)	0,0 (0,0)	80,0 (18,3)	50,0 (50,0)	86,7 (7,5)
AG lex	83,2 (7,8)	63,4 (13,2)	11,7 (6,9)	67,0 (20,3)	54,0 (33,3)	84,1 (8,9)
DDAG	80,0 (14,6)	69,0 (8,1)	5,0 (11,2)	80,0 (29,8)	80,0 (27,4)	83,9 (10,8)
ADAG	80,0 (14,6)	69,1 (8,1)	5,0 (11,2)	80,0 (29,8)	80,0 (27,4)	83,6 (11,4)
Árvore	78,6 (11,3)	66,0 (14,2)	5,0 (11,2)	73,3 (36,5)	80,0 (27,4)	86,7 (7,5)
WW	84,3 (11,7)	67,3 (11,7)	10,0 (13,7)	73,3 (27,9)	60,0 (54,8)	86,7 (7,5)
CS	82,9 (13,9)	63,4 (16,2)	5,0 (11,2)	80,0 (18,3)	50,0 (50,0)	83,3 (11,8)

Tabela B.20: Taxas de acerto nas classes do conjunto de dados *glass*

	Classes						
Téc.	<i>Brickface</i>	<i>Sky</i>	<i>Foliage</i>	<i>Cement</i>	<i>Window</i>	<i>Path</i>	<i>Grass</i>
1ct	99,1 (2,1)	100,0 (0,0)	96,1 (2,5)	93,0 (5,2)	80,3 (6,4)	100,0 (0,0)	99,1 (1,5)
tct	98,8 (2,1)	100,0 (0,0)	94,9 (3,5)	94,2 (4,4)	89,1 (4,6)	100,0 (0,0)	99,4 (1,3)
ECOC	99,1 (2,1)	100,0 (0,0)	95,8 (2,9)	92,1 (5,4)	78,2 (5,3)	100,0 (0,0)	99,1 (1,5)
AG lex	98,6 (2,3)	100,0 (0,1)	92,1 (4,2)	94,2 (3,9)	90,1 (5,1)	99,6 (0,6)	98,9 (1,4)
DDAG	98,8 (2,1)	100,0 (0,0)	95,2 (2,9)	94,7 (4,6)	88,8 (4,1)	100,0 (0,0)	99,7 (1,0)
ADAG	98,8 (2,1)	100,0 (0,0)	95,1 (2,9)	94,8 (4,6)	88,8 (4,1)	100,0 (0,0)	99,7 (1,0)
Árv.	98,8 (2,1)	100,0 (0,0)	94,6 (3,1)	93,3 (4,0)	88,8 (4,8)	100,0 (0,0)	98,8 (1,6)
WW	98,8 (2,1)	100,0 (0,0)	94,2 (3,3)	94,2 (4,2)	85,8 (5,0)	99,7 (1,0)	99,1 (1,5)
CS	98,8 (2,1)	100,0 (0,0)	95,8 (2,6)	94,2 (4,4)	89,1 (5,2)	99,7 (1,0)	99,1 (1,5)

Tabela B.21: Taxas de acerto nas classes do conjunto de dados *segment*

	Classes			
Téc.	B	E	L	M
1ct	90,5 (12,4)	94,6 (6,4)	88,4 (6,0)	86,8 (12,0)
tct	93,0 (11,4)	93,6 (6,2)	91,2 (5,8)	90,8 (9,6)
ECOC	92,5 (12,1)	94,6 (6,4)	90,1 (6,0)	90,7 (8,6)
AG lex	86,7 (9,9)	93,7 (4,2)	88,9 (5,6)	87,8 (9,2)
DDAG	93,0 (11,4)	94,6 (6,2)	87,7 (6,2)	90,3 (9,5)
ADAG	93,0 (11,4)	94,6 (6,2)	87,6 (6,2)	90,7 (8,9)
Árvore	91,0 (11,7)	93,6 (6,2)	88,7 (4,3)	86,1 (13,4)
WW	90,5 (12,4)	94,6 (6,4)	88,4 (6,9)	87,0 (8,9)
CS	90,5 (12,4)	94,6 (6,4)	89,1 (7,6)	87,0 (8,9)
	N	LLC	P	CC
1ct	97,6 (2,8)	88,7 (7,9)	73,0 (19,5)	80,0 (19,3)
tct	97,2 (3,4)	87,4 (9,1)	81,5 (19,4)	75,7 (16,6)
ECOC	98,4 (2,1)	87,4 (7,1)	79,5 (18,3)	80,0 (19,3)
AG lex	95,9 (3,8)	87,5 (8,8)	71,1 (18,7)	77,9 (16,3)
DDAG	97,0 (3,0)	87,5 (8,5)	82,1 (15,9)	77,3 (18,6)
ADAG	97,0 (3,1)	87,6 (8,4)	81,1 (16,2)	77,6 (18,8)
Árvore	97,2 (3,9)	86,8 (8,3)	71,5 (20,8)	74,3 (17,6)
WW	95,6 (4,4)	89,4 (9,0)	71,0 (27,3)	78,6 (18,1)
CS	96,0 (4,3)	88,8 (8,9)	73,0 (23,6)	78,6 (18,1)

Tabela B.22: Taxas de acerto nas classes do conjunto de dados *leukocyte*

Téc.	Classes				
	<i>mit.</i>	<i>nuc.</i>	<i>end.</i>	<i>ext.</i>	<i>cyt.</i>
1ct	42,4 (5,8)	95,2 (2,0)	2,1 (4,5)	94,9 (4,9)	20,3 (4,0)
tct	50,4 (5,6)	90,5 (2,5)	0,0 (0,0)	64,4 (9,8)	55,5 (6,8)
ECOC	45,9 (6,6)	94,2 (2,4)	0,0 (0,0)	86,7 (6,1)	45,6 (5,6)
AG lex	55,9 (5,6)	84,7 (3,3)	0,6 (0,7)	84,3 (7,8)	57,5 (5,2)
DDAG	47,6 (6,7)	92,3 (2,6)	0,0 (0,0)	72,9 (9,5)	53,7 (5,2)
ADAG	47,4 (6,6)	92,5 (2,6)	0,0 (0,0)	72,7 (9,6)	53,7 (5,2)
Árvore	70,4 (11,7)	56,4 (5,7)	0,0 (0,0)	40,7 (11,2)	63,3 (9,4)
WW	55,2 (6,9)	87,5 (1,7)	1,0 (3,2)	77,7 (7,2)	54,5 (5,9)
CS	50,2 (5,8)	91,9 (3,3)	1,0 (3,2)	93,6 (6,4)	46,0 (5,9)
	<i>pla.</i>	<i>gol.</i>	<i>per.</i>	<i>vac.</i>	
1ct	0,0 (0,0)	0,0 (0,0)	2,9 (6,0)	0,0 (0,0)	
tct	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	
ECOC	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	
AG lex	0,1 (0,4)	0,0 (0,0)	0,1 (0,2)	0,2 (0,7)	
DDAG	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	
ADAG	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	
Árvore	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	
WW	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	
CS	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	0,0 (0,0)	

Tabela B.23: Taxas de acerto nas classes do conjunto de dados *fungi*

Téc.	Classes			
	<i>all-α</i>	<i>all-β</i>	<i>α/β</i>	<i>$\alpha+\beta$</i>
1ct	83,0	82,9	93,8	41,9
tct	82,0	82,1	93,8	48,4
ECOC	82,0	83,8	94,5	41,9
AG lex	81,2 (2,6)	78,9 (4,5)	93,4 (2,1)	50,4 (9,4)
DDAG	82,0 (0,0)	82,1 (0,0)	93,3 (0,3)	50,0 (0,0)
ADAG	82,0 (0,0)	82,1 (0,0)	93,3 (0,3)	50,0 (0,0)
Árvore	67,2	88,0	95,2	38,7
WW	83,6	82,9	94,5	43,6
CS	80,3	84,6	95,2	41,9

Tabela B.24: Taxas de acerto nas classes do conjunto de dados *structural*

Téc.	Classes					
	<i>Red soil</i>	<i>Cotton crop</i>	<i>Grey soil</i>	<i>Damp grey</i>	<i>Vegetation</i>	<i>Very damp grey</i>
1ct	99,8	96,4	95,0	63,0	89,0	87,7
tct	99,6	96,9	94,7	61,6	88,2	87,5
ECOC	99,6	95,5	94,7	62,1	88,6	87,7
AG lex	99,2 (0,7)	96,1 (0,5)	94,9 (1,1)	62,2 (3,1)	88,9 (0,9)	86,9 (1,4)
DDAG	99,3 (0,2)	96,2 (0,3)	94,7 (0,0)	59,4 (0,2)	88,3 (0,2)	87,0 (0,2)
ADAG	99,4 (0,2)	96,2 (0,3)	94,7 (0,0)	59,4 (0,2)	88,3 (0,2)	87,0 (0,2)
Árvore	99,1	97,3	95,5	59,2	88,2	87,0
WW	99,4	96,0	94,5	64,5	89,5	86,8
CS	99,8	95,5	94,5	63,0	89,9	87,9

Tabela B.25: Taxas de acerto nas classes do conjunto de dados *sat-image*

	Classes				
Téc.	0	1	2	3	4
lct	100,0	97,8	97,7	97,3	97,8
tct	99,4	98,9	92,7	94,0	97,8
ECOC	100,0	97,8	98,3	98,4	97,8
AG lex	99,9 (0,4)	98,0 (0,9)	94,8 (2,9)	97,2 (1,6)	97,4 (0,8)
DDAG	100,0 (0,0)	98,4 (0,0)	95,4 (0,5)	94,8 (0,7)	98,9 (0,0)
ADAG	100,0 (0,0)	98,4 (0,0)	95,4 (0,4)	94,9 (0,6)	98,9 (0,0)
Árvore	100,0	98,9	95,5	95,1	98,9
WW	100,0	97,3	98,3	98,4	97,8
CS	99,4	97,8	97,7	97,3	97,2
	5	6	7	8	9
lct	98,9	98,3	95,5	94,3	95,6
tct	98,4	98,3	95,0	92,0	96,7
ECOC	99,5	98,9	95,0	94,3	96,1
AG lex	98,1 (1,3)	98,2 (0,9)	95,0 (1,7)	92,1 (1,7)	96,3 (0,8)
DDAG	99,5 (0,0)	99,5 (0,0)	94,9 (0,5)	92,4 (0,6)	96,7 (0,0)
ADAG	99,5 (0,0)	99,5 (0,0)	94,9 (0,4)	92,4 (0,4)	96,7 (0,0)
Árvore	98,4	98,3	93,9	91,4	97,2
WW	99,5	98,3	94,4	93,7	95,0
CS	99,5	98,3	94,4	94,3	95,0

Tabela B.26: Taxas de acerto nas classes do conjunto de dados *opt-digits*

	Classes				
Téc.	0	1	2	3	4
lct	97,5	97,8	99,5	99,1	99,5
tct	94,5	96,4	98,9	99,1	97,8
ECOC	97,3	97,3	99,5	98,8	98,6
AG lex	97,3 (0,6)	97,9 (1,4)	98,8 (0,5)	98,9 (0,4)	98,2 (0,8)
DDAG	96,6 (0,5)	96,7 (0,0)	98,7 (0,1)	98,9 (0,1)	98,4 (0,1)
ADAG	96,6 (0,4)	96,7 (0,0)	98,8 (0,1)	98,9 (0,1)	98,4 (0,1)
Árvore	97,8	97,5	99,2	99,1	98,9
WW	97,3	97,5	98,9	99,1	99,7
CS	97,3	98,1	98,4	99,1	99,7
	5	6	7	8	9
lct	98,5	100,0	91,5	99,4	96,7
tct	98,5	100,0	91,5	99,4	97,3
ECOC	98,5	99,7	91,8	99,7	97,3
AG lex	98,3 (0,8)	98,8 (1,1)	90,2 (1,7)	99,2 (0,6)	96,3 (0,8)
DDAG	98,5 (0,0)	100,0 (0,0)	92,6 (0,3)	99,1 (0,0)	97,7 (0,2)
ADAG	98,5 (0,0)	100,0 (0,0)	92,6 (0,3)	99,1 (0,0)	97,7 (0,2)
Árvore	96,1	99,7	92,6	99,4	95,5
WW	98,5	99,4	91,8	99,4	95,8
CS	97,9	99,4	90,4	99,1	97,0

Tabela B.27: Taxas de acerto nas classes do conjunto de dados *pen-digits*

	Classes					
Téc.	i	I	E	A	a:	Y
1ct	71,4	45,2	66,7	78,6	28,6	35,7
tct	73,8	59,5	52,4	61,9	28,6	83,3
ECOC	83,3	50,0	66,7	78,6	28,6	33,3
AG lex	70,7 (9,0)	50,2 (13,0)	68,4 (16,0)	58,1 (18,0)	24,4 (10,8)	42,6 (12,4)
DDAG	74,4 (1,0)	63,6 (2,7)	37,9 (3,4)	64,3 (0,0)	33,3 (0,0)	77,0 (1,4)
ADAG	74,5 (1,1)	62,6 (1,6)	39,0 (2,4)	64,3 (0,0)	33,3 (0,0)	76,4 (0,7)
Árvore	71,4	59,5	52,4	54,8	16,7	64,3
WW	71,4	38,1	59,5	66,7	26,2	59,5
CS	73,8	45,2	57,1	73,8	23,8	35,7
	O	C:	U	u:	3:	
1ct	19,1	76,2	64,3	42,9	40,5	
tct	45,2	47,6	81,0	33,3	66,7	
ECOC	21,4	85,7	35,7	47,6	69,1	
AG lex	39,6 (21,0)	65,3 (11,6)	58,6 (12,6)	32,9 (10,9)	42,1 (26,1)	
DDAG	51,0 (1,2)	61,9 (0,0)	67,1 (1,0)	26,9 (1,1)	49,0 (7,2)	
ADAG	51,2 (1,2)	61,9 (0,0)	67,4 (1,1)	26,7 (1,0)	47,9 (6,7)	
Árvore	50,0	78,6	73,8	19,1	71,4	
WW	50,0	71,4	66,7	38,1	45,2	
CS	52,4	69,1	59,5	38,1	35,7	

Tabela B.28: Taxas de acerto nas classes do conjunto de dados *vowel*