



**COPPE/UFRJ**

PROGRAMAÇÃO GENÉTICA MULTI-POPULACIONAL E  
CO-EVOLUCIONÁRIA PARA CLASSIFICAÇÃO DE DADOS

Douglas Adriano Augusto

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Civil, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Civil.

Orientadores: Nelson Francisco Favilla Ebecken  
Helio José Corrêa Barbosa

Rio de Janeiro

Agosto de 2009

PROGRAMAÇÃO GENÉTICA MULTI-POPULACIONAL E  
CO-EVOLUCIONÁRIA PARA CLASSIFICAÇÃO DE DADOS

Douglas Adriano Augusto

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ  
COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE)  
DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS  
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR  
EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

---

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

---

Prof. Helio José Corrêa Barbosa, D.Sc.

---

Prof. Alexandre Gonçalves Evsukoff, Dr.

---

Prof<sup>ª</sup>. Beatriz de Souza Leite Pires de Lima, D.Sc.

---

Prof. Eduardo Raul Hruschka, D.Sc.

---

Prof<sup>ª</sup>. Fernanda Araujo Baião, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

AGOSTO DE 2009

Augusto, Douglas Adriano

Programação Genética Multi-populacional e Co-evolucionária para Classificação de Dados/Douglas Adriano Augusto. – Rio de Janeiro: UFRJ/COPPE, 2009.

XVIII, 147 p.: il.; 29, 7cm.

Orientadores: Nelson Francisco Favilla Ebecken

Helio José Corrêa Barbosa

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Civil, 2009.

Referências Bibliográficas: p. 110 – 120.

1. Programação Genética. 2. Co-evolução. 3. Combinação de Classificadores. 4. Classificação de Dados. I. Ebecken, Nelson Francisco Favilla *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Civil. III. Título.

*Àqueles que em algum grau  
possam usufruir desta obra.*

# Agradecimentos

À minha família, em especial aos meus pais.

Aos meus orientadores, Helio e Nelson, pelo nobre exercício da orientação, sabedoria, suporte e amizade.

À Agência Nacional do Petróleo, por intermédio do Programa de Recursos Humanos para o Setor Petróleo e Gás (PRH-02-ANP/MCT), pelo suporte financeiro à esta pesquisa na forma de concessão de bolsa de estudos.

Ao Núcleo de Atendimento em Computação de Alto Desempenho (NACAD) da COPPE/UFRJ, pela disponibilização do soberbo parque computacional no qual foram realizados os numerosos experimentos.

Ao Laboratório Nacional de Computação Científica (LNCC), pelo auxílio nos recursos computacionais que viabilizaram a finalização da bateria de experimentos dentro do prazo.

Aos grandes filósofos e engenheiros do Software Livre, pela disponibilização *livre* e irrestrita de ferramentas e ambientes computacionais tecnicamente privilegiados.

A todos pesquisadores e seus esforços prévios, que contribuíram (e contribuem) não somente com a confecção deste trabalho, mas principalmente no desenvolvimento sólido do conhecimento científico.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PROGRAMAÇÃO GENÉTICA MULTI-POPULACIONAL E  
CO-EVOLUCIONÁRIA PARA CLASSIFICAÇÃO DE DADOS

Douglas Adriano Augusto

Agosto/2009

Orientadores: Nelson Francisco Favilla Ebecken  
Helio José Corrêa Barbosa

Programa: Engenharia Civil

Este trabalho apresenta um novo método evolucionário de combinação de soluções para classificação de dados. A abordagem é baseada em um algoritmo de programação genética multi-populacional que explora a técnica de co-evolução em dois níveis. No nível inter-populacional as populações cooperam em um regime semi-isolado, enquanto que no nível intra-populacional os classificadores candidatos co-evoluem competitivamente com amostras de treinamento. O classificador final é, ao término do processo evolutivo, um comitê de votação composto pelos melhores membros de todas as populações.

Duas principais contribuições resultam desta tese: (i) um novo modelo de algoritmo evolucionário para classificação de dados, como descrito acima; e (ii) uma sofisticada implementação deste algoritmo, disponível livremente, para uso em ambientes de alto desempenho.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

COEVOLUTIONARY MULTI-POPULATION GENETIC PROGRAMMING  
FOR DATA CLASSIFICATION

Douglas Adriano Augusto

August/2009

Advisors: Nelson Francisco Favilla Ebecken

Helio José Corrêa Barbosa

Department: Civil Engineering

This work presents a new evolutionary ensemble method for data classification. The approach is based on a multiple-population genetic-programming algorithm which exploits the technique of co-evolution at two levels. On the inter-population level the populations cooperate in a semi-isolated fashion, whereas on the intra-population level the candidate classifiers co-evolve competitively with the training data samples. The final classifier is, after the termination of the evolutionary process, a voting committee composed by the best members of all the populations.

Two main contributions result from this thesis: (i) a new evolutionary-algorithm model for data classification, as described above; and (ii) a sophisticated implementation of this algorithm, freely available, to be used in high performance environments.

# Sumário

<b>Lista de Figuras</b>	<b>xiii</b>
<b>Lista de Tabelas</b>	<b>xv</b>
<b>Lista de Símbolos</b>	<b>xvii</b>
<b>1 Introdução</b>	<b>1</b>
1.1 A Tarefa de Classificação de Dados . . . . .	2
1.2 Combinação de Classificadores . . . . .	4
1.2.1 Abordagens para Combinação de Classificadores . . . . .	5
1.3 Organização da Tese . . . . .	6
<b>2 Programação Genética — PG</b>	<b>8</b>
2.1 Introdução . . . . .	8
2.2 Representação dos Programas . . . . .	11
2.3 Conjuntos Primitivos de Funções e Terminais . . . . .	12
2.3.1 Suficiência . . . . .	13
2.3.2 Consistência . . . . .	14
2.4 Criação da População . . . . .	14
2.5 Avaliação dos indivíduos . . . . .	15

2.6	Critério de parada . . . . .	16
2.7	Esquemas de seleção . . . . .	17
2.8	Operadores genéticos . . . . .	18
2.8.1	Cruzamento . . . . .	18
2.8.2	Mutação . . . . .	18
2.9	Métodos de Reprodução . . . . .	19
<b>3</b>	<b>O Algoritmo <i>PGMC</i></b>	<b>21</b>
3.1	<i>Bagging</i> e <i>Boosting</i> . . . . .	21
3.1.1	<i>Bagging</i> . . . . .	22
3.1.1.1	<i>Bootstrap</i> : Formação dos Conjuntos de Treinamento	22
3.1.1.2	Formação do Classificador Final . . . . .	23
3.1.2	<i>Boosting</i> . . . . .	24
3.1.2.1	Formação e Manutenção dos Conjuntos de Treina- mento . . . . .	24
3.1.2.2	Formação do Classificador Final . . . . .	24
3.1.3	Comparação entre os Métodos . . . . .	25
3.2	Estado da Arte . . . . .	26
3.3	<i>PGMC — Programação Genética Multi-populacional e Co-evolucionária</i>	28
3.3.1	Introdução . . . . .	28
3.3.1.1	Criação dos Indivíduos Classificadores . . . . .	31
3.3.2	Co-evolução Competitiva Amostra-Classificador . . . . .	32
3.3.2.1	Aptidão Cumulativa . . . . .	34
3.3.2.2	O Algoritmo Co-evolucionário . . . . .	35

3.3.3	Modelo de Paralelismo por Ilhas . . . . .	39
3.3.4	Método de Amostragem . . . . .	43
3.3.5	Política de Combinação dos Classificadores . . . . .	44
<b>4</b>	<b>Implementação Computacional</b>	<b>47</b>
4.1	Introdução . . . . .	47
4.1.1	A Importância Científica do Software Livre . . . . .	48
4.2	Características e Funcionalidades . . . . .	49
4.2.1	Representação Formal por Gramática . . . . .	52
4.2.1.1	Definição Formal . . . . .	54
4.2.1.2	Integração com a Programação Genética . . . . .	56
4.2.1.3	Gramáticas predefinidas no <i>learner</i> . . . . .	58
4.2.1.4	Meta-linguagem de definição de gramáticas . . . . .	60
4.2.2	Tratamento dos Tipos de Atributos . . . . .	61
4.2.3	Formato da Base de Dados . . . . .	62
4.2.4	Evolução Sequencial e Distribuída . . . . .	63
4.2.4.1	Topologia de Ilhas . . . . .	64
4.2.5	Evolução Canônica e Co-evolucionária . . . . .	66
4.2.6	Método de Combinação de Classificadores . . . . .	67
4.2.7	Particionamento da Base de Dados . . . . .	67
4.2.8	Controle de Complexidade – <i>bloat</i> . . . . .	68
4.2.8.1	Considerações Técnicas . . . . .	72
4.2.9	Especificação de Parâmetros . . . . .	73
<b>5</b>	<b>Experimentos</b>	<b>74</b>

5.1	Ambiente Computacional de Alto Desempenho . . . . .	75
5.2	Metodologia . . . . .	75
5.2.1	Objetivo . . . . .	75
5.2.2	Avaliação da Qualidade dos Classificadores . . . . .	76
5.2.2.1	Limitações . . . . .	77
5.3	Parâmetros . . . . .	78
5.3.1	Gramática . . . . .	79
5.4	Apresentação das Bases de Dados . . . . .	79
5.4.1	<i>abalone</i> . . . . .	81
5.4.2	<i>allhypo</i> . . . . .	81
5.4.3	<i>arcene</i> . . . . .	81
5.4.4	<i>geochemical</i> . . . . .	82
5.4.5	<i>kr-vs-kp</i> . . . . .	82
5.4.6	<i>optdigits</i> . . . . .	82
5.4.7	<i>segmentation</i> . . . . .	83
5.4.8	<i>splice</i> . . . . .	83
5.4.9	<i>waveform</i> . . . . .	83
5.5	Resultados . . . . .	84
5.5.1	Análise e Discussão . . . . .	95
5.5.2	Particionamento do Conjunto de Treinamento . . . . .	96
5.5.2.1	Análise e Discussão . . . . .	103
5.6	Conclusões . . . . .	104

6.1	Trabalhos Futuros . . . . .	107
	<b>Referências Bibliográficas</b>	<b>110</b>
	<b>A Síntese dos Experimentos</b>	<b>121</b>
A.1	Experimentos <i>Sem</i> Particionamento . . . . .	121
A.1.1	Taxas de Erro sobre o Conjunto de Teste . . . . .	121
A.1.2	Taxas de Erro sobre o Conjunto de Treinamento . . . . .	128
A.2	Experimentos <i>Com</i> Particionamento . . . . .	134
A.2.1	Taxas de Erro sobre o Conjunto de Teste . . . . .	134
A.2.2	Taxas de Erro sobre o Conjunto de Treinamento . . . . .	141

# Lista de Figuras

2.1	Fluxograma do funcionamento da programação genética . . . . .	10
2.2	Indivíduos representados pela estrutura árvore . . . . .	12
2.3	Cruzamento padrão . . . . .	19
2.4	Mutação padrão . . . . .	20
3.1	Esquema gráfico da PG multi-populacional e co-evolucionária. . . . .	30
3.2	Esquema gráfico da competição entre classificadores e amostras. . . . .	33
4.1	Esquema gráfico dos módulos <i>learner</i> e <i>classifier</i> . . . . .	49
4.2	Cruzamento gramatical . . . . .	58
4.3	Mutação gramatical . . . . .	59
4.4	Topologia de ilhas I . . . . .	64
4.5	Topologia de ilhas II . . . . .	65
4.6	Topologia de ilhas III . . . . .	66
5.1	<i>Abalone</i> : curva de erro no conjunto de teste. . . . .	85
5.2	<i>Abalone</i> : curva de erro no conjunto de treinamento. . . . .	85
5.3	<i>Allhypo</i> : curva de erro no conjunto de teste. . . . .	86
5.4	<i>Allhypo</i> : curva de erro no conjunto de treinamento. . . . .	87

5.5	<i>Arcene</i> : curva de erro no conjunto de teste. . . . .	87
5.6	<i>Arcene</i> : curva de erro no conjunto de treinamento. . . . .	88
5.7	<i>Geochemical</i> : curva de erro no conjunto de teste. . . . .	89
5.8	<i>Geochemical</i> : curva de erro no conjunto de treinamento. . . . .	89
5.9	<i>Kr-vs-kp</i> : curva de erro no conjunto de teste. . . . .	90
5.10	<i>Kr-vs-kp</i> : curva de erro no conjunto de treinamento. . . . .	91
5.11	<i>Optdigits</i> : curva de erro no conjunto de teste. . . . .	91
5.12	<i>Optdigits</i> : curva de erro no conjunto de treinamento. . . . .	92
5.13	<i>Segmentation</i> : curva de erro no conjunto de teste. . . . .	93
5.14	<i>Segmentation</i> : curva de erro no conjunto de treinamento. . . . .	93
5.15	<i>Splice</i> : curva de erro no conjunto de teste. . . . .	94
5.16	<i>Splice</i> : curva de erro no conjunto de treinamento. . . . .	94
5.17	<i>Waveform</i> : curva de erro no conjunto de teste. . . . .	95
5.18	<i>Waveform</i> : curva de erro no conjunto de treinamento. . . . .	96
5.19	<i>Abalone</i> : curva de erro no conjunto de teste (versão particionada). . .	97
5.20	<i>Allhypo</i> : curva de erro no conjunto de teste (versão particionada). . .	98
5.21	<i>Arcene</i> : curva de erro no conjunto de teste (versão particionada). . .	99
5.22	<i>Geochemical</i> : curva de erro no conjunto de teste (versão particionada). . .	99
5.23	<i>Kr-vs-kp</i> : curva de erro no conjunto de teste (versão particionada). . .	100
5.24	<i>Optdigits</i> : curva de erro no conjunto de teste (versão particionada). . .	101
5.25	<i>Segmentation</i> : curva de erro no conjunto de teste (versão particionada). . .	101
5.26	<i>Splice</i> : curva de erro no conjunto de teste (versão particionada). . . .	102
5.27	<i>Waveform</i> : curva de erro no conjunto de teste (versão particionada). . .	102

# Lista de Tabelas

5.1	Parâmetros fixos dos experimentos. . . . .	79
5.2	Síntese das características das bases de dados. . . . .	80

# Lista de Algoritmos

3.1	Princípio geral do <i>bagging</i> e do <i>boosting</i> . . . . .	22
3.2	Formação do conjunto $\mathcal{A}'$ sobre $\mathcal{A}$ via <i>bootstrap</i> . . . . .	23
3.3	<i>Boosting</i> : atualização dos pesos da iteração $n$ . . . . .	25
3.4	<b>PGMC</b> – PG Multi-populacional e Co-evolucionária . . . . .	31
3.5	Processo co-evolucionário com aptidão cumulativa . . . . .	35
4.1	Atualização <i>online</i> de ambas variância e covariância . . . . .	73
4.2	Atualização <i>online</i> da covariância . . . . .	73

# Lista de Símbolos

$A_j^{P_n}$	$j$ -ésima amostra de dados da população $P_n$ , p. 30
$C_*^{P_n}$	Melhor classificador obtido pela população $P_n$ , p. 29
$C_i^{P_n}$	$i$ -ésimo classificador da população $P_n$ , p. 30
$C_{final}$	Classificador final agregado, p. 30
$G$	Número de gerações, p. 30
$I$	Número de indivíduos (tamanho da população), p. 30
$J$	Quantidade de amostras de dados, p. 30
$N$	Número de iterações/populações, p. 21
$P_n$	$n$ -ésima população (“ilha”), p. 29
$Y$	Conjunto de classes, p. 23
$\beta$	Pressão de seleção das amostras de dados, p. 39
$\ell$	Tamanho atual de uma árvore, p. 72
$\ell'$	Novo tamanho de uma árvore, p. 72
$\epsilon$	Somatório ponderado dos erros de um classificador, p. 25
$\eta$	Tamanho do histórico de confrontos, p. 36
$\gamma$	Tamanho do ciclo co-evolucionário, p. 36
$\mathbb{F}$	Conjunto de funções (operadores), p. 12

$\mathbb{T}$	Conjunto de terminais (operadores), p. 12
$\mathcal{P}_n$	Distribuição de pesos da $n$ -ésima população/iteração, p. 25
$\mu_{\vec{\ell}}$	Média das aptidões dos classificadores, p. 71
$\mu_{\vec{f}}$	Média dos tamanhos das árvores, p. 71
$\tau$	Tamanho médio esperado das árvores classificadoras, p. 71
$\vec{\ell}$	Vetor de tamanhos das árvores da população, p. 71
$\vec{f}$	Vetor de aptidões dos indivíduos classificadores, p. 71
$c$	Coefficiente de penalização por complexidade, p. 70
$f$	Aptidão atual de um classificador, p. 72
$f'$	Nova aptidão de um classificador, p. 72
$y_{A_j^{P_n}}$	Classe esperada da amostra $A_j^{P_n}$ , p. 45

# Capítulo 1

## Introdução

Dominar e compreender inteiramente um problema é uma tarefa normalmente árdua para os algoritmos de aprendizagem baseados em otimização; muito comumente o que se obtém é o entendimento sobre porções do seu domínio, isto é, a obtenção de soluções ótimas localmente. Esta frustrante constatação, por outro lado, dá margem para o instigante questionamento: *seria possível decompor o problema de tal maneira que suas partes fossem mais facilmente resolvidas e então recompô-las na forma de uma única solução representando todo o problema original?*

Esta visão lembra o conceito de *divisão-e-conquista*, e possui dois grandes desafios: a decomposição do problema, isto é, como explorar seus subespaços isoladamente; e como unificar conhecimentos individuais em uma inteligência mais completa, idealmente como a soma integral das partes. Nesse espírito enquadram-se os métodos de combinação de soluções parciais, que procuram resolver os desafios mencionados.

Esta tese propõe uma nova abordagem de combinação de soluções parciais destinada à tarefa de classificação de dados, sendo ela, portanto, um combinador de estruturas classificadoras. A proposta baseia-se na meta-heurística evolucionária denominada programação genética, e explora os recursos de múltiplas populações e co-evolução, sendo inspirada ainda nos conceitos dos populares combinadores *bagging* [1] e *boosting* [2, 3]. Na abordagem proposta, o conjunto original de treinamento é apropriadamente distribuído entre as populações, onde cada uma, em paralelo, co-evolui competitivamente com suas respectivas amostras de treinamento em uma

espécie de “corrida armamentista”<sup>1</sup>. Ao final do processo evolutivo elege-se em cada população um representante—obtendo-se assim um conjunto de representantes—que são então devidamente reunidos para constituírem a solução final.

O restante deste capítulo dedica-se à introdução de conceitos elementares relativos a esta tese. Na próxima seção é apresentada a tarefa de classificação de dados e suas aplicações; em seguida, discute-se sobre o método de combinação de classificadores; finalmente, a organização do conteúdo desta tese é sintetizada na última seção deste capítulo.

## 1.1 A Tarefa de Classificação de Dados

Mapear características em certos padrões discretos é uma das operações mais comumente praticadas no nível cognitivo, pois é reconhecidamente um meio de auxílio à compreensão do ambiente. Igualmente importante para o entendimento da natureza de bases de dados é o descobrimento de mapeamentos entre atributos e classes, a fim de expor o princípio de formação desses dados. Esta tarefa pertence à área de mineração de dados, e é chamada de *classificação de dados*.

A classificação de dados pode ser descrita como a tarefa que, fornecido um conjunto de dados como treinamento, visa construir uma entidade com a propriedade de mapear atributos de entrada em um valor discreto que representa a classe/categoria de uma dada amostra. Esta entidade é denominada *classificador* ou *preditor*. A construção do classificador, embora possa ser feita servindo-se dos mais variados algoritmos, é um processo de treinamento que se baseia em uma coleção prévia de exemplos—são conhecidas as classes para cada conjunto de atributos—, portanto, a classificação de dados é uma tarefa de aprendizado *supervisionado*.

Existem duas maneiras de aplicação da classificação de dados; com o treina-

---

<sup>1</sup>*Corrida armamentista* é um termo que foi cunhado durante a Guerra Fria, que descrevia a co-evolução armamentista entre os Estados Unidos e a União Soviética. Cada uma das nações objetivava ter em mãos o maior poderio bélico, visando a dominação em um possível confronto militar. Para tanto, além do recrutamento interno para pesquisa e desenvolvimento de novas armas de destruição, as superpotências também investiam em tecnologia de espionagem, possibilitando-se assim conhecer o potencial do inimigo e então focar-se em meios de anulá-lo ou superá-lo. O paralelo com a natureza é direto; a soberania (“aptidão”) de uma das nações é inversamente proporcional ao poder bélico de outra.

mento de classificadores: (i) focando-se na habilidade de realização de predições automatizadas de novas amostras; ou (ii) concentrando-se na construção de modelos compactos e legíveis que permitem o entendimento/extração das regras que governam a base de dados. É sempre desejável que ambos os objetivos sejam alcançados, mas em muitos problemas eles se mostram conflitantes, ou seja, ou se obtém um classificador preciso ou compacto/legível, mas não ambos ao mesmo tempo.

O assunto da presente tese, e portanto as contribuições desta, concentram-se na abordagem do primeiro modo de aplicação da tarefa de classificação de dados, isto é, na construção de soluções que visam primordialmente a *habilidade de predição* de novas amostras de dados. Embora os classificadores produzidos pelo algoritmo proposto sejam simbólicos e, em princípio, legíveis, é próprio dos métodos de combinação de soluções a degradação da interpretabilidade das soluções, devido à complexidade resultante da agregação de uma coleção de soluções parciais.

## Aplicações

A classificação de dados é uma atividade que goza de expressiva popularidade, decorrente de sua importância e grande aplicabilidade em várias áreas do conhecimento. A relação apresentada a seguir lista, em caráter ilustrativo, uma pequena parcela dos domínios e aplicações nos quais a tarefa de classificação tem sido bem sucedida.

**Sistemas petrolíferos** Partindo-se de uma base de dados geológicos, pode-se através da classificação de dados prever as chances de sucesso na procura de reservas de óleo e gás natural em uma determinada região; ainda, por meio de dados de cromatografia gasosa pode ser possível classificar diferentes óleos.

**Medicina** Diagnóstico automatizado de doenças mediante informações de exames do paciente; ou, análise de imagens médicas para detecção e avaliação do grau de tumores.

**Economia, comércio e indústria** Análise de crédito e cálculo de risco baseando-se em informações históricas e vigentes acerca do tomador de crédito; ou então, demarcação de segmentos de clientes/negócio potencialmente lucrativos.

**Reconhecimento de padrões** Reconhecimento óptico de caracteres digitais ou manuscritos; também, biometria e classificação de padrões em imagens.

## 1.2 Combinação de Classificadores

Usualmente as técnicas de aprendizado aplicadas à tarefa de classificação de dados limitam-se em *aproximar* o conhecimento acerca do problema, seja em função de uma amostragem insuficiente ou ruidosa, ou simplesmente por restrição da própria técnica. Também é característica marcante nestes algoritmos de aprendizagem a possibilidade da geração de *diferentes aproximações* para um mesmo problema por meio de perturbações nos dados de treinamento ou nos parâmetros de iniciação dos algoritmos.

Estas várias aproximações, cada qual possivelmente retendo conhecimento sobre diferentes partes do domínio, se devidamente agregadas formam uma inteligência coletiva potencialmente mais precisa e completa do que quando representada individualmente. Dietterich [4] aponta três razões fundamentais, embora relacionadas entre si, para o funcionamento dos métodos de combinação de classificadores: *estatística*, *computacional*, e *representativa*.

A razão estatística revela-se, sobretudo, quando o conjunto de treinamento é relativamente pequeno frente à real caracterização da base de dados—o espaço de hipóteses. Dessa forma, as amostras de treinamento fornecem pouco poder de discriminação, e portanto diferentes classificadores podem ser obtidos sob uma mesma alta precisão de classificação. Em decorrência disto, a combinação dessas soluções faz com que um classificador mais *centrado* seja obtido, que estatisticamente representa uma melhor aproximação para o verdadeiro classificador procurado.

Por sua vez, a razão computacional surge em decorrência da constatação de que na prática é comumente inviável a obtenção de soluções ótimas, mesmo na disponibilidade de dados suficientes de treinamento.<sup>2</sup> Em consequência disso e, supondo-se que as soluções sub-ótimas, em execuções independentes, sejam obtidas

---

<sup>2</sup>Tipicamente, a não obtenção de soluções ótimas por algoritmos de classificação deve-se: (i) ao mecanismo de busca inerentemente local; (ii) falta de garantias de convergência global; ou, na existência de garantias de convergência global, (iii) custo computacional proibitivo.

com relativa diversidade (sub-ótimos distintos), a combinação dessas soluções tem o efeito de produzir uma solução *mediana* a estas, que tende a situar-se próximo à solução ótima.

Finalmente, a terceira razão apontada considera casos em que a representação das soluções candidatas de um algoritmo de classificação pode, eventualmente, ser incapaz de descrever exatamente a real solução procurada, limitando-se à obtenção de soluções vizinhas (sub-ótimas). Nesse caso, com base no mesmo princípio das duas outras razões, a combinação de classificadores pode, por meio da *ponderação* das predições dessas soluções vizinhas, aproximar a predição da verdadeira solução.

### 1.2.1 Abordagens para Combinação de Classificadores

No contexto da classificação de dados, pode-se dizer que a exploração desse tema efetivamente iniciou-se na década de 90 com o desenvolvimento de dois métodos de combinação de classificadores, *bagging* [1] e *boosting* [2, 3]. Eles destacam-se pela simplicidade e principalmente pelos expressivos resultados, produzindo melhoras significativas sobre as técnicas existentes de classificação de dados no que tange a precisão e robustez. Ambos são meta-algoritmos, isto é, atuam sobre outros algoritmos, e compartilham o mesmo fundamento: *treinar uma coleção de classificadores cada qual com um conjunto de treinamento estrategicamente definido e, ao término, constituir um comitê votante—que decide a classe final de uma dada instância—governado por uma certa política de votação*. O treinamento dos classificadores fica a cargo de terceiros, como algoritmos de árvore de decisão ou redes neurais; ao *bagging* e *boosting* cabem a escolha cautelosa do conjunto de treinamento a cada rodada e o modo de agregação das soluções parciais a fim de produzir uma solução final aprimorada. A diferença entre os dois está em (i) como gerenciar o conjunto de treinamento para cada classificador; e (ii) a política de votação.<sup>3</sup>

Inspirados no *bagging* e *boosting*, e de uma forma mais ampla, no conceito de combinação de classificadores, surgiram muitos trabalhos buscando inserir essas ideias diretamente em outras técnicas (algoritmos de classificação de dados), tornando-as

---

<sup>3</sup>A Seção 3.1 apresenta o *bagging* e *boosting* e suas diferenças.

nativamente integradas.<sup>4</sup> A estruturação tipo meta-algoritmo é simples e prática, mas ao mesmo tempo compromete a flexibilização e diversidade à medida que a comunicação entre os algoritmos é restrita. Diferentemente, a integração forte permite o fino ajuste e abre um leque de novas possibilidades de interação entre as técnicas.

Nesse sentido, as abordagens com vocação para hibridização são um caminho tentador para se alcançar a boa união de técnicas. Em especial, destacam-se os algoritmos de classificação de dados baseados em programação genética, uma linhagem da computação evolucionária com enfoque na evolução de “programas” de computador (programas classificadores, no caso), que, assim como sua família, é uma abordagem bastante afável à integração. A natureza populacional da programação genética, isto é, a existência concorrente de várias soluções candidatas, sugere que a combinação de classificadores possa ser convenientemente obtida usando-se conjuntos de indivíduos ou conjuntos de grupos de indivíduos. Sob este raciocínio, uma possibilidade direta é evoluir sub-populações e então eleger em cada população um classificador votante, formando-se ao final um comitê de votação. Outras formas de integração neste contexto incluem, por exemplo, o uso de *nichos* [5], *otimização multi-objetivo* [6] e técnicas *pareto-co-evolucionárias* [7].

### 1.3 Organização da Tese

O conteúdo desta tese está organizado da seguinte maneira. Introduce-se no Capítulo 2, de maneira geral e abrangente, o ramo da computação evolucionária denominado programação genética, cuja especialidade é a indução de programas de computador em linguagens arbitrárias via processo simulado de evolução por seleção natural. A programação genética figura como um dos pilares desse trabalho, o alicerce sobre o qual os algoritmos e as implementações são construídos.

Propõe-se no Capítulo 3, como uma das principais contribuições desta tese, o algoritmo que desenvolve a ideia de combinação de classificadores, inspirado nos conceitos do *bagging* e *boosting*, por meio de cooperação entre múltiplas populações semi-conectadas e co-evolução competitiva no nível intra-populacional.

---

<sup>4</sup>A Seção 3.2 discute alguns desses trabalhos no âmbito da programação genética.

Uma implementação do algoritmo proposto e dos populares *bagging* e *boosting* sobre a programação genética é exposta no Capítulo 4. Esta implementação consta como a segunda distinta contribuição deste trabalho e está disponibilizada livremente e sem restrições.

No Capítulo 5 são realizados lotes de experimentos computacionais controlados que demonstram a superioridade do algoritmo proposto sobre os tradicionais *bagging* e *boosting* no domínio da programação genética. Os algoritmos são aplicados em uma variada coleção de problemas de classificação de dados.

Finalmente, no Capítulo 6 são apresentadas as considerações finais e também direções promissoras de trabalhos futuros ligadas a este trabalho.

# Capítulo 2

## Programação Genética — PG

Este capítulo apresenta a meta-heurística denominada programação genética, em sua formulação tradicional<sup>1</sup> [8], que provê o terreno fundamental sobre o qual desenvolvem-se as contribuições desta tese.

### 2.1 Introdução

A programação genética, cujo desenvolvimento é atribuído a John Koza [8], é uma meta-heurística estocástica de otimização global baseada no princípio darwiniano de seleção natural, sendo, pois, uma abordagem da computação evolucionária. A PG destina-se à evolução de programas de computador em linguagens arbitrárias, em outras palavras, a PG otimiza estruturas funcionais capazes de realizar operações, sejam elas lógicas, aritméticas, condicionais e de desvios, que normalmente mapeiam *entradas* em *saídas*.

Nesta concepção, submete-se uma população contendo um determinado número de indivíduos—programas candidatos criados aleatoriamente—ao processo simulado de evolução segundo a seleção natural. Neste processo iterativo figuram a cada geração: a seleção de indivíduos promissores para procriação, a formação de seus descendentes por meio de operações genéticas como cruzamento e mutação, e finalmente a inserção destes novos indivíduos na população, marcando-se o início de uma nova geração. Muito embora não existam garantias de progresso e tampouco de obtenção de soluções ótimas, como característico em qualquer meta-heurística, essa dinâmica

---

<sup>1</sup>Referenciada nesta tese como *programação genética canônica*.

tende, ao longo das gerações, a produzir soluções candidatas incrementalmente mais adaptadas.

Destacam-se como características típicas da programação genética as seguintes qualidades:

- *Robustez* – o modelo populacional e o processo estocástico são os dois principais fatores por trás da programação genética responsáveis pela solidificação da tolerância a ruídos.
- *Exigência de pouco conhecimento* sobre o domínio – os requisitos acerca do domínio de aplicação podem ser relaxados; basicamente, espera-se apenas uma função capaz de comparar a qualidade relativa dos indivíduos com certa precisão.
- Possui *paralelismo natural* – as demandas computacionais da programação genética podem ser trivialmente particionadas em vários níveis, com emprego ou não de modelos distribuídos bioinspirados, permitindo-se assim redução no tempo de execução e/ou escalabilidade para problemas mais complexos.
- Produz *soluções simbólicas* – usualmente as soluções obtidas pela programação genética são imediatamente legíveis e interpretáveis (“código-fonte disponível”), em decorrência do processo evolutivo atuar diretamente no nível simbólico das estruturas.
- É facilmente *extensível/modificável* – a programação genética é uma meta-heurística extremamente versátil, e dessa forma admite, por exemplo, hibridizações (funcionamento junto a outras técnicas), integração de outros modelos evolucionários (p. ex., co-evolução, nichos, múltiplos objetivos) e as mais diversas representações e linguagens para os programas (como representação formal por gramáticas).

O processo evolutivo da programação genética canônica pode ser visualizado pelo fluxograma da Figura 2.1. Como exposto pela figura, os componentes e etapas da programação genética, que são discutidos no decorrer deste capítulo, podem ser sintetizados por:

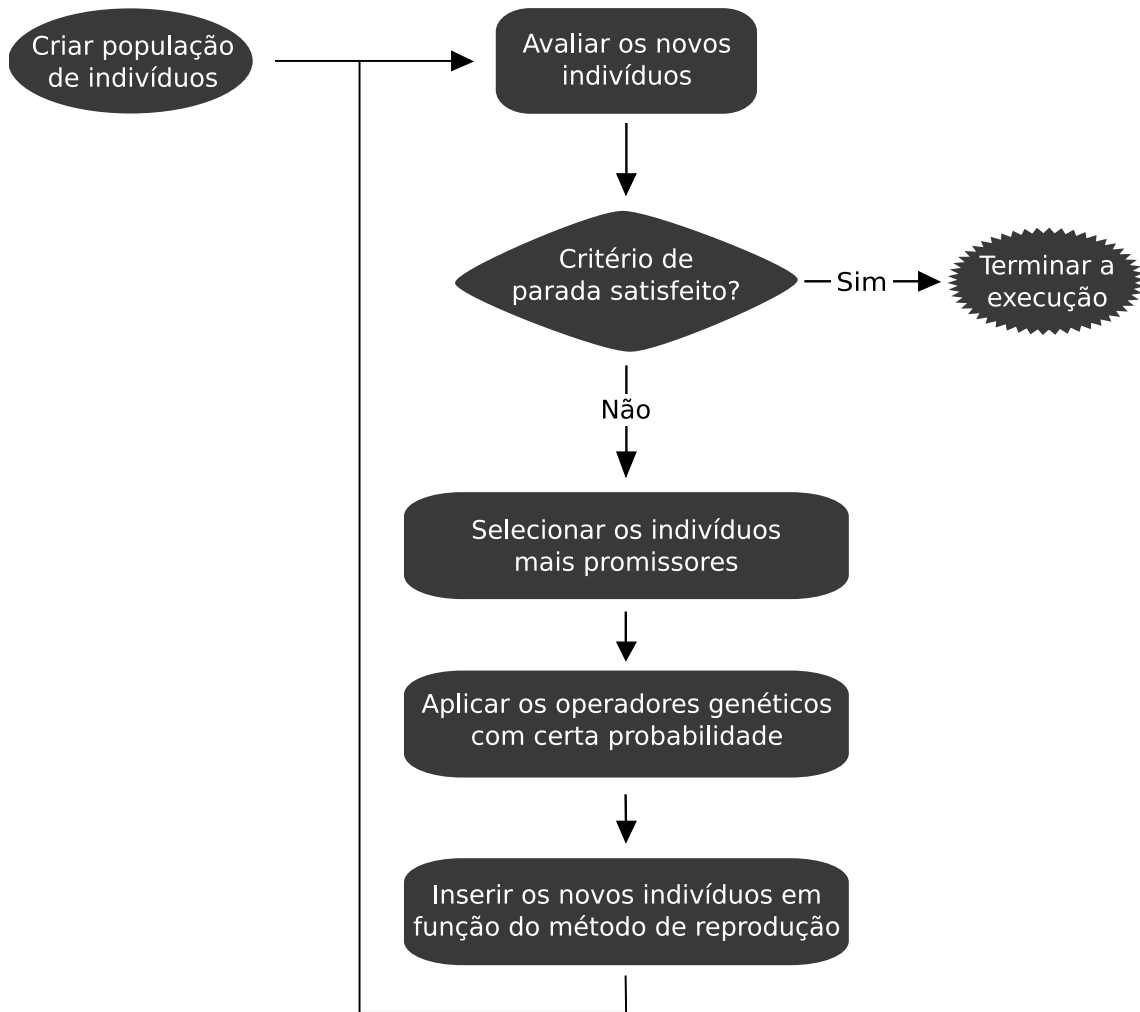


Figura 2.1: Fluxograma do funcionamento da programação genética

1. *Criação da população* – determina quem e como são formados os primeiros indivíduos candidatos. Para tanto, dois conceitos devem estar definidos:
  - (a) representação dos programas – a estrutura funcional na qual o processo evolutivo atuará, isto é, a entidade que será submetida às avaliações e operações genéticas.
  - (b) conjuntos primitivos de funções e terminais – estabelece as instruções básicas (“palavras-chave”) da linguagem dos programas que, quando devidamente combinadas, expressam a solução esperada.
2. *Avaliação dos indivíduos* – função que mede quão bem um determinado indivíduo executa a tarefa do problema, em outras palavras, é a função objetivo a ser maximizada.
3. *Critério de parada* – determina se o processo deve ser interrompido, normal-

mente por obtenção de uma solução satisfatória ou por limites de tempo de execução.

4. *Esquemas de seleção* – definem como os indivíduos mais bem adaptados são favorecidos na disputa pela procriação.
5. *Operadores genéticos* – são as técnicas de criação de novos indivíduos com base no material genético de seus genitores. Tipicamente subdividem-se em:
  - (a) cruzamento – recombinação do material genético de dois indivíduos.
  - (b) mutação – deriva um indivíduo como uma variação de um outro.
6. *Métodos de reprodução* – estipulam como os descendentes serão inseridos na população.

## 2.2 Representação dos Programas

Existe uma infinidade de estruturas capazes de representar programas de computador no âmbito da programação genética.<sup>2</sup> Há no entanto três principais grupos conceituais de representação: *linear* [9], por *árvore* [8] ou por *grafos* [10, 11, 12].

Dentre os grupos, a representação por árvore é a mais popular, status este possivelmente decorrente da sua tradição de uso na programação genética, aliada ao poder/espontaneidade de expressão de programas e relativa simplicidade. Todavia, esta classificação é *conceitual*, e difere da *representação computacional*; por exemplo, muito embora a árvore seja uma estrutura naturalmente hierárquica, ela pode ser implementada e operada linearmente sem que haja violação de suas propriedades [13, 14]. Ainda, um terceiro nível de classificação pode ser introduzido: a *representação semântica*. Neste, a representação conceitual é complementada com regras/restrições que governam como a estrutura em questão pode ser operada; pode-se citar como membros desta classificação, por exemplo, a representação formal por gramática sobre a estrutura árvore (Seção 4.2.1) e sobre a estrutura linear [15], e restrições de tipo para a estrutura árvore [16].

---

<sup>2</sup>Essas estruturas são muitas vezes referidas por *cromossoma* ou *genoma* do indivíduo.

A despeito do extenso leque de opções de representação, considera-se neste capítulo—e de uma maneira geral, nesta tese—a representação pela estrutura de dados *árvore*.<sup>3</sup> Nesta representação, o cromossoma de um indivíduo apresenta um aspecto gráfico como exemplificado na Figura 2.2. O indivíduo **A** descreve a expressão:

$$\text{se } A \wedge B \text{ então } C, \text{ senão } (3.14 \times X)$$

enquanto que o indivíduo **B** a função matemática:

$$\sin(\sqrt{0.987} - \cos(X \times Y))$$

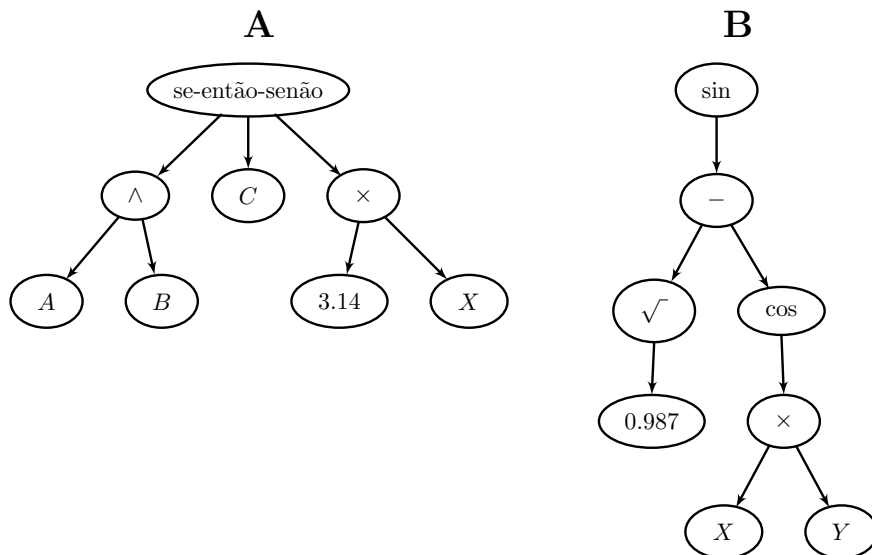


Figura 2.2: Indivíduos representados pela estrutura árvore

## 2.3 Conjuntos Primitivos de Funções e Terminais

Essenciais ao funcionamento da programação genética, como originalmente concebida, são os denominados conjunto de *funções* ( $\mathbb{F}$ ) e *terminais* ( $\mathbb{T}$ ), que são relativos ao domínio da aplicação. São eles que definem as ferramentas (as primitivas) que estarão a disposição do processo evolucionário para a construção de estruturas mais complexas, isto é, definem a linguagem—e o feitiço do espaço de busca—onde a evolução poderá explorar.

<sup>3</sup>Está além do escopo desta tese uma investigação comparativa entre as diversas formas de representação.

O conjunto  $\mathbb{F}$  é responsável pelo fornecimento de funções que requerem argumentos, ou seja, os *operadores*. Exemplos incluem operadores matemáticos ( $+$ ,  $-$ ,  $\times$ ,  $\div$ ,  $\sin$ ,  $\cos$ ,  $\sqrt{\quad}$ ,  $\log$ ,  $x^y$ ), lógicos ( $\vee$ ,  $\wedge$ ,  $\neg$ ), relacionais ( $<$ ,  $>$ ,  $=$ ), condicionais (se  $x$  então  $y$  senão  $z$ ) e laços iterativos (enquanto  $x$  faça  $y$ ).

Por sua vez, o conjunto  $\mathbb{T}$  provê os *operandos*, isto é, variáveis/atributos ( $x$ ,  $y$ , *idade*, *salário*), constantes ( $\pi$ ,  $e$ ) e funções que não requerem argumentos (*random*, *tempo percorrido*).

Os conjuntos de funções e terminais podem ser livremente, e de forma recursiva, combinados; a única restrição é que seja obedecida a aridade<sup>4</sup> de cada primitiva. Portanto, todo fragmento ou estrutura de um programa na população, em qualquer estágio do processo evolutivo, nada mais é do que uma combinação particular entre elementos de  $\mathbb{F}$  e  $\mathbb{T}$ .

### 2.3.1 Suficiência

Do fato de que qualquer estrutura da população é um arranjo particular de  $\mathbb{F}$  e  $\mathbb{T}$ , decorre que é condição *necessária* para a obtenção da solução esperada que esta possa ser expressa como uma combinação das primitivas definidas para o problema. Esta óbvia constatação é conhecida como *propriedade de suficiência*, e diz que os elementos dos conjuntos  $\mathbb{F}$  e  $\mathbb{T}$  devem ser suficientes para, em ao menos uma combinação específica, descrever a solução esperada.

Na prática, entretanto, a propriedade de suficiência não soa tão óbvia, devido ao fato de que em muitos problemas—especialmente os reais e de grande porte—não se conhece o aspecto da solução, e dessa forma pouco se pode cogitar a respeito da formação dos conjuntos de funções e terminais. Neste cenário, é tentador adicionar aos conjuntos tantas primitivas quanto se possa; contudo, tal procedimento aumenta enormemente o espaço de busca, e pode, portanto, seriamente degradar o progresso da evolução.

---

<sup>4</sup>Aridade é o número requerido de argumentos. Os terminais em  $\mathbb{T}$  têm sempre aridade *zero*.

### 2.3.2 Consistência

Uma segunda propriedade que concerne aos conjuntos de primitivas é a denominada *consistência*. Consistência é a condição de integridade, e requer que cada operador do conjunto de funções seja suficientemente versátil e robusto para aceitar como argumento qualquer terminal em  $\mathbb{T}$  ou valores de retorno de qualquer função em  $\mathbb{F}$ . Em outras palavras, os conjuntos de terminais e funções têm que ser tais que, quaisquer combinações entre  $\mathbb{F}$  e  $\mathbb{T}$ , ou  $\mathbb{F}$  e ele próprio, desde que respeitadas as aridades, sejam válidas.

A consistência é uma propriedade demasiadamente relaxada no que se refere às relações e restrições entre as primitivas, isto é, no que tange a definição da linguagem dos programas. Ao mesmo tempo em que os conceitos dos conjuntos de funções e terminais da programação genética canônica proveem simplicidade e uniformidade, a ausência de restrições causa: (i) crescimento não justificado do espaço de busca—regiões conceitualmente sem sentido passam a fazer parte dele; e (ii) dificuldade de satisfazer a condição de consistência quando estão envolvidos diferentes tipos e estrutura de dados, tornando-a inviável ou, na melhor das hipóteses, forçando uma adaptação crassa.<sup>5</sup> Felizmente, para domínios de aplicação onde as deficiências e limitações da noção dos conjuntos de terminais/funções mostram-se preocupantes ou proibitivas, é possível adotar representações mais sofisticadas, como, entre outras [16, 15], a representação formal por gramática [17], que é descrita na Seção 4.2.1.

## 2.4 Criação da População

A criação da população visa formar uma amostragem representativa do espaço de programas, que seja preferencialmente bem distribuída e sem tendenciosidade a favor de configurações particulares. Entretanto, dado que o espaço de possíveis programas é geralmente infinito—quando não se impõe limites acerca do tamanho das soluções—, torna-se impossível amostragens estritamente uniformes; o que se pode fazer, todavia, é tentar evitar grandes distorções [18].

---

<sup>5</sup>Por exemplo, não faz sentido conceitual a multiplicação de valores booleanos, no entanto, seria necessário garantir a factibilidade desta operação para que o processo evolutivo fosse viável.

Na prática, a criação dos indivíduos iniciais, e portanto da população, é realizada de maneira essencialmente *aleatória*, isto é, as instruções básicas dos conjuntos  $\mathbb{F}$  e  $\mathbb{T}$  são recursivamente combinadas ao acaso, em número e complexidade variáveis, na estrutura do indivíduo.<sup>6</sup> Embora o processo de criação seja não-determinístico, é preciso que se respeite a aridade de cada operador; em outras palavras, se uma determinada função possui  $n$  argumentos, então, considerando-se a estrutura árvore, o nó representando este operador terá exatamente  $n$  nós filhos, cada qual constituindo um argumento.

## 2.5 Avaliação dos indivíduos

A avaliação dos indivíduos, que depende fundamentalmente do domínio da aplicação, mede o êxito com que os programas realizam a tarefa alvo, isto é, define a *aptidão*. Em problemas de regressão, por exemplo, interessa saber a discrepância entre os valores esperados para os pontos fornecidos e os valores obtidos pela função sendo avaliada; portanto, uma medida de avaliação poderia ser o inverso da soma do quadrado dos erros:

$$\left[ \sum_{j=1}^J [f(p_j) - y_j]^2 \right]^{-1}$$

onde  $J$  é o número de pontos,  $p_j \in \mathbb{R}^n$  é o  $j$ -ésimo ponto,  $f$  é a função sendo avaliada ( $f : \mathbb{R}^n \mapsto \mathbb{R}$ ), e  $y_j \in \mathbb{R}$  é o valor esperado para o ponto de índice  $j$ .

Já em problemas de classificação de dados o objetivo primário é medir a acurácia de predição sobre um determinado conjunto de amostras de treinamento; dessa forma, a aptidão de uma árvore classificadora  $C$  ( $C : \mathbb{R}^n \mapsto \mathbb{N}$ ), normalizada para o intervalo  $[0, 1]$ , poderia ser dada pela taxa de acertos:

$$\frac{1}{J} \sum_{j=1}^J \mathbb{I}[C(a_j) = y_j]$$

onde  $J$  é o número total de amostras de treinamento,  $a_j \in \mathbb{R}^n$  e  $y_j \in \mathbb{N}$ , respectivamente, a  $j$ -ésima amostra e sua classe esperada, e  $\mathbb{I}[\pi]$  retorna 1 se a condição  $\pi$  é

---

<sup>6</sup>Em certas aplicações, no entanto, conhecendo-se alguns padrões úteis e promissores, é possível introduzi-los diretamente na população inicial, mas é preciso cautela para que estas sementes não causem convergência prematura.

satisfeita ou 0 caso contrário.

No entanto, nesses casos geralmente é útil ponderar a aptidão do indivíduo em função de sua complexidade/tamanho, visando assim a obtenção de soluções mais compactas, legíveis e potencialmente com maior poder de generalização. Uma medida comum é a introdução de um coeficiente de penalização por complexidade, que se propõe a reduzir a aptidão de um indivíduo proporcionalmente ao tamanho de sua estrutura (Seção 4.2.8).

## 2.6 Critério de parada

Idealmente, uma execução da programação genética deveria terminar quando, e somente quando, uma solução ótima fosse encontrada segundo a função de aptidão definida. Infelizmente, dependendo da complexidade do problema e recursos computacionais disponíveis, isto nem sempre é viável. Nesses casos, o relaxamento desta condição no intuito de obtenção de soluções “suficientemente boas” pode ser admissível—ou ser de fato a única alternativa. A implementação deste conceito é feita pela adição de um ou mais critérios de parada, comumente:

- Aptidão dentro da faixa aceitável – o processo é interrompido quando uma solução aproximada é obtida.
- Número máximo de gerações ou avaliações – a execução é interrompida ao se atingir um número pré-estabelecido de gerações ou avaliações.
- Tempo limite de execução – análogo ao item anterior, mas é guiado pelo tempo despendido.
- Estagnação do processo – a execução termina quando é detectada estagnação do processo evolutivo, isto é, quando por um certo período nenhuma solução aprimorada é obtida.

## 2.7 Esquemas de seleção

O esquema de seleção é o instrumento pelo qual os algoritmos evolucionários conduzem a busca para as regiões mais promissoras do espaço e, que, efetivamente, permite o processo de otimização.

A seleção é responsável por escolher os indivíduos da população que deixarão descendentes para a geração seguinte, ou seja, aqueles que participarão das operações genéticas. Portanto, é natural esperar que as funções de seleção favoreçam aqueles indivíduos mais bem adaptados, pois a transmissão de bom material genético potencializa o desenvolvimento de soluções aperfeiçoadas.

A diferença entre a probabilidade de seleção dos melhores indivíduos e a probabilidade de seleção dos piores denomina-se *pressão de seleção*. Quando a pressão de seleção é baixa, indivíduos de menor qualidade terão mais chance de disseminar seus genes; por outro lado, quando a pressão é alta, somente aqueles indivíduos muito bem avaliados, em média, têm oportunidade de ser selecionados. Embora possa parecer que, a princípio, a alta pressão de seleção é sempre vantajosa, na realidade ela pode levar o processo evolucionário rapidamente à indesejável convergência prematura. Isto ocorre, notoriamente, devido à degradação da diversidade na população, já que apenas um pequeno seleto grupo de indivíduos domina o processo.

Embora existam vários esquemas de seleção [19], cada qual diferindo particularmente no que tange a dinâmica da pressão de seleção e custo computacional, uma variação bastante simples, eficiente e conveniente, nomeada *torneio* [20], é comumente empregada na programação genética. Neste esquema,  $k$  indivíduos são selecionados *aleatoriamente* da população, suas  $k$  aptidões são comparadas e o indivíduo detentor da melhor aptidão é selecionado. Nota-se que o valor de  $k$ , de fato, define a pressão de seleção, isto é, quanto maior o número de competidores mais rigorosa a seleção se torna.

## 2.8 Operadores genéticos

A função dos operadores genéticos é construir soluções derivadas a partir de outro(s) indivíduo(s). É através desse processo de experimentação de recombinações e/ou variações sobre padrões promissores que o espaço de busca é explorado.

Os dois principais operadores genéticos, descritos a seguir, são o *cruzamento*, que atua sobre dois indivíduos, e *mutação*, que atua sobre um único indivíduo.

### 2.8.1 Cruzamento

O operador de cruzamento propõe-se a recombinar partes de estruturas previamente selecionadas pelo processo de seleção, na esperança de se derivar estruturas ainda melhores. Devido ao status de operador primário na programação genética, o cruzamento é tipicamente aplicado com alta probabilidade.

Na representação por árvore, o operador de cruzamento comuta sub-árvores, cujos nós raiz são arbitrariamente escolhidos, de dois indivíduos, gerando dessa maneira dois novos descendentes. A Figura 2.3 exemplifica a aplicação do operador de cruzamento sobre os pais **A** e **B**, produzindo os filhos **A'** e **B'**.

### 2.8.2 Mutação

A mutação deriva um novo descendente mediante modificações na estrutura de um indivíduo selecionado. Estas modificações são normalmente de pequena escala (“perturbações”), e visam tanto a exploração de regiões vizinhas àquele indivíduo quanto a (re)introdução de material genético a fim de preservar a diversidade da população. As modificações sobre as estruturas são realizadas, tipicamente, de maneira absolutamente aleatória.<sup>7</sup>

A Figura 2.4 ilustra uma aplicação hipotética do operador canônico de mutação sob a representação por árvore. Primeiramente, uma sub-árvore do indivíduo **A** é escolhida ao acaso, então removida, e finalmente uma nova sub-árvore, criada

---

<sup>7</sup>Mas existem operadores especializados de mutação que agem deterministicamente; por exemplo, operadores de eliminação de redundância ou de simplificação de expressões em árvores.

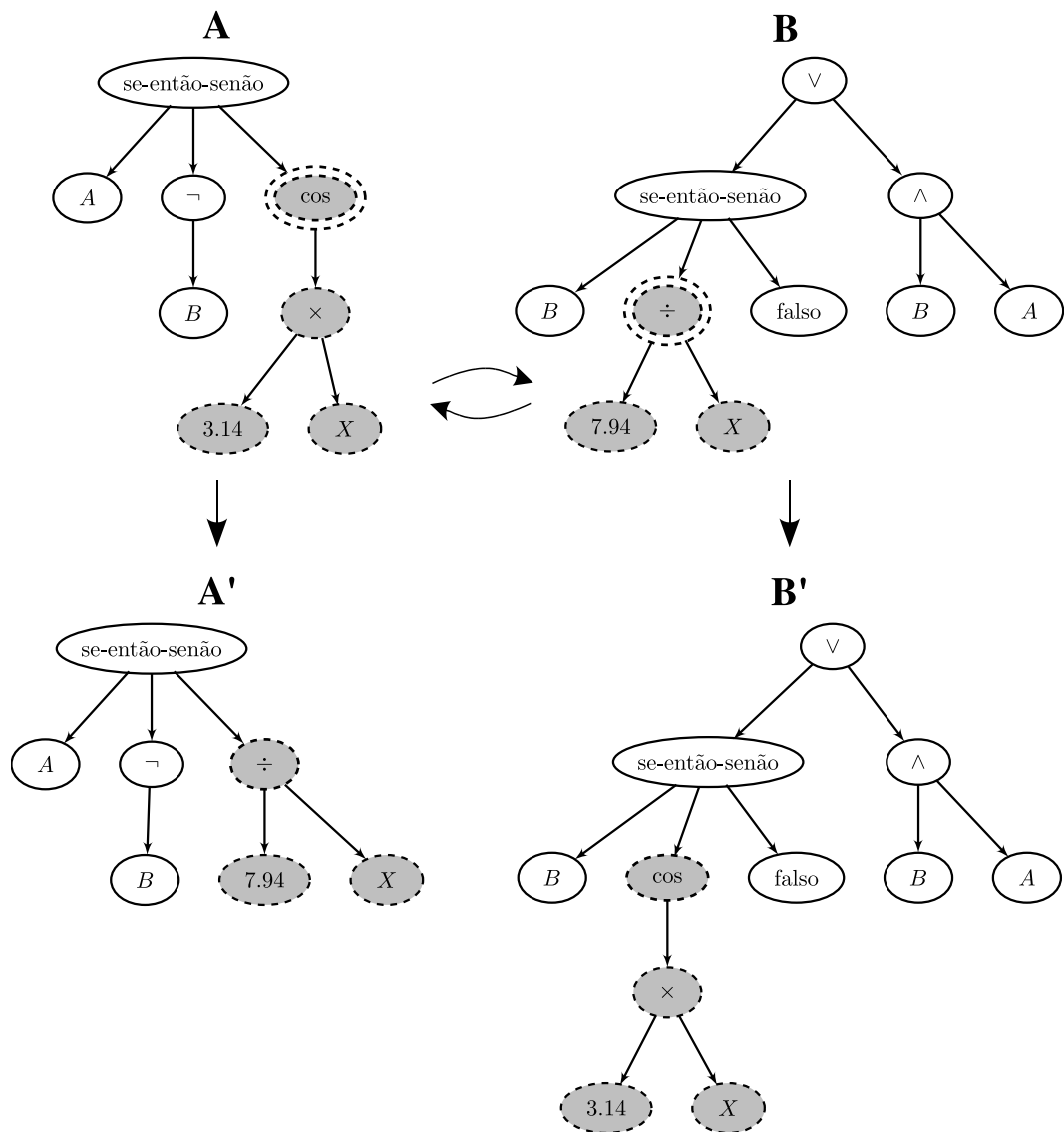


Figura 2.3: Cruzamento padrão

aleatoriamente, é introduzida no local, gerando assim o descendente **A'**.

## 2.9 Métodos de Reprodução

Os métodos de reprodução dizem respeito à política sob a qual os descendentes recém criados são introduzidos no processo evolucionário. Existem dois métodos fundamentais: *geracional* e *steady-state* [21, 22].

O método geracional foi o primeiro a surgir na literatura dos algoritmos de computação evolucionária; funciona basicamente assim: a cada geração os descendentes recém criados são alocados em uma população temporária; quando o número de indi-

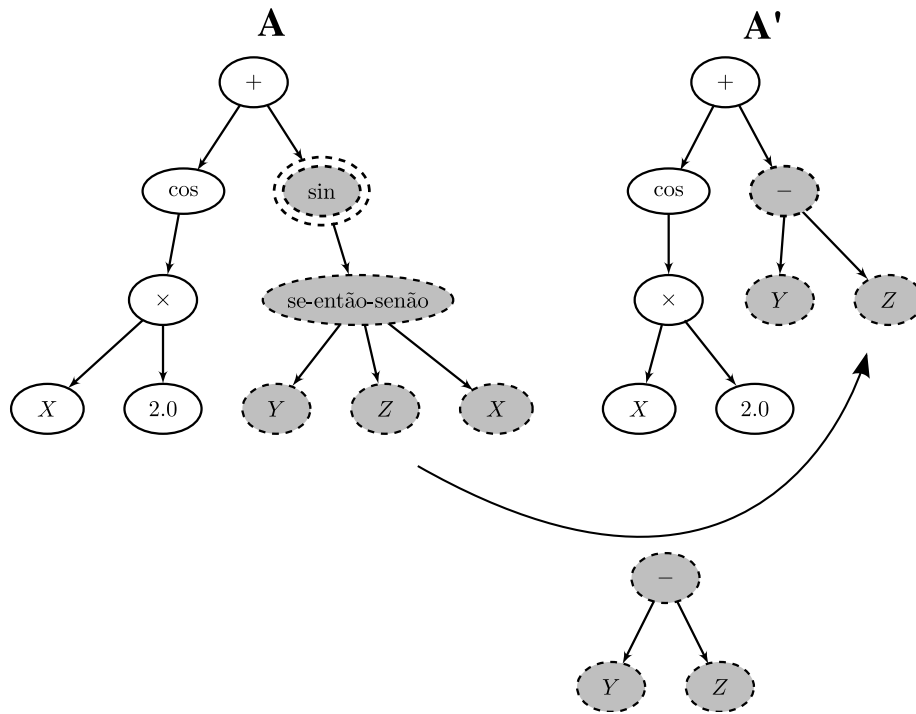


Figura 2.4: Mutação padrão

víduos criados se iguala ao da população principal, a população temporária substitui a principal, e assim todos os indivíduos da geração passada são descartados—uma variação muitas vezes útil, conhecida por *elitismo*, é preservar os  $n$  melhores indivíduos da geração passada.

Uma desvantagem do método geracional, que pode ser decisiva em certas aplicações, é a necessidade de se alocar uma segunda população durante o processo evolutivo. O método de reprodução *steady-state*, por outro lado, insere os descendentes recém criados imediatamente na população atual, logo, eles passam a coexistir com seus antecedentes. Quando um novo descendente é inserido, um indivíduo da população vigente deve ser eliminado para que o tamanho da população permaneça constante. Habitualmente, escolhe-se para ser substituído um indivíduo pouco adaptado (ou o menos adaptado), dessa forma, a evolução descarta soluções pouco promissoras e obtém-se, naturalmente, o efeito de conservação dos melhores indivíduos análogo ao *elitismo* da reprodução geracional.

# Capítulo 3

## O Algoritmo *PGMC*

O algoritmo **PGMC**—sigla de *Programação Genética Multi-populacional e Co-evolucionária*—proposto nesta tese é descrito no decorrer deste capítulo.

O capítulo está organizado da seguinte maneira. Primeiramente, os populares combinadores *bagging* e *boosting* são apresentados e então comparados. Em seguida, discute-se brevemente algumas das principais técnicas de combinação de classificadores no âmbito da programação genética. Por fim, descreve-se como os conceitos do *bagging* e *boosting* podem ser naturalmente integrados à programação genética por meio de mecanismos de competição e cooperação em uma nova abordagem distribuída denominada **PGMC**.

### 3.1 *Bagging e Boosting*

O *bagging* e *boosting* são métodos concebidos para combinar soluções de algoritmos de aprendizagem—em especial classificadores e regressores—na intenção de produzir uma solução final agregada mais robusta e com menor taxa de erro. No âmbito da classificação de dados, o classificador resultante é uma função que prediz uma única *classe* dentre as várias sugestões emitidas pelas soluções parciais. Normalmente esta função é definida como a *votação majoritária*, ponderada ou não por algum quesito (p. ex., “confiabilidade” dos membros).

Sinteticamente, ambos os métodos partilham o princípio geral descrito no Algoritmo 3.1. No algoritmo, o parâmetro  $N$  é o número de iterações, isto é, define

a quantidade de soluções parciais (tamanho do comitê) a ser produzida. A cada iteração um conjunto de treinamento é formado, e então um algoritmo de classificação constrói um classificador com base nesse conjunto de treinamento.<sup>1</sup> Finalmente, após a obtenção de  $N$  soluções parciais, um classificador final é construído como uma função destas.

---

**Algoritmo 3.1** Princípio geral do *bagging* e do *boosting*

---

**para**  $n \leftarrow 1$  até  $N$  **faça**

    Formar um conj. de treinamento  $T_n$  sob alguma distribuição de probabilidade

    Obter o classificador  $C_n$  de um certo algoritmo de classificação usando  $T_n$

**fim para**

    Produzir o classificador final  $C_{final}$  como uma combinação dos  $C_n$ ,  $1 \leq n \leq N$

---

As diferenças operacionais entre *bagging* e *boosting* incidem fundamentalmente em dois pontos. A estratégia de formação dos conjuntos de treinamento ao longo das iterações é um deles. O segundo ponto de dessemelhança é relativo à política de contabilização dos votos de cada solução parcial, ou seja, a função que define o classificador final. Estes, diga-se de passagem, são os pontos chave que governam o conceito da combinação de classificadores, ou seja, a *exploração de sub-regiões* do domínio do problema (separação/diversificação) e a *unificação das soluções avulsas* (agregação).

### 3.1.1 *Bagging*

#### 3.1.1.1 *Bootstrap*: Formação dos Conjuntos de Treinamento

Para cada iteração o *bagging* cria um (novo) conjunto de treinamento baseado no procedimento estatístico de amostragem denominado *bootstrap*<sup>2</sup>[23]. Seu emprego na área de estatística é notavelmente variado e, devido à estreita relação, também o é em mineração de dados [24, 25, 26, 27, 28], porém cabe neste contexto descrever tão somente sua utilidade frente ao *bagging*: produzir conjuntos de treinamento com a propriedade de causar perturbações na direção de busca no algoritmo de classificação, isto é, incitar a busca por regiões suficientemente diversas do domínio do problema

---

<sup>1</sup>O *boosting* e especialmente o *bagging* requerem que o algoritmo de classificação seja *instável*, isto é, pequenas perturbações no conjunto de treinamento devem produzir soluções significativamente diferentes.

<sup>2</sup>O termo *bagging*, aliás, é um acrônimo de *bootstrap aggregating*.

a cada iteração.

A geração de conjuntos de treinamento por *bootstrap* é direta, e funciona como mostrado pelo Algoritmo 3.2. Sendo  $\mathcal{A}$  o conjunto original de treinamento de cardinalidade  $|\mathcal{A}|$ , o novo conjunto  $\mathcal{A}'$  de uma determinada iteração é construído por elementos aleatoriamente selecionados (com reposição) de  $\mathcal{A}$ . Isto significa que, ao final, possivelmente  $\mathcal{A}'$  terá amostras repetidas, ao mesmo tempo em que outras amostras de  $\mathcal{A}$  não serão incluídas.<sup>3</sup>

---

**Algoritmo 3.2** Formação do conjunto  $\mathcal{A}'$  sobre  $\mathcal{A}$  via *bootstrap*

---

**procedimento** BOOTSTRAP( $\mathcal{A}$ )                     $\triangleright$   $\mathcal{A}$  é o conjunto original de amostras  
 $\mathcal{A}' \leftarrow \emptyset$   
**para**  $i \leftarrow 1$  até  $|\mathcal{A}|$  **faça**  
     $x \leftarrow$  elemento (amostra) selecionado aleatoriamente de  $\mathcal{A}$ ,  
    segundo uma distribuição uniforme.  
     $\mathcal{A}' \leftarrow \mathcal{A}' \cup x$   
**fim para**  
**retorne**  $\mathcal{A}'$   
**fim procedimento**

---

### 3.1.1.2 Formação do Classificador Final

A política de agregação do *bagging* é bastante simples, conhecida como *votação majoritária*. A predição da classe de uma instância de dados submetida ao classificador final é decidida em razão do maior número de votos, ou seja, a classe mais votada é a classe predita. Cada membro do comitê tem o mesmo peso de voto. Matematicamente,  $C_{final}$  é expresso como:

$$C_{final}(x) = \arg \max_{y \in Y} \sum_{n=1}^N \llbracket C_n(x) = y \rrbracket \quad (3.1)$$

onde  $x$  representa uma amostra de dados qualquer,  $Y$  é o conjunto das classes do problema,  $N$  o número de iterações (tamanho do comitê);  $\llbracket \pi \rrbracket$  retorna 1 se a condição  $\pi$  é satisfeita ou 0 caso contrário.

---

<sup>3</sup>Mais precisamente, o conjunto  $\mathcal{A}'$  formado terá aproximadamente 63,2% das amostras do conjunto original  $\mathcal{A}$ . Os 36,8% restantes das amostras em  $\mathcal{A}'$  são, portanto, duplicatas [29].

## 3.1.2 *Boosting*

Diferentemente do *bagging*, as iterações do *boosting* são interdependentes, por consequência o seu processamento no todo é obrigatoriamente sequencial. Esta estrutura sequencial deve-se ao fato de cada iteração estar atrelada à sua predecessora, ou melhor, o conjunto de treinamento da iteração corrente baseia-se no comportamento da anterior.

### 3.1.2.1 Formação e Manutenção dos Conjuntos de Treinamento

O conjunto de treinamento  $\mathcal{A}'$  da primeira iteração é idêntico ao conjunto original  $\mathcal{A}$ . Para as próximas iterações, no entanto, os conjuntos de treinamento são formados visando direcionar a busca às amostras mais dificilmente classificáveis segundo as experiências nas iterações passadas. Espera-se com isso que cada solução parcial domine preferencialmente um segmento até então inexplorado, diversificando-se portanto as habilidades de cada classificador. O esquema é implementado usando-se probabilidades de seleção—guiando o treinamento do algoritmo de classificação—atribuídas a cada amostra de dados de  $\mathcal{A}'$ , proporcionais à quão árdua a amostra se apresenta aos classificadores: aumentam-se as chances de seleção das amostras “difíceis” enquanto diminuem-se as das amostras “fáceis”.

Os detalhes de como os pesos são atualizados dependem do tipo de implementação do *boosting*.<sup>4</sup> Todavia, uma versão bastante popular do *boosting* chamada *AdaBoost* [36, 3] procede, exceto para a primeira iteração que é iniciada com probabilidades uniformes, conforme descrito no Algoritmo 3.3. Percebe-se que as amostras que não oferecem resistência ao classificador têm seus pesos reduzidos, isto é, participam menos intensamente do processo de treinamento da iteração seguinte.

### 3.1.2.2 Formação do Classificador Final

Ao contrário do *bagging*, o classificador resultante do *boosting* discrimina os membros da comissão de votação, atribuindo-lhes medidas de “confiança”. O critério

---

<sup>4</sup>Na realidade, *boosting* é um princípio que abriga uma família de implementações (versões) [2, 3, 30, 31, 32, 33, 34, 35], que embora respeitem o perfil do método, variam particularmente em questões quanto ao modo de atualização de pesos e a função de combinação das soluções parciais.

---

**Algoritmo 3.3** *Boosting*: atualização dos pesos da iteração  $n$ 

---

**procedimento** ADABOOST( $\mathcal{P}_n$ ) ▷  $\mathcal{P}_n$  é o conjunto de pesos das amostras de treinamento

$C_n \leftarrow$  Execute o algoritmo de classificação com  $\mathcal{P}_n$  sendo os pesos das amostras

Erro  $\epsilon_n \leftarrow$  somatório dos pesos das amostras *mal classificadas*

**para todo**  $p \in \mathcal{P}_n$  **faça**

$p \leftarrow p \times \begin{cases} \epsilon_n / (1 - \epsilon_n) & \text{se classificou corretamente} \\ 1 & \text{caso contrário} \end{cases}$

**fim para**

Normalize  $\mathcal{P}_n$

**retorne**  $\mathcal{P}_n$  atualizado

**fim procedimento**

---

apoia-se na qualidade do classificador em questão, isto é, quão bem se saiu diante de seu conjunto de treinamento—de acordo com a distribuição de pesos  $\mathcal{P}_n$ . Este raciocínio é bastante intuitivo e justo, à medida que admite discrepâncias nas aptidões dos votantes e então tenta alinhá-las por meio da ponderação.

Novamente, as versões do *boosting* habitualmente discordam quanto às formas exatas de implementação. O *AdaBoost* faz uso da medida de erro  $\epsilon_n$  introduzida no Algoritmo 3.3. Precisamente, cada classificador  $C_n$  do comitê tem peso de voto

$$\ln \frac{1 - \epsilon_n}{\epsilon_n}, \tag{3.2}$$

onde  $\epsilon_n$  é o erro relativo ao classificador em questão e indica seu poder de classificação com base no conjunto de pesos de treinamento no qual foi submetido.<sup>5</sup> Portanto, formalmente tem-se  $C_{final}$  como:

$$C_{final}(x) = \arg \max_{y \in Y} \sum_{n=1}^N \left( \ln \frac{1 - \epsilon_n}{\epsilon_n} \right) \llbracket C_n(x) = y \rrbracket \tag{3.3}$$

### 3.1.3 Comparação entre os Métodos

As aplicações de *bagging* e *boosting*<sup>6</sup> usualmente produzem resultados comparáveis. Contudo, experimentos presentes na literatura [3, 37, 38, 39, 40] sugerem que em média o *boosting* goza de uma razoável margem de vantagem, oferecendo

---

<sup>5</sup>Observa-se que para  $\epsilon > 0.5$  o peso torna-se negativo. Na prática isto não ocorre porque o *AdaBoost* interrompe prematuramente as iterações quando uma solução parcial fracassa com erro  $\epsilon$  acima de  $1/2$ .

<sup>6</sup>Em essência, esta seção comparativa guia-se pela versão *AdaBoost* do modelo *boosting*.

melhores taxas de classificação. Por outro lado, o *boosting* mostra-se mais sensível a ruídos quando comparado ao *bagging*, devido principalmente à sua agressiva política de atualização de pesos para amostras mal comportadas [31, 4, 38, 41, 34]; nesses casos o *bagging* geralmente obtém nítida vantagem.

O *bagging* é um modelo mais simples, tanto conceitualmente como na implementação. Entretanto, a maior complexidade do *boosting* dita, ao mesmo tempo, a flexibilidade e potencial de extensão do método, facultando a construção de versões com diferentes propriedades, possivelmente aprimoradas—em [42] apresenta-se uma visão geral de muitas variações da família *boosting*.

Por fim, diferentemente do *boosting*, o *bagging* é, em razão da sua completa independência entre as iterações, um algoritmo trivialmente paralelizável.

## 3.2 Estado da Arte

A despeito do uso da programação genética como um “mero” algoritmo de classificação por técnicas como *boosting* e *bagging*, muitos trabalhos visaram a efetiva integração/incorporação do conceito de *combinação de classificadores* no âmbito da PG. Esta adequação à estrutura da programação genética provê um ambiente mais natural e versátil, viabilizando a genuína exploração de modelos e dinâmicas evolucionárias.

Hitoshi Iba foi um dos precursores na concepção de forte integração dessa ideia, introduzindo as técnicas denominadas *BagGP* e *BoostGP* [43], que remetem respectivamente ao uso de *bagging* e *boosting* dentro da PG. Seu método basicamente consiste em aplicar as técnicas de *bagging* (no *BagGP*) e *boosting* (no *BoostGP*) sobre sub-populações, escolhendo-se ao final da evolução os melhores indivíduos de cada população como os membros do classificador (ou regressor) final. Para o *BoostGP*, Iba preferiu não implementar de fato o conjunto de pesos para seleção das amostras, e sim usar re-amostragem para tentar simular o efeito. Iba obteve resultados expressivos quando comparado à PG tradicional.

Paris *et al.* [44], motivados pelo *boosting*, desenvolveram o chamado *GPBoost* e o aplicaram em problemas de regressão. A proposta assemelha-se ao *BoostGP*

de Iba [43], no entanto, em vez de simularem a manutenção de pesos através da re-amostragem, optaram por efetivamente implementá-la, por meio da função de aptidão. Esta solução, embora menos direta, agrega precisão ao processo à medida que, assim como no *boosting*, concede a cada amostra, por menor que seja seu peso, chances reais de participar do treinamento em uma determinada iteração. Os resultados encontrados pelos autores em seus experimentos reforçam o vantajoso emprego dos esquemas de combinação de soluções.

Abordagens usando a combinação de classificadores sob o modelo de paralelismo foram propostas por Folino *et al.* no contexto de *Cellular Genetic Programming* (CGP). Em um dos trabalhos [45] os autores incorporam características do *bagging* e desenvolvem o *BagCGPC*<sup>7</sup>; no outro [46] inspiram-se no *boosting* para criar o *BoostCGPC*. Ambos particionam o conjunto original de treinamento e alocam cada parte a uma determinada sub-população. As populações evoluem em paralelo e eventualmente trocam indivíduos vizinhos de modo assíncrono. Confrontando-os com o CGP puro, os autores encontraram resultados claramente satisfatórios, não apenas melhoras sobre as taxas de classificação, mas obtiveram também ganhos nos tempos de execução.

Tanto *BagCGPC* como *BoostCGPC* partilham fundamentos com o *BagGP* e *BoostGP* de Iba, contudo diferem notoriamente quanto à arquitetura paralela e repartição do conjunto de treinamento dentre as sub-populações.

Os ingredientes usados por Folino *et al.* [45, 46] aproximam-se dos propostos para este trabalho: o esquema *bagging/boosting* e o paralelismo (“sub-populações”) sobre a programação genética. Entretanto, os autores privilegiaram a fidelidade na transcrição do *bagging* e *boosting*, o que os deixou de certo modo subordinados à estrutura destes. Por exemplo, muito embora o *BoostCGPC* atue paralelamente ao longo dos subconjuntos de treinamento, a atualização dos pesos dá-se por iterações, isto é, o reajuste de pesos é feito somente após um número pré-determinado de gerações. Isto não é necessariamente ruim, mas a atualização contínua—como a proporcionada pela co-evolução (Seção 3.3.2)—parece ser uma estratégia mais promissora.

Outra diferença estrutural reside no modelo de paralelismo. Enquanto no traba-

---

<sup>7</sup>*Bagging Cellular Genetic Programming Classifier.*

lho de Folino *et al.* [45, 46] adotou-se a topologia espacial particionada em “células” (*Cellular Genetic Programming*) onde indivíduos interagem na sua vizinhança, nesta proposta usa-se o modelo *Ilha*, no qual as populações evoluem semi-isoladamente, sob topologias arbitrárias, com migração casual de material genético em intensidade configurável.

### 3.3 PGMC — *Programação Genética Multipopulacional e Co-evolucionária*

O algoritmo descrito nesta seção, influenciado pelas ideias de *bagging* e *boosting*, objetiva introduzir no paradigma da programação genética as boas características de cada técnica, ao passo que tenta evitar suas principais deficiências.

Dentre as propriedades positivas, pode-se destacar: a arquitetura potencialmente paralela e robustez do *bagging*; a flexibilidade/extensibilidade, o esquema de direcionamento da busca (“pesos das amostras”), e a estimativa de confiabilidade dos votantes encontradas no *boosting*.

Todavia, evitar a introdução concomitante das propriedades ruins, particularmente a sensibilidade aos ruídos, é um desafio. No entanto, a relativa independência entre as populações da PG no modelo proposto, lembrando a estrutura do *bagging*, parece acenar em favor da robustez.

Cabe notar que, assim como a maioria das técnicas da computação evolucionária, o **PGMC** vale-se da flexibilidade e liberdade características desses algoritmos e admite variações dentro do conceito; portanto, em termos estritos, o **PGMC** figura como um *modelo* de algoritmo e não um método rigoroso no qual se define passos precisos e imutáveis.

#### 3.3.1 Introdução

O algoritmo de classificação proposto está fundamentado em três pilares: (i) *programação genética*; (ii) *múltiplas populações* em regime cooperativo (sistema distribuído/paralelismo); e (iii) *co-evolução competitiva* intra-populacional. Adicional-

mente, dois passos o completam: (iv) a *amostragem do conjunto de treinamento*<sup>8</sup> e (v) a *combinação dos classificadores*.

O desenvolvimento do **PGMC** foi motivado com base nas seguintes ponderações:

- A abordagem de combinação de classificadores é absolutamente vantajosa no que concerne à redução das taxas de erro de predição.
- O *bagging* é uma técnica trivialmente paralelizável e apresenta boa robustez frente a ruídos, mas o ganho com a precisão de classificação é modesto.
- O *boosting* normalmente obtém bons índices de classificação, mas é um algoritmo intrinsecamente sequencial e eventualmente sofre com ruídos.
- A programação genética é um rico, poderoso e versátil paradigma da computação evolucionária; a transcrição natural de conceitos para este paradigma beneficia-se diretamente de todos seus atributos, dentre eles os mais diversos modelos evolutivos, arquitetura inerentemente paralela/distribuída, flexibilidade e robustez.

Pode-se visualizar na Figura 3.1 o esquema estrutural do algoritmo proposto nesta tese. No gráfico está esquematizado um grupo de populações<sup>9</sup>  $\{P_1, P_2, \dots, P_N\}$  posicionadas ao longo de uma topologia distribuída tipo Ilha (Seção 3.3.3). Cada população, internamente, co-evolui competitivamente nos moldes descritos na Seção 3.3.2 com o seu próprio conjunto de treinamento  $\{A_1^{P_n}, A_2^{P_n}, \dots, A_J^{P_n}\}$ , formados pelo método de amostragem como definido na Seção 3.3.4. Paralelamente, no nível inter-populacional, as populações cooperam entre si mediante o compartilhamento de soluções promissoras descobertas localmente. Ao término do processo evolucionário os melhores indivíduos classificadores de cada população  $\{C_*^{P_1}, C_*^{P_2}, \dots, C_*^{P_N}\}$  são eleitos como membros do comitê de votação, compondo-se assim o classificador final  $C_{final}$  como uma função (Seção 3.3.5) destes membros.

---

<sup>8</sup>Ou “população de amostras”, segundo a terminologia do sistema co-evolucionário. Vide Seção 3.3.2.

<sup>9</sup>Estritamente falando, cada população—ou *ilha*, tido neste trabalho como sinônimos quando em caráter geral—na realidade é composta de duas sub-populações: a de *classificadores* e de *amostras de dados*.

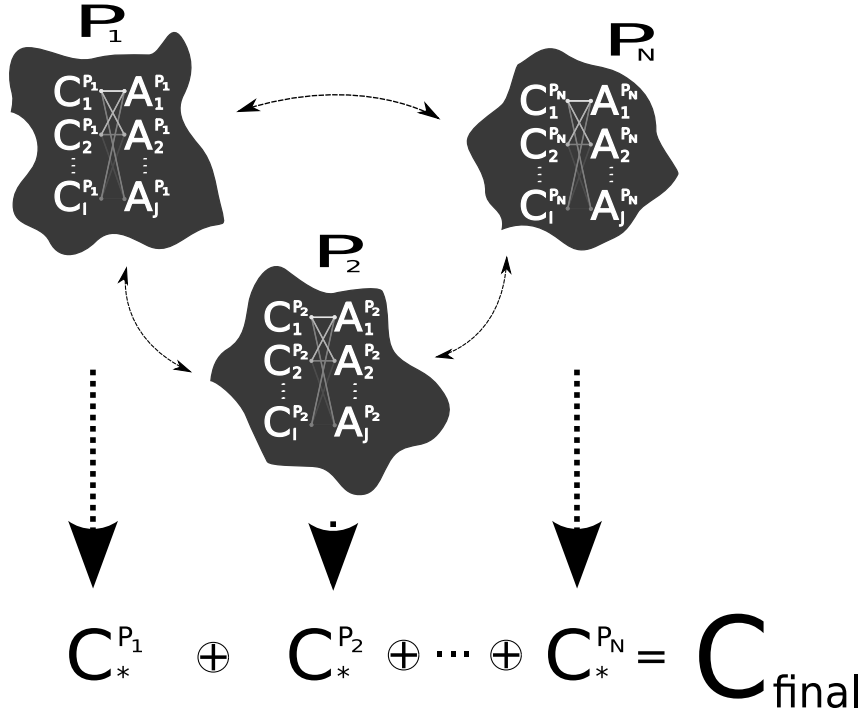


Figura 3.1: Internamente às populações há co-evolução entre os indivíduos classificadores e as amostras de dados; no nível externo as populações cooperam via troca de informações.  $C_i^{P_n}$  é o  $i$ -ésimo classificador da população  $P_n$  (que contém  $I$  indivíduos), enquanto que  $A_j^{P_n}$  é a  $j$ -ésima amostra de dados, de um total de  $J$  amostras, da população  $P_n$ .  $C_*^{P_n}$  é o melhor indivíduo da população  $n$  obtido após o processo evolutivo.  $C_{\text{final}}$  é o classificador produto da agregação dos  $C_*^{P_n}$ .

De forma equivalente, porém mais bem estruturada, pode-se expressar a proposta deste trabalho através do Algoritmo 3.4. Dado como entrada o conjunto original de treinamento  $T$ , o algoritmo **PGMC** retorna um classificador final  $C_{\text{final}}$ . Os parâmetros  $N$  e  $G$ , que configuram-se como parâmetros que variam conforme o problema, estipulam respectivamente a quantidade de populações (“decomposições”) e o número máximo de gerações (“épocas”) executadas pelo algoritmo. A função  $\text{CO-EVOLUA}(P_n)$  corresponde a uma chamada do Algoritmo 3.5 co-evolucionário sobre a população  $P_n$ , composta de indivíduos classificadores e amostras de dados.

Percebe-se que, diferentemente do *boosting*, as populações (“iterações”) do **PGMC** são, exceto no que tange as migrações, *independentes* entre si; em outras palavras, o direcionamento de busca (“manutenção de peso”)—cuja inspiração se deve ao *boosting*—realizado pelo procedimento  $\text{CO-EVOLUA}(P_n)$  é restrito e local à população  $P_n$ . No *boosting*, entretanto, a distribuição de pesos da iteração corrente só pode ser determinada conhecendo-se os valores das iterações passadas. Essa

---

**Algoritmo 3.4 PGMC** – PG Multi-populacional e Co-evolucionária

---

**procedimento** PGMC( $T$ )  $\triangleright T$  é o conjunto original de treinamento  
**para** população  $n \leftarrow 1$  **até**  $N$  (**em paralelo**) **faça**  
    Forme sobre  $T$  o conj. de amostras de treinamento  $\{A_1^{P_n} \dots A_{J^n}^{P_n}\}$  para  $P_n$   
    Crie  $I$  indivíduos classificadores  $\{C_1^{P_n} \dots C_I^{P_n}\}$  para  $P_n$   
    Avalie os indivíduos  $\{C_1^{P_n} \dots C_I^{P_n}\}$  com amostras escolhidas aleatoriamente  
**fim para**  
**para** população  $n \leftarrow 1$  **até**  $N$  (**em paralelo**) **faça**  
    **para** geração  $g \leftarrow 1$  **até**  $G$  **faça**  
        **enquanto** número de classificadores gerados  $< I$  **faça**  
            CO-EVOLUA( $P_n$ )  
        **fim enquanto**  
        **se** período de migração **então**  
            Troque material genético com outra(s) população(ões)  
            segundo uma política preestabelecida  
        **fim se**  
    **fim para**  
    **fim para**  
     $C_{final} \leftarrow C_*^{P_1} \oplus C_*^{P_2} \oplus \dots \oplus C_*^{P_N}$   
    **retorne**  $C_{final}$   
**fim procedimento**

---

independência entre as populações do **PGMC** assemelha-se à estrutura do *bagging*, e é condição necessária para a implementação do paralelismo.

Pode-se argumentar, por outro lado, que a manutenção global de pesos ao estilo *boosting* soe ser mais efetiva. Contudo, isto não é necessariamente verídico e, não obstante, a atualização sequencial e continuada de pesos é uma política mais agressiva e favorece, desse modo, o risco de perda de generalização (*overfitting*).

### 3.3.1.1 Criação dos Indivíduos Classificadores

Os indivíduos classificadores são programas de computador representados por uma estrutura qualquer que, fornecido um conjunto de atributos de uma amostra de dados como argumento, devem retornar um valor discreto que denota a classe predita da amostra em questão. Um indivíduo é, portanto, uma estrutura funcional capaz de realizar sobre os argumentos fornecidos (valores dos atributos) operações lógicas, relacionais, condicionais/de desvio e, às vezes, operações aritméticas.

O **PGMC**, em sintonia com os algoritmos de programação genética em geral, não especifica ou restringe a linguagem/estrutura do programa classificador. No entanto,

a implementação do algoritmo **PGMC** deste trabalho descreve os classificadores pela estrutura hierárquica árvore, servindo-se ainda de uma representação formal por gramática—a Seção 4.2.1 discursa em pormenores a respeito da representação adotada e como os indivíduos são criados.

### 3.3.2 Co-evolução Competitiva Amostra-Classificador

O principal mecanismo que norteia os algoritmos estilo *boosting* reside no contínuo direcionamento do esforço de busca às amostras de dados mais dificilmente classificáveis. A co-evolução competitiva como descrita nesta seção é capaz de, indireta e naturalmente, oferecer efeito semelhante à medida que, de forma análoga, conduz o processo evolucionário às amostras mais difíceis.

Paredis, no trabalho intitulado *Steps Towards Co-Evolutionary Classification Neural Networks* [47]<sup>10</sup>, desenvolve um algoritmo genético co-evolucionário e aplica-o no problema de ajuste de pesos de redes neurais no domínio de classificação de dados. Neste sistema, uma população de redes neurais classificadoras co-evolui competitivamente com a população constituída de padrões de entrada (“amostras de treinamento”).

A dinâmica co-evolucionária segundo Paredis ocorre da seguinte forma. Indivíduos de ambas as populações são selecionados proporcionalmente à sua aptidão—quando maior a aptidão maior a chance—para se confrontarem. Os confrontos dão-se emparelhando dois indivíduos por vez, um representante das redes classificadoras e outro dos dados de entrada. Se uma rede neural consegue classificar corretamente um padrão de entrada, então sua aptidão é *acrescida* enquanto que a aptidão do padrão de entrada é *diminuída*. A situação inversa é análoga, isto é, se o padrão de entrada induz a rede neural a classificá-lo incorretamente, aumenta-se sua aptidão e diminui-se a da rede neural.

Nota-se nesse esquema que, ao passo que as redes neurais classificadoras experimentam inteiramente o processo evolucionário, isto é, sofrem modificações no curso da evolução, a população de amostras de dados, obviamente, não se submete às

---

<sup>10</sup>Paredis também desenvolve o tema em [48, 49, 50, 51].

mudanças, pois do contrário descaracterizaria-se o conjunto de treinamento.

Apesar de não haver alteração genética na população de amostras, esta provê uma pressão contrária (co-evolutiva) à população de redes neurais, à medida que os pontos mais dificilmente classificáveis tendem a ser mais ofertados para disputarem com as redes. A tensão surge, assim, quando as melhores redes neurais são mais frequentemente escolhidas para competir contra, justamente, as mais difíceis amostras. Basicamente, quanto mais um indivíduo se destaca como mais bem adaptado, mais ele terá que provar o seu valor, do contrário é derrubado.

A competição redes neurais *versus* padrões de entrada pode ser elucidada graficamente, como disposto na Figura 3.2. Neste esquema, os círculos à esquerda representam a população de redes classificadoras, enquanto que os da direita as de amostras de dados. Ambas estão ordenadas, com os indivíduos mais adaptados no topo. As ligações representam os encontros entre os indivíduos da população de classificadores e os membros da população de pontos de entrada.<sup>11</sup>

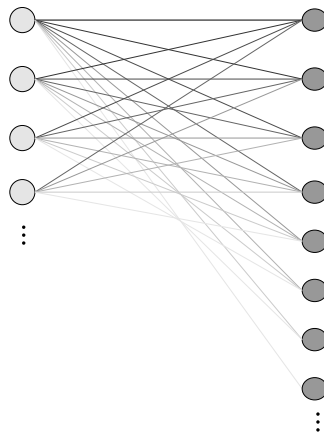


Figura 3.2: Esquema gráfico da competição entre classificadores e amostras.

Sutilmente, nota-se que os indivíduos mais ao topo tendem a passar por maior provação (mais encontros), enquanto os menos adaptados são pouco convocados, devido à condição “comprovada” de baixa qualidade. Este processo é interessante e demonstra um ponto importante: a co-evolução direciona a busca para os testes mais difíceis, os exemplos não resolvidos, poupando-se assim recursos computacionais que certamente seriam desperdiçados na tentativa repetida de resolver casos fáceis; mas não apenas isso, o processo co-evolucionário, em contrapartida, também evita

<sup>11</sup>As linhas mais escuras denotam as posições do *ranking* com maior frequência de competição.

dispêndios com indivíduos classificadores pouco promissores, à medida que estes indivíduos são mais raramente avaliados e pouco participam de operações genéticas.

### 3.3.2.1 Aptidão Cumulativa

Em vez da tradicional aptidão atribuída ao indivíduo após sua avaliação, Paredis desenvolveu um sistema cumulativo<sup>12</sup>, originado no conceito de “energia”, no qual o grau de adaptação de um indivíduo—e daí o sucesso de reprodução—é medido pela quantidade de energia acumulada durante sua existência.<sup>13</sup>

No âmbito da programação genética, a aptidão cumulativa funciona da seguinte maneira. Um número finito é determinado, e este estabelece a dimensão da “memória”, isto é, quantos confrontos serão lembrados para cada indivíduo. Na ocorrência de uma nova disputa o registro mais antigo cede lugar ao mais recente. Neste histórico guarda-se a cada posição uma informação do tipo “venceu” ou “não venceu” seu adversário. A proporção entre vitórias e derrotas define, assim, a aptidão de um indivíduo.

Uma importante propriedade da aptidão cumulativa é sua melhor robustez no que concerne ao gerenciamento de ruídos (“*outliers*”). No modelo co-evolucionário, um dado ruidoso poderia facilmente degradar o processo evolutivo—e assim a solução final—, pois estar além do escopo do conhecimento da base de dados implica em ser inerentemente hostil aos classificadores. A degradação ocorreria à medida que a co-evolução desviaria esforços em direção à(s) amostra(s) ruidosa(s), na tentativa de classificá-la(s) a todo custo, afinal, esta é a essência da co-evolução competitiva. No entanto, duas características da aptidão cumulativa amenizam os danos causados pelas amostras ruidosas: (i) ao manter-se uma memória finita e relativamente curta de avaliações, automaticamente restringe-se a distância entre indivíduos árdus e triviais, abrandando-se conseqüentemente a pressão de seleção a favor dos ruídos; e (ii) a memória volátil/temporária assegura que avaliações ruidosas sejam, em algum momento, descartadas, e portanto deixem de impactar tão incisivamente.

---

<sup>12</sup>Originalmente, *Life-Time Fitness Evaluation*

<sup>13</sup>Para maior informação sobre o conceito de energia e o contexto de sua aplicação, ver [52].

### 3.3.2.2 O Algoritmo Co-evolucionário

O cerne do processo co-evolucionário de Paredis adaptado ao contexto da programação genética na tarefa de classificação de dados é mostrado no Algoritmo 3.5, onde se observa uma única iteração—uma execução do procedimento  $\text{CO-EVOLUA}(P)$ . Assume-se que ambas as populações estão iniciadas e com os indivíduos avaliados—na primeira avaliação, cada membro da população de classificadores compete contra um certo número de exemplos *aleatórios* do conjunto de treinamento, automaticamente atualizando-se também as aptidões das amostras de dados. As funções *acumule vitória em* e *acumule derrota em* inserem o registro de vitória (um) e derrota (zero), respectivamente, no histórico de encontros dos membros em competição. Obviamente, o valor mais antigo é automaticamente descartado, como em uma fila—o primeiro a entrar é o primeiro a sair (*FIFO*). Assim que os novos descendentes são gerados através da aplicação de operadores genéticos, por não possuírem qualquer aferição prévia de qualidade, eles são avaliados confrontando-os com amostras escolhidas ao acaso, de forma análoga como é feito na avaliação dos indivíduos das populações recém criadas. O método de reprodução adotado é o *steady-state* [21, 22], no qual os novos indivíduos são imediatamente inseridos na população corrente, e portanto coexistem com seus antepassados.

---

**Algoritmo 3.5** Processo co-evolucionário com aptidão cumulativa

---

**procedimento**  $\text{CO-EVOLUA}(P)$   $\triangleright P$  é uma população/ilha  
**enquanto** *ciclo* estiver incompleto **faça**  
    Competidor  $C^P \leftarrow$  selecione em  $P$  um classificador *bem adaptado*  
    Competidor  $A^P \leftarrow$  selecione em  $P$  uma amostra *bem adaptada*  
    *Promova competição* entre  $C^P$  e  $A^P$ :  
    **se**  $C^P$  classifica  $A^P$  **então**  
        acumule vitória em  $C^P$   
        acumule derrota em  $A^P$   
    **senão**  
        acumule derrota em  $C^P$   
        acumule vitória em  $A^P$   
    **fim se**  
**fim enquanto**  
 $C_{X_1}^P \leftarrow$  selecione em  $P$  um classificador *bem adaptado*  
 $C_{X_2}^P \leftarrow$  selecione em  $P$  outro classificador *bem adaptado*  
Aplique *operadores genéticos* em  $C_{X_1}^P$  e  $C_{X_2}^P$  e gere os descendentes  $C_{Y_1}^P$  e  $C_{Y_2}^P$   
Avalie  $C_{Y_1}^P$  e  $C_{Y_2}^P$  e insira-os em  $P$ , substituindo classificadores *pouco* adaptados  
**fim procedimento**

---

As passagens do Algoritmo 3.5 que carecem de esclarecimentos ou merecem ser tratadas com maior profundidade são abordadas a seguir.

## Parâmetros

O número de competições por cada iteração do procedimento  $\text{CO-EVOLUA}(P)$  é determinado pelo parâmetro *tamanho do ciclo*—aqui representado pela letra grega  $\gamma$ . Indiretamente relacionado a este é o parâmetro que define a memória de confrontos, de ambos classificadores e amostras de dados, denominado *tamanho do histórico*—simbolizado por  $\eta$ .

O comportamento de  $\gamma$  e  $\eta$  não é bem conhecido, na realidade, a determinação de seus valores é uma tarefa essencialmente intuitiva, arbitrária [51].<sup>14</sup> Estes dois parâmetros são correlacionados, e decerto relacionam-se também com outras variáveis; por exemplo, o tamanho do ciclo, além da estreita relação com tamanho do histórico, é ainda influenciado pelo número de indivíduos classificadores, quantidade de amostras de dados e pressão de seleção de competição. A despeito desta intrincada trama, que aliás é habitual à maioria dos parâmetros dos algoritmos evolucionários [53], algumas delineações no que tange a estipulação dos valores de  $\gamma$  e  $\eta$  podem ser esboçadas.

Analisado sob uma outra perspectiva, nota-se que a especificação do tamanho do ciclo dita a razão entre a intensidade de competições (“avaliações”) e a geração de descendentes. Valores baixos para  $\gamma$  aceleram o ritmo de geração de descendentes, mas por outro lado reduzem o rigor com que a população é avaliada e, conseqüentemente, estimulam a introdução de novos classificadores de qualidade questionável—a aptidão de seus pais não pôde ser apropriadamente avaliada. Há ainda uma séria implicação prática quando se atribuem valores decididamente baixos para  $\gamma$ : operações genéticas em estruturas mais complexas, como a árvore, são computacionalmente caras—certamente mais dispendiosas do que uma *competição/classificação*—, e quanto menor o  $\gamma$  maior é o número de operações genéticas dentro de um certo limite de avaliações. No outro extremo, valores demasiadamente altos para o tamanho do ciclo causam desperdício de avaliações, que podem inclu-

---

<sup>14</sup>Paredis emprega em seus trabalhos correlatos o valor constante 20, tanto para o tamanho do ciclo quanto para o do histórico.

sive anular o benefício da alocação “inteligente” de esforço computacional oferecido pela dinâmica co-evolucionária. Isto decorre do fato de que o histórico, sendo finito, lembra-se tão somente dos resultados das competições mais recentes.

No que concerne ao parâmetro que define o tamanho do histórico de confrontos,  $\eta$ , a atribuição de valores excessivamente baixos acarreta dois problemas: (i) perde-se a capacidade discriminatória da qualidade dos indivíduos, ou seja, diminui-se a graduação/suavidade das aptidões—o que é particularmente crítico quando há um grande número de indivíduos ou amostras de dados; e (ii) favorece-se a ocorrência da contínua alternância improdutiva de “estratégias” dentro da população, em um fenômeno conhecido como *problema do ciclo* [54, 55]—a dinâmica evolutiva torna-se instável e estagna. No caso oposto, onde se define um histórico cujo tamanho seja demasiadamente grande, expõe-se o processo evolucionário a um excesso de estabilidade, comprometendo-o em função do retardo (“assincronia”) nas respostas co-evolutivas. Por exemplo, pode-se imaginar o cenário em que uma amostra de dados tenha um histórico de baixa aptidão devido à facilidade com que vem sendo classificada até então; se por algum motivo os classificadores evoluem um certo padrão onde perdem a habilidade de classificação desta suposta amostra de dados, haverá uma latência—proporcional ao tamanho do histórico—até que esta amostra de dados acumule vitórias suficiente para que possa promover sua aptidão a um nível crítico e contra-atacar o pretensioso padrão que acabara de emergir.

Por fim, um terceiro parâmetro relacionado à iteração co-evolucionária ( $\beta$ ), discutido adiante, é responsável pela dosagem do foco de treinamento às amostras de dados difíceis.

### Cálculo da aptidão

A aptidão normalizada de um indivíduo  $x$ , seja ele um classificador ou amostra de dados, é dada pela proporção de vitórias obtidas nas  $\eta$  competições mais recentes:

$$f(x) = \frac{\sum_{i=1}^{\eta} H_i}{\eta}, \quad (3.4)$$

onde  $H_i$  é o  $i$ -ésimo valor do histórico de confrontos, que pode ser: 1 se venceu aquela competição<sup>15</sup> ou 0 caso contrário.

Precisamente, a aptidão real de um indivíduo *classificador*, visando o controle de complexidade, é uma ponderação entre o valor provido pela Equação 3.4 e o tamanho de sua estrutura (número de nós); uma descrição detalhada desse método encontra-se na Seção 4.2.8.

## Seleção dos indivíduos

Dois tipos de esquemas de seleção estão envolvidos no Algoritmo 3.5: a função de seleção dos *indivíduos classificadores* e a das *amostras de dados*. Em termos estritos, ambos os tipos de seleção devem apenas, em média, favorecer as soluções mais adaptadas. Na prática, no entanto, alguns esquemas são mais apropriados e/ou convenientes do que outros no que concerne à dinâmica da pressão de seleção, complexidade computacional e facilidade de implementação [19].

A seleção por *torneio*, no qual alguns indivíduos são sorteados da população e aquele dentre estes com melhor aptidão é o indivíduo selecionado/vencedor, é um esquema prático, eficiente e popular. Dessa forma, este é o tipo de seleção adotado para a população principal de indivíduos classificadores.

Quanto à população de amostras, Paredis [47] sugere o esquema de seleção do tipo *ranking*. Neste, o conjunto de treinamento é ordenado de maneira que as amostras de dados mais árduas (melhor aptidão) situem-se nas primeiras posições. Baseado neste arranjo, uma função de seleção estocástica retorna índices do *ranking* cuja distribuição tende a favor da seleção dos primeiros elementos. A função de seleção por *ranking*, contudo, pode ser linear ou não-linear; Paredis adota em seus trabalhos [47, 51] a função linear descrita pela Equação 3.5 [22] com o parâmetro de pressão de seleção  $\beta = 1,5$ , o que significa que o melhor membro (índice 1) tem 1,5 vezes mais chance de ser selecionado do que o membro situado no meio do *ranking* (índice  $\lceil J/2 \rceil$ ). Na equação,  $j$  é o índice retornado da amostra de dados selecionada ( $1 \leq j \leq J$ ),  $J$  é o número de amostras de treinamento, e  $\text{RANDOM}()$  é uma função

---

<sup>15</sup>Se o indivíduo em questão é um classificador, então vencer a competição significa ter tido êxito na classificação—predição correta; se é uma amostra de dados, ter vencido significa ter resistido à classificação—predição errada.

que devolve um valor aleatório real, uniformemente distribuído, no intervalo  $[0, 1)$ .

$$j = \left\lceil J \times \frac{\beta - \sqrt{\beta^2 + \text{RANDOM}() \times 4(1 - \beta)}}{2(\beta - 1)} \right\rceil + 1, \quad 1 < \beta \leq 2 \quad (3.5)$$

Avaliações empíricas não oficiais, no entanto, demonstraram que uma função *ranking* de feitiço exponencial seria mais ajustada ao **PGMC**; colocado de outro modo, uma política mais agressiva de direcionamento às amostras mais dificilmente classificáveis mostrou-se mais vantajosa.<sup>16</sup> Devido a esta constatação, um esquema de seleção não-linear foi proposto, como apresentado pela Equação 3.6. De forma análoga à Equação 3.5, o parâmetro  $\beta$  estipula a pressão de seleção. Exige-se, no entanto, cautela ao ajustar o valor de  $\beta$ ; valores muito baixos não focam efetivamente a busca às amostras difíceis, ao passo que valores abusivamente altos estimulam o *overfitting*.

$$j = \left\lceil [\text{RANDOM}()]^\beta \times J \right\rceil + 1, \quad \beta > 1 \quad (3.6)$$

### Operadores genéticos

Entende-se por *operadores genéticos*, como consta no Algoritmo 3.5, qualquer procedimento que recombine e/ou modifique a estrutura (árvore) de um ou mais indivíduos classificadores, gerando portanto descendentes. O **PGMC** não impõe operadores genéticos específicos, a despeito disso, os mais populares são as versões convencionais de *cruzamento* e *mutação*.

### 3.3.3 Modelo de Paralelismo por Ilhas

No modelo de paralelismo por *ilhas* as populações são espalhadas logicamente segundo uma determinada topologia de conexão, formando uma espécie de arquipélago virtual. Cada população evolui paralelamente sob uma dinâmica que imita as restrições características da separação física de ilhas naturais. Eventualmente, as populações (“ilhas”) comunicam-se umas com outras e trocam material genético

---

<sup>16</sup>A função não-linear, não obstante, está em melhor sintonia com o esquema de atualização não-linear de pesos do *boosting* (Algoritmo 3.3).

(via migrações), seja indivíduos ou outras entidades definidas para uma população.<sup>17</sup> Este esquema é excepcionalmente versátil, admite uma infinidade de configurações e ajuste fino nos seus atributos.

Ao mesmo tempo simples, em termos conceituais e de implementação, a topologia de paralelismo por ilhas adequa-se precisamente aos requisitos da presente proposta, legitimando portanto sua adoção neste trabalho. O objetivo na inserção desse modelo é obter algo além do puro paralelismo, isto é, agregar os benefícios de um sistema distribuído. Dessa maneira, atinge-se a co-evolução por cooperação no nível inter-populacional—compartilhando o “conhecimento”—em contraponto com a competição intra-populacional entre os classificadores e amostras de dados.

Uma propriedade fundamental do modelo ilha no contexto da combinação de classificadores é o caráter *semi-isolado* de evolução. A importância deste atributo decorre da nítida vantagem em se construir comitês cujos membros sejam diversificados [57, 58, 59], ou seja, que apresentem comportamentos distintos.<sup>18</sup> No **PGMC**, a diversidade inicial é provida pelo método de amostragem e estocasticidade própria de cada população (diferentes “sementes” aleatórias). Todavia, ao se permitir a troca de material genético entre as populações, deve-se tomar cautela para que a diversidade seja preservada ao longo de todo o processo evolutivo; idealmente, a intensidade de comunicação entre as ilhas deve ser suficiente para estimular/acelerar o processo evolutivo sem no entanto prejudicar a diversidade, isto é, sem comprometer a individualidade de cada ilha.

A definição precisa do modelo por ilhas implica na determinação de algumas variáveis, que são listadas a seguir. Não cabe ao **PGMC** a estipulação de uma configuração particular desses parâmetros, mesmo porque decerto não existe um único arranjo que seja ótimo—ou mesmo apropriado—para todos os problemas de classificação de dados.

- *Conectividade da topologia* – quais ilhas estão interconectadas, isto é, quem se

---

<sup>17</sup>Em [56] descreve-se de forma abrangente o modelo ilha (e os mais variados esquemas paralelos e distribuídos) e realiza-se experimentos que demonstram a clara vantagem sobre o modo convencional, mesmo quando a população original é particionada entre as ilhas.

<sup>18</sup>Em [58] são apresentadas várias *medidas de diversidade* em comitês de classificadores; boa parte dessas medidas, no entanto, baseiam-se simplesmente em quanto os classificadores discordam nas predições das amostras fornecidas.

comunica com quem e em qual direção de fluxo. Sendo  $N$  o número de ilhas em um sistema distribuído, existem exatamente  $4^{(N^2-N)/2}$  conexões topológicas possíveis<sup>19</sup>, portanto a quantidade de configurações admissíveis cresce exponencialmente em relação ao número de ilhas. Apesar desta constatação, na prática apenas um subconjunto predefinido de padrões topológicos é usado. Os padrões incluem a conectividade na forma de fila, anel, estrela, entre outras—na Seção 4.2.4.1 são descritas as topologias empregadas na implementação do **PGMC** deste trabalho.

- *Frequência de comunicação* – intervalo em que as migrações realizam-se. Tipicamente, o processo de comunicação ocorre a cada geração, embora nada impeça que seja realizado em intervalos menores, como por exemplo a cada *ciclo co-evolucionário* ou até mesmo a cada *competição*—quanto maior a frequência, no entanto, maior é a carga sobre as instalações físicas de comunicação (p. ex., a rede de computadores). As comunicações podem ser *síncronas* ou *assíncronas*. No modo síncrono as populações interrompem o processamento e aguardam pela chegada do material genético e, dependendo da implementação, a ilha que iniciou a comunicação aguarda até que a destinatária receba o material. Por sua vez, o modo assíncrono não espera até que o material genético chegue ou seja recebido, sendo portanto nitidamente mais eficiente. Não obstante, a plausibilidade biológica da comunicação assíncrona é superior: na natureza os eventos migratórios dão-se normalmente de modo paralelo e sem sincronia.
- *Taxas migratórias* – define a quantidade de material trocado por migração. No contexto de combinação de classificadores esta variável deve ser definida com prudência, pois taxas altas de transferência de material genético comprometem a diversidade do comitê e, por conseguinte, a qualidade do classificador final. Esta condição é sensivelmente menos problemática quando as ilhas são treinadas com conjuntos *disjuntos* de amostras de dados, porque neste caso há menor chance de uma solução sub-ótima transferida à uma segunda ilha dominar a população desta, cuja função objetivo é tecnicamente distinta—ainda

---

<sup>19</sup>Tal fórmula considera as possibilidades de *não conexão*, *conexão direcionada* (nos dois sentidos) e *conexão não direcionada* a cada par de ilhas; isto explica a base 4.

que ambas as funções objetivo aproximem um mesmo conceito, que é o da base de dados em questão.

- *Tipo de material genético* – as entidades que serão permutadas; usualmente indivíduos, mas não restrito a estes. Grande parcela das aplicações que utilizam o modelo ilha limita-se a transferir soluções candidatas, no entanto, não é difícil imaginar a utilidade do compartilhamento de outras informações. Por exemplo, um sistema distribuído poderia explicitamente induzir a formação de nichos, valendo-se de informações contínuas sobre as regiões do espaço de busca que cada ilha está explorando. O **PGMC**, como descrito neste trabalho, também se limita à permutação de indivíduos classificadores, mas a investigação da troca de outras informações é atraente e parece ser promissora (Seção 6.1).
- *Política de seleção* – regras de escolha dos membros que serão transferidos e substituídos. A transferência de material genético deve agregar benefícios à população destinatária, do contrário perde-se a razão de fazê-la. Por agregação de benefícios entende-se, a longo prazo, idealmente como a obtenção de um classificador com exímia capacidade de generalização e características próprias (distinto). Tais benefícios podem advir tanto do compartilhamento de blocos de construção promissores<sup>20</sup> quanto da (re)introdução de diversidade. Para tanto, a seleção dos emigrantes deve favorecer os indivíduos mais adaptados, mas não necessariamente restringir-se ao *melhor* indivíduo da população, que aliás pode causar efeito colateral de degradação da diversidade global. No processo de imigração, por outro lado, um indivíduo nativo precisa ser eliminado da população local a fim de ceder espaço para que o imigrante seja acolhido—o tamanho das populações deve permanecer constante durante a evolução. O raciocínio agora é inverso: seleciona-se um indivíduo pouco adaptado, portanto irrelevante à população, para ser substituído pelo recém chegado integrante.

---

<sup>20</sup>Blocos de construção são, basicamente, fragmentos de material genético que, muito embora não tenham valor isoladamente, quando devidamente combinados produzem soluções bem adaptadas.

### 3.3.4 Método de Amostragem

Elementar aos métodos de combinação de classificadores é a formação de um comitê cujos membros sejam diversos [57, 58, 59]. Tipicamente, a diversidade é promovida valendo-se de: (i) amostragem particular para cada população; (ii) perturbação nos parâmetros que guiam a evolução; e/ou (iii) adoção de diferentes algoritmos para o treinamento de cada membro do comitê. Com exceção do terceiro método, que por motivos óbvios não se aplica ao modelo proposto, tanto a amostragem diferenciada quanto a perturbação dos parâmetros podem—e devem—ser empregados no **PGMC** como instrumentos de promoção da diversidade do comitê.

A fim de evitar a necessidade do ajuste de um conjunto específico de parâmetros para cada população, pode-se alternativamente atingir a diversidade neste nível usando-se *sementes distintas* para os geradores de números aleatórios das populações; em outras palavras, basta tornar independentes os processos evolutivos.

Sob a dinâmica co-evolucionária, entretanto, as sementes distintas têm efeito diminuído, devido à política de direcionamento da busca às regiões menos triviais do espaço. Esta política faz com que, para conjuntos de treinamento idênticos, os processos evolutivos comportem-se de maneira bastante similar, focando a busca em um *mesmo* subconjunto de amostras de dados, e portanto comprometendo a singularidade de cada classificador evoluído. Portanto, como forma de garantir as condições *necessárias*<sup>21</sup> para a ocorrência e conservação da diversidade, faz-se necessária a adoção de um segundo método, a amostragem por população do conjunto original de treinamento.

O objetivo da amostragem é prover uma função objetivo, própria a cada população, que foque em uma região específica do espaço das amostras de dados. A ideia é fomentar a diversidade por meio da exploração de regiões diferentes, enviesando o processo evolutivo de cada classificador do comitê. Nesse sentido, é imediato pensar no emprego do *bootstrap* (Seção 3.1.1.1) como método de amostragem das populações do **PGMC**, herdando-se assim a proposta do *bagging* no que tange a diversificação inicial dos classificadores.

---

<sup>21</sup>As condições de *suficiência* para a manutenção da diversidade são muito mais intrincadas/complexas e fogem do escopo deste trabalho.

No entanto, o efetivo particionamento do conjunto original de treinamento, como nas propostas do *BagCGPC* [45] e *BoostCGPC* [46], conjugado ou não com o *bootstrap*, soa como uma possibilidade interessante, especialmente no contexto da computação de alto desempenho, e portanto figura como uma variação de amostragem para o **PGMC**.

### 3.3.5 Política de Combinação dos Classificadores

O último estágio do algoritmo **PGMC** é dado pela combinação dos classificadores mais aptos, onde um representante é selecionado em cada população. Em outras palavras, este estágio define a política sob a qual as soluções parciais são agregadas visando a construção de uma solução final mais precisa e completa.

Existem dois esquemas elementares de combinação de classificadores: o simples, por votação majoritária tipo *bagging*, e o mais refinado, por voto ponderado ao modo *boosting*, onde o peso do votante é proporcional à sua qualidade. Dado que o **PGMC**, em decorrência da dinâmica co-evolucionária, opera, ainda que implicitamente, em um processo de avaliação e manutenção de “pesos” similar ao estilo *boosting*, faz sentido adotar uma política de combinação alinhada à política deste. Apesar desta analogia, a ponderação mostra-se uma propriedade bem adequada aos métodos estocásticos, pois estes métodos apresentam dispersão: não raramente execuções independentes produzem resultados com notável variação de qualidade; dessa forma, o voto ponderado atua como instrumento de normalização dessas discrepâncias, evitando que uma solução sub-ótima, por exemplo, tenha a mesma relevância de uma solução ótima.

Como apresentado no Algoritmo 3.3 e Equação 3.2, o *boosting* computa o peso de voto de cada classificador com base no seu desempenho segundo a *distribuição* na qual ele foi treinado. Entretanto, a distribuição de pesos não é uma informação explícita no **PGMC**, dessa maneira, a obtenção de valores correspondentes a pesos requer investigação da dinâmica co-evolucionária. Em outras palavras, é preciso extrair informações sobre a “distribuição de probabilidade” pela qual a população—por conseguinte o melhor classificador—foi treinada. Duas estratégias candidatam-se:

1. *Análise da distribuição da amostragem* – dado um conjunto de treinamento criado, por exemplo, via *bootstrap*, pode-se determinar a probabilidade (peso) de seleção de cada amostra de dados em função do seu número de ocorrências no conjunto.
2. *Contabilização das amostras selecionadas* – os pesos são inferidos, ao final da evolução, pela quantidade de vezes em que cada amostra de dados foi selecionada durante o processo co-evolucionário, ou seja, quanto maior o número de seleções para confronto, maior efetivamente é o peso da amostra.

A primeira hipótese é simplista, pouco exprime a essência co-evolucionária—aliás, ignora-a. Ao contrário, a contabilização das amostras selecionadas no curso da co-evolução representa claramente a distribuição das probabilidades dos pesos do conjunto de treinamento; as amostras mais difíceis são aquelas mais frequentemente selecionadas e, portanto, assim como no *boosting*, são as que têm maior peso. Posto formalmente, o cômputo pelo algoritmo **PGMC** do peso normalizado da  $j$ -ésima amostra de dados da  $n$ -ésima população é dado por:

$$\mathcal{P}_n^j = \frac{\sigma(A_j^{P_n})}{\sum_{i=1}^J \sigma(A_i^{P_n})}, \quad j = 1, \dots, J \quad (3.7)$$

onde  $\sigma(A)$  denota a quantidade de vezes em que a amostra  $A$  foi selecionada (competiu) durante o processo co-evolucionário.

De posse da distribuição de pesos, elege-se como o classificador representante da  $n$ -ésima população,  $C_*^{P_n}$ , aquele indivíduo que minimiza o erro  $\epsilon$  segundo a distribuição  $\mathcal{P}_n$ :

$$C_*^{P_n} = \arg \min_{C_i^{P_n}, i \in I} \epsilon_{C_i^{P_n}} \quad (3.8)$$

onde  $\epsilon_{C_i^{P_n}}$  é definido como:

$$\epsilon_{C_i^{P_n}} = \sum_{j=1}^J \mathcal{P}_n^j \times \llbracket C_i^{P_n}(A_j^{P_n}) \neq y_{A_j^{P_n}} \rrbracket \quad (3.9)$$

e  $y_{A_j^{P_n}}$ , por sua vez, é a classe esperada/correta da amostra  $A_j^{P_n}$ .

Finalmente, definidos todos os membros do comitê  $\{C_*^{P_1}, C_*^{P_2}, \dots, C_*^{P_N}\}$  e suas respectivas medidas de erro  $\{\epsilon_{C_*^{P_1}}, \epsilon_{C_*^{P_2}}, \dots, \epsilon_{C_*^{P_N}}\}$ , a política exata do classificador

final  $C_{final}$  do **PGMC**, via mecanismo de ponderação, pode ser concluída. Uma possibilidade é fazê-la idêntica à política descrita para o *boosting* na Equação 3.3. No entanto, a expressão  $\ln \frac{1-\epsilon}{\epsilon}$  possui dois inconvenientes: (i) erros acima de 0,5 tornam o resultado negativo; e (ii) a expressão é inválida em caso de ausência de erros (100% de precisão de classificação). Uma expressão mais conveniente, também de perfil exponencial, e que delimita os pesos de voto dos classificadores no intervalo  $[0, 1]$  é dada por [60]:

$$\frac{1}{e^{\alpha\epsilon}}, \quad \alpha > 0$$

onde  $\alpha$  é um fator que determina a curvatura da função, isto é, quão relevante/grave é o erro para o cômputo do peso de voto do classificador em questão<sup>22</sup>. Portanto,  $C_{final}$  é em fim descrito por:

$$C_{final}(x) = \arg \max_{y \in Y} \sum_{n=1}^N \left( \frac{1}{e^{\alpha\epsilon_{C_*^{P_n}}}} \right) \mathbb{I}[C_*^{P_n}(x) = y] \quad (3.10)$$

---

<sup>22</sup>A implementação do **PGMC** nesta tese adota  $\alpha = 10$ , cujo valor produziu bons resultados em testes informais.

# Capítulo 4

## Implementação Computacional

Este capítulo descreve os aspectos relevantes da implementação computacional do classificador de dados desenvolvido como parte da contribuição desta tese. O código-fonte do *software*, sob o apelido **gpclassifier**<sup>1</sup>, está disponível publicamente e pode ser obtido, em sua totalidade e sem restrições de uso, no seguinte endereço eletrônico:

<http://sourceforge.net/projects/gpclassifier>

### 4.1 Introdução

O *gpclassifier* é um projeto em contínuo desenvolvimento de um classificador de dados desenhado para a computação de alto desempenho, explorando-a por meio de uma implementação distribuída multi-populacional da programação genética. O paralelismo se dá na evolução concorrente das populações em regime semi-isolado, isto é, as populações eventualmente comunicam-se e trocam materiais genéticos como forma de cooperação em benefício do processo evolucionário geral. Finalmente, as soluções parciais oriundas de cada população são combinadas e o classificador final é formado reunindo-se as qualidades próprias de cada solução.

O sistema *gpclassifier* está disponibilizado sob uma licença livre, a *Licença Pública Geral* na sua terceira versão<sup>2</sup>, e é portanto um *Software Livre*. Este licenciamento garante, em termos gerais, o uso irrestrito do *software*, bem como a permissi-

---

<sup>1</sup>*Genetic Programming Classifier*

<sup>2</sup>**GNU General Public License v3** – <http://www.gnu.org/copyleft/gpl.html>

bilidade da livre distribuição, estudo e modificação (obras derivadas) do código-fonte por terceiros.

#### 4.1.1 A Importância Científica do Software Livre

Percebe-se o nítido alinhamento dos termos da licença adotada neste projeto com os requisitos do desenvolvimento científico, cujo progresso faz-se mediante livre estudo e ampliação do conhecimento existente. Igualmente importante, a publicação do código-fonte de um *software* permite/faculta um dos mais fundamentais princípios da ciência experimental moderna, a validação independente, em outras palavras, o *falsificacionismo* [61].

No artigo intitulado *The Need for Open Source Software in Machine Learning* [62], Sonnenburg *et al.* preconizam a adoção do *software* de código aberto no âmbito da Aprendizagem por Máquina e argumenta a seu favor, além da óbvia e crítica questão da reprodução independente, os pontos:

- facilidade e incentivo às contribuições oriundas de terceiros;
- problemas (“bugs”) podem ser identificados muito mais rapidamente em decorrência do processo de *peer review*;
- comparações com outras técnicas/implementações tornam-se mais diretas;
- novos métodos tendem a ser incorporados e adotados mais rapidamente, seja no contexto acadêmico ou industrial;
- aceleração do avanço científico em decorrência da possibilidade de reuso direto de códigos-fonte existentes;
- disponibilidade e suporte ao *software* perduram mesmo no afastamento do seu autor, situação distinta do que ocorre em sistemas proprietários (código-fonte fechado).

É sob essa bandeira do espírito colaborativo e cientificamente ajustado que esta implementação computacional do *gpclassifier* se faz presente. O restante deste ca-

pítulo discorre sobre as características técnicas das funcionalidades presentes no *software*.

## 4.2 Características e Funcionalidades

O *gpclassifier* foi programado em C++ sob o sistema operacional GNU/Linux e é compatível ainda com qualquer ambiente que implemente as especificações POSIX [63], cuja dominância é absoluta na computação de alto desempenho.

O sistema foi desenvolvido em dois módulos independentes, *learner* e *classifier*, com separação lógica e física (compilação e execução). O primeiro módulo é responsável pelo treinamento—em regime supervisionado—das árvores classificadoras, enquanto que o segundo dedica-se à classificação de dados propriamente dita, e tem como entrada os classificadores anteriormente treinados e as amostras cujas classes devem ser preditas (Figura 4.1).

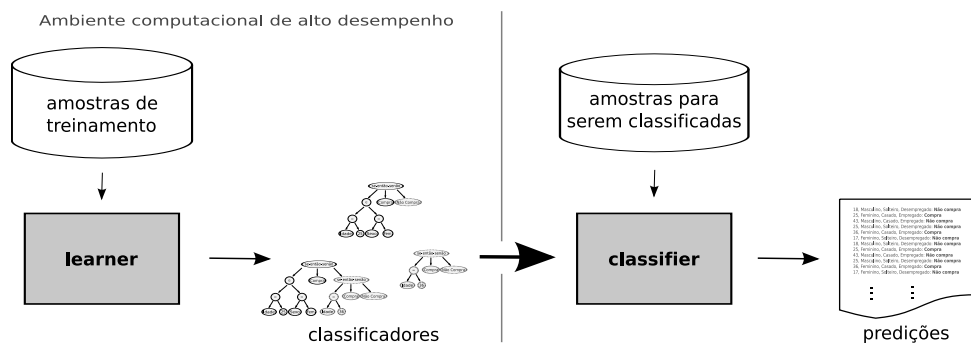


Figura 4.1: Esquema gráfico dos módulos *learner* e *classifier*

Essa divisão também expressa a diferença conceitual entre os módulos. O *learner*, implementando o complexo trabalho de aprendizado, é uma aplicação computacionalmente intensiva e é efetivamente um membro da computação de alto desempenho; já o *classifier*, cuja tarefa, embora não menos importante, resume-se a executar as árvores classificadoras sobre uma coleção de amostras de dados, exige comparativamente pouco poder computacional.

### Módulo *classifier*

A função primária do módulo *classifier* é gerar predições para amostras de dados a partir de classificadores previamente treinados pelo *learner*.

Quando mais de um classificador é fornecido como entrada, ou seja, há um comitê, o *classifier* computa a predição final como uma *combinação* das predições de cada um deles (ou de uma quantidade arbitrária). Duas formas de combinação são possíveis:

- combinação por *voto majoritário* – a classe com maior número de votos é eleita como a classe predita, isto é, cada membro do comitê tem mesmo peso de voto;
- combinação por *voto ponderado* – o peso de voto de cada membro varia— em função de algum mérito, normalmente proporcional à sua qualidade como classificador; a classe predita é, então, aquela que soma o maior valor ao final da votação.

No *classifier* as amostras a serem classificadas podem ser fornecidas de três maneiras: (i) em um único passo, dado uma base de dados; (ii) de forma avulsa como argumentos de linha de comando; ou (iii) incrementalmente em tempo de execução. O último modo é especialmente préstimo em aplicações onde amostras de dados são obtidas por algum dispositivo externo e a predição deve ser imediata; ou ainda, quando uma coleção de amostras não pode ser alocada por inteiro em memória, e devem portanto ser classificadas continuamente, uma a uma.

É possível a utilização do módulo *classifier* simplesmente como avaliador da precisão de classificação do comitê. Esta operação faz-se mediante o fornecimento de amostras de *teste*—aquelas que possuem a classe correta/esperada como um de seus atributos. Estatísticas são então geradas, incluindo precisão de classificação e a matriz de confusão, que reporta o número de *falsos positivos*, *falsos negativos*, *verdadeiros positivos* e *verdadeiros negativos* [64].

Finalmente, as árvores classificadoras que compõem o comitê podem ser impressas (individualmente ou não). Esta funcionalidade exprime uma das eminentes características da programação genética, a direta leitura/interpretação humana de suas soluções e viabiliza assim a aquisição de conhecimento sobre a base de dados em estudo.

## Módulo *learner*

A importante e intensiva tarefa de treinar os membros do comitê é delegada ao *learner*. A fim de resolver o gargalo computacional ao se treinar uma coleção de classificadores, adota-se neste módulo uma abordagem paralela e distribuída de múltiplas populações, onde cada população responsabiliza-se por evoluir sua própria árvore classificadora.

Esse paralelismo é implementado via mecanismo de troca de mensagens e adota como protocolo de comunicação o **MPI**<sup>3</sup> em sua versão 2, referenciado por MPI-2 [65, 66].

O *learner* segue o estilo de reprodução *steady-state* e evita dessa maneira o requerimento de aproximadamente o dobro de memória computacional que seria destinada à alocação de uma segunda população, como no modo *geracional*. Dois tipos de seleção são empregados; na população principal de classificadores usa-se a seleção por *torneio*, enquanto que na população de amostras de dados usa-se a seleção por *ranking*. Os indivíduos classificadores são codificados por intermédio de uma gramática usando a estrutura de dados *árvore*. O *cruzamento* e *mutação*, em suas versões tradicionais, são os dois operadores genéticos adotados. A seguir são sintetizadas as principais características e funcionalidades do *learner*, sendo cada item discutido separadamente nas seções posteriores.

- Representação formal de árvores por *gramática*; pode-se usar vários modelos predefinidos ou definir gramáticas próprias que são declaradas em uma “meta-linguagem” generalizada.
- Reconhecimento e tratamento apropriado dos tipos de atributos *inteiro*, *real*, *lógico*, *nominal* e *ordinal*.
- Especificação aperfeiçoada do formato de arquivo de base de dados.
- Operação em modo sequencial ou distribuído (paralelismo via MPI), onde populações evoluem de forma semi-isolada com topologia e intensidade de migrações configuráveis.

---

<sup>3</sup>*Message Passing Interface* – MPI

- Programação genética co-evolucionária ou canônica.
- Modelo de combinação de classificadores pelo algoritmo **PGMC**<sup>4</sup>, *bagging* ou *boosting*.
- Possibilidade de particionamento e distribuição da base de dados entre as populações a fim de dividir o esforço computacional.
- Sofisticado controle de complexidade (*bloat*) de árvores classificadoras.
- Todos os parâmetros relevantes do sistema podem ser especificados em tempo de execução.

### 4.2.1 Representação Formal por Gramática

A programação genética evolui programas em linguagens arbitrárias, cuja definição depende do domínio particular da aplicação. Linguagens são descritas por gramáticas, que segundo o linguista Noam Chomsky [67] podem ser hierarquizadas em quatro níveis, do mais geral ao mais restrito: as gramáticas *enumeráveis recursivamente*, *sensíveis ao contexto*, *livre de contexto*, e *regulares*. Destaca-se, contudo, no âmbito computacional, a gramática livre de contexto (GLC), cuja simplicidade aliada ao marcante poder de expressão fazem dela o tipo de gramática predileta de virtualmente todas as linguagens de programação de alto nível existentes atualmente.

Uma gramática estabelece as regras e princípios que regem os vários elementos de uma linguagem. Mais precisamente, as gramáticas são dispositivos de geração de sentenças, que nada mais são do que cadeias de caracteres que satisfazem às regras gramaticais. Uma linguagem é, pois, definida pelo conjunto, possivelmente infinito, de todas as sentenças válidas passíveis de serem produzidas por uma gramática.

As linguagens recorrentes na programação genética incluem, por exemplo: linguagem de fórmulas matemáticas, para regressão simbólica e ajuste de curvas; linguagem com instruções lógicas, condicionais e relacionais, para classificação de padrões; e linguagens mais amplas, adicionando laços iterativos e desvios condicionais.

---

<sup>4</sup>O algoritmo **PGMC** é descrito no Capítulo 3.

Tradicionalmente, a programação genética define a linguagem de evolução servindo-se de um mecanismo simplista e limitado, os populares conjuntos de *funções*, fornecendo os operadores, e de *terminais*, fornecendo os operandos (vide Seção 2.3). Nestes, salvo a restrição de casamento do número de argumentos (aridade), efetivamente não há regras; quaisquer combinações entre operadores e operandos, e operadores e eles mesmos, devem ser factíveis tanto sob o ponto de vista *computacional* como *conceitual*. Ser computacionalmente factível significa obedecer restrições lógicas e matemáticas, como por exemplo garantir que o fatorial seja aplicado somente em números naturais, ou que a divisão nunca opere em um denominador zero. Por factibilidade conceitual entende-se que semanticamente a combinação deve ser válida; por exemplo, *somar* um valor *numérico* a um valor *lógico* é conceitualmente inválido, assim como, em termos mais abstratos, dizer que a *idade* de alguém (atributo numérico) seria *menor* ou *maior* do que sua *cidade natal* (atributo nominal).

A fim de garantir que um programa permaneça factível ao longo de todo o processo evolucionário, introduziu-se na programação genética, em sua forma canônica, o requisito de *consistência*. Todas as constantes, variáveis, argumento de funções e valores de retorno devem ser do mesmo tipo de dados para que as operações genéticas (p. ex., cruzamento e mutação) possam realizar-se em qualquer ponto da estrutura das soluções candidatas. A consistência é um conceito simples e uniforme, porém, ela cria dois problemas principais: (i) devido ao fato de que funções e terminais podem ser combinados com quaisquer outros, e muitas dessas combinações não fazem sentido em termos conceituais, haverá um aumento desnecessário e indesejável no tamanho do espaço de busca [68, 69] em decorrência da assimilação dessas regiões estéreis; e (ii) a consistência é uma propriedade difícil, e às vezes impossível, de ser satisfeita em problemas que requerem diferentes tipos de dados—a classificação de dados é um desses problemas.

Alternativas foram propostas para lidar com o requisito da consistência na PG canônica [8, 16, 15]; uma comparação entre as vantagens e desvantagens de algumas das principais propostas podem vistas em [70, 55, 18]. O *learner* adota como representação dos programas a consolidada alternativa desenvolvida por Whigham [17, 71, 70, 55], que (i) faz uso da gramática livre de contexto como

instrumento de garantia da factibilidade das estruturas frente às operações genéticas, (ii) admite a definição de relações semânticas complexas, e (iii) permite ainda que a evolução dê-se transparentemente seja qual for a linguagem descrita pela GLC. A gramática livre de contexto, sua integração com a programação genética e implementação são descritas no decorrer desta seção.

#### 4.2.1.1 Definição Formal

Formalmente as gramáticas são representadas por uma quádrupla  $G = (N, \Sigma, S, P)$  onde

- $N$  é o alfabeto (conjunto finito) de símbolos **não-terminais**.
- $\Sigma$  é o alfabeto de **terminais**<sup>5</sup>, tal que  $N \cap \Sigma = \emptyset$  e  $N \cup \Sigma = V$ , onde  $V$  é o conjunto de símbolos da linguagem.
- $S$  é o símbolo **inicial** ou de partida,  $S \in N$ .
- $P$  é o conjunto de **regras de produção** na forma  $\alpha \rightarrow \beta$ , onde  $\alpha \in N$  e  $\beta \in V^*$ .  $V^*$  denota o conjunto de todas as sequências compostas por elementos de  $V$ .

Os símbolos não-terminais são aqueles que expressam conceitos, e que inevitavelmente serão substituídos por símbolos terminais para a constituição de uma sentença. Na gramática da língua portuguesa, por exemplo, o substantivo *verbo* é um não-terminal que expressa a classe de palavras (terminais) que compartilham o conceito de verbos; da mesma forma, o não-terminal *<numérico>* de uma GLC provavelmente denotaria uma classe relacionada aos números, tal como constantes numéricas, variáveis/atributos e outras. Além disso, um não-terminal pode expressar outro não-terminal; por exemplo, o *<numérico>* poderia ter subclasses de não-terminais do tipo *<inteiros>* e/ou *<reais>*.

---

<sup>5</sup>O significado da palavra *terminal* usado aqui difere de seu significado na programação genética. Na PG canônica, um *terminal* denota uma constante, variável, ou uma função que não requer argumentos—isto é, um nó folha da árvore do programa. Mas no contexto gramatical, um *terminal* tem significado mais geral e pode ser não apenas um nó folha, mas também funções que requerem argumentos (nós internos).

Por outro lado, os símbolos terminais denotam significados concretos. Em uma certa gramática o número  $\pi$  e a função *seno* (que retorna um valor numérico), por exemplo, poderiam ser símbolos terminais vinculados ao não-terminal  $\langle \text{numérico} \rangle$ .

As regras de produção definem como os não-terminais relacionam-se com terminais e outros não-terminais; são escritas como  $\alpha \rightarrow \beta$ , que significa que o não-terminal  $\alpha$  pode ser substituído pela sequência de símbolos contidos em  $\beta$ . A primeira regra é, na verdade, dada pelo símbolo de partida  $S$ , que especifica como a sentença deve ser iniciada. O poder expressivo da GLC reside em suas regras de produção, e este poder é efetivamente explorado quando as regras de produção são recursivamente combinadas entre si para a formação de sentenças arbitrárias. As regras podem ser livremente combinadas, desde que a cabeça de uma regra (o não-terminal  $\alpha$ ) coincida com algum símbolo não-terminal pertencente ao corpo ( $\beta$ ) de outra regra; chama-se de *compatível* a regra de produção cuja cabeça coincide com pelo menos um símbolo não-terminal do corpo de uma outra regra.

## Passos de Derivação

O passo de derivação é o processo de obtenção de uma nova sequência de símbolos—também conhecida como *palavra* ou *cadeia de caracteres*—mediante a aplicação de uma produção a um símbolo não-terminal da sequência sendo derivada. Em outras palavras, é a transformação de um estado em outro substituindo-se um não-terminal por uma produção compatível.

A derivação inicia-se pela aplicação de uma produção compatível sobre o símbolo inicial  $S$ . Se a sequência gerada contém pelo menos um símbolo não-terminal, então o processo de derivação continua, recursivamente, até que todos os símbolos não-terminais sejam substituídos por terminais<sup>6</sup>; neste ponto diz-se que a forma sentencial foi alcançada.

Uma propriedade interessante acerca da derivação é que a sequência de derivações pode ser expressa na forma de uma árvore. Isso torna a integração da GLC com a programação genética bastante natural. De fato, a codificação gramatical de um indivíduo é na realidade uma sequência completa de passos de derivação dispostos

---

<sup>6</sup>A cada estágio de derivação substitui-se o não-terminal mais à esquerda.

hierarquicamente em árvore—a ela dá-se o nome de *árvore sintática* ou *de derivação*.

#### 4.2.1.2 Integração com a Programação Genética

##### Gramática do problema

A definição de uma gramática é a primeira tarefa a ser realizada na modelagem de um problema. Em aplicações típicas da PG, uma GLC define funções, constantes, variáveis, bem como as relações e restrições entre estes elementos. Preferencialmente, as gramáticas deveriam gozar de poder suficiente (flexibilidade) para expressar a solução teórica do problema, ao mesmo tempo em que deveriam limitar tanto quanto possível o espaço de busca, admitindo, via regras de produções, somente construções atraentes para o problema.

Problemas de classificação de dados, por exemplo, normalmente incluirão regras de produção tratando de condicionais (p. ex. *se-então-senão*), operações lógicas (p. ex. *ou, e, não*), relacionais (p. ex. *menor, maior, igual*), e eventualmente funções aritméticas visando manipular dados numéricos.

##### Criação dos indivíduos

Cada indivíduo contém uma árvore de derivação completa, incluindo os símbolos não-terminais da derivação<sup>7</sup>. Na população inicial as árvores são geradas aleatoriamente, isto é, durante o processo de derivação qualquer regra de produção, desde que compatível com o não-terminal a ser substituído, tem a mesma chance de ser escolhida.

##### Avaliação da aptidão

A introdução de uma GLC não altera o método de avaliação da aptidão, mas neste estágio apenas os *símbolos terminais*, a sentença, atuam como unidade funcional (executável). Por mais que os símbolos não-terminais façam parte da árvore de um programa, eles não são requeridos em tempo de execução; eles agem apenas

---

<sup>7</sup>Os símbolos não-terminais são armazenados juntamente para assegurar que a factibilidade dos indivíduos seja preservada em meio às operações genéticas.

como garantidores da integridade semântica das estruturas perante as operações genéticas.

Na dinâmica co-evolucionária os classificadores são executados em amostras avulsas de dados, e a aptidão é proporcional à média de classificações corretas tomadas em um certo período (Seção 3.3.2); na avaliação tradicional, no entanto, um classificador é executado em todas as amostras de treinamento e, analogamente, sua aptidão é proporcional à precisão das predições.

## Operadores genéticos

Dois dos principais operadores genéticos na programação genética são o *cruzamento* e a *mutação*. O primeiro recombina materiais genéticos bem sucedidos na esperança de criar indivíduos melhor adaptados; o segundo (re)introduz materiais genéticos (aumenta a diversidade) causando pequenas perturbações aleatórias na estrutura de alguns indivíduos. Na concepção típica, o cruzamento comuta subárvores quaisquer de dois indivíduos (os pais), enquanto a mutação substitui uma subárvore qualquer de um indivíduo por uma inteiramente nova (gerada aleatoriamente).

Com o objetivo de preservar a factibilidade dos indivíduos, é necessário que cada operador genético respeite as regras de produção da gramática. Em outras palavras, todos os operadores genéticos devem obedecer às restrições impostas pelos não-terminais de uma árvore de derivação. Por conseguinte, no cruzamento e mutação, as subárvores sendo comutadas/substituídas devem possuir o mesmo não-terminal como nó raiz.

A Figura 4.2 ilustra o cruzamento entre os indivíduos **A** e **B**. Primeiro, um nó contendo um não-terminal é aleatoriamente selecionado na árvore **A**, no caso o símbolo *<class>*. Em seguida, na árvore **B**, um não-terminal *com o mesmo rótulo* é escolhido aleatoriamente. Finalmente, as subárvores enraizadas por *<class>* são trocadas, criando portanto dois novos indivíduos, **A'** e **B'**.

O operador de mutação é exemplificado na Figura 4.3. Novamente, um símbolo não-terminal aleatório é selecionado em **A**, por exemplo *<class>*. Então, a subárvore enraizada por este não-terminal é substituída por uma subárvore inteiramente nova,

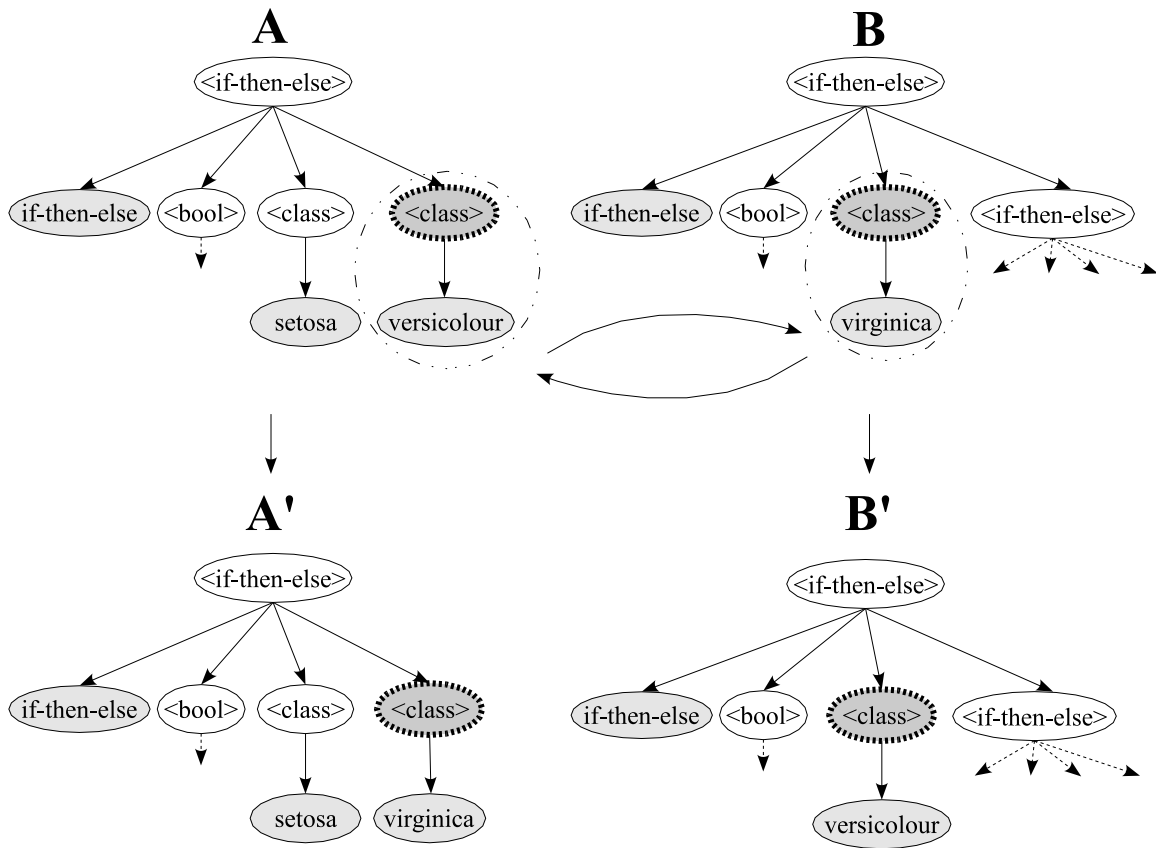


Figura 4.2: Cruzamento gramatical

cujo nó raiz é também  $\langle class \rangle$ . É oportuno salientar que o não-terminal selecionado fez o papel do símbolo inicial ( $S = \langle class \rangle$ ) no processo de derivação que deu origem à nova subárvore.

#### 4.2.1.3 Gramáticas predefinidas no *learner*

As gramáticas GLC predefinidas no *learner* governam as relações entre todos elementos de uma base de dados, ou seja, atributos, constantes, classes e tipos de dados. São implementadas três espécies de gramáticas, cada qual em um nível de complexidade.

##### Atributo-valor

Na mais simples das gramáticas, a *atributo-valor*, as comparações são limitadas entre o *atributo* e sua faixa possível de *valores* (constantes). Este tipo de gramática assemelha-se à estrutura das *árvores de decisão*, popularizadas pelo algoritmo

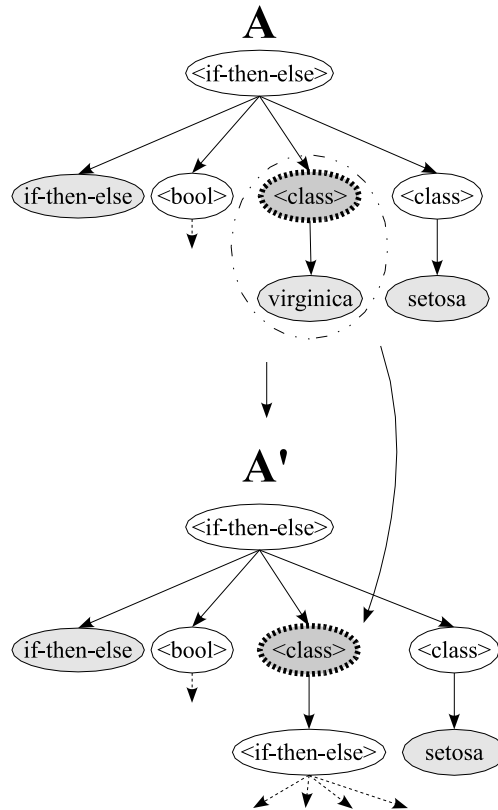


Figura 4.3: Mutaç o gramatical

C4.5 [72]. Uma construo admitida, por exemplo, seria:

se *largura* < 0,5 ento ...

onde *largura*   um atributo num rico qualquer. A gram tica atributo-valor, por ser a mais restrita, tem a vantagem de descrever um espao de busca mais conciso.

### Inter-atributo

Situando-se em um n vel intermedi rio de complexidade, a gram tica *inter-atributo*, al m de permitir formaes atributo-valor, aceita comparaes diretas entre os atributos. Uma situao poderia ser:

se *largura* < *comprimento* ento ...

em que a largura de alguma medida   comparada com o comprimento. Este tipo de construo pode ter papel determinante no que tange a *preciso* de classificao e a *complexidade* dos classificadores;   bem poss vel que o conhecimento impl cito em uma base de dados siga a formao inter-atributo, e portanto s  possa ser adequa-

damente (parcimoniosamente) expresso valendo-se de construções dessa forma.

### Aritmética

Finalmente, a gramática *aritmética* acrescenta a possibilidade de realização de operações aritméticas envolvendo atributos e constantes. Uma das construções poderia ter a forma:

$$\text{se } largura^2 < 3 \times (\text{comprimento} - 0,5) \text{ então...}$$

O raciocínio no que concerne aos casos onde a essência de uma base de dados só pode ser expressa por formações mais requintadas é análogo ao discutido para a gramática *inter-atributo*; no entanto, a gramática *aritmética* vai mais longe e é capaz de expressar diretamente uma maior complexidade. Contudo, o preço a ser pago é o inchaço considerável do espaço de busca, o que pode seriamente retardar o processo evolutivo.

#### 4.2.1.4 Meta-linguagem de definição de gramáticas

Em termos estritos, a definição de uma gramática é específica a uma determinada base de dados. Este fato é um reflexo direto do que vem a constituir uma gramática, isto é, dos elementos que compõem a quádrupla  $(N, \Sigma, S, P)$ , em especial o conjunto de terminais  $\Sigma$ . Os atributos, faixas de constantes<sup>8</sup> e classes de uma base de dados particular são todos símbolos *terminais* pertencentes a  $\Sigma$ . Isto implica que, em princípio, não há como portar diretamente uma gramática construída sobre uma certa base de dados para outra base de dados, exceto se os *cabeçalhos* (quantidade e tipo dos atributos, faixas de constantes, e número de classes) coincidirem.

Poderia-se pensar em simplesmente substituir os terminais relativos à uma base de dados (o cabeçalho) de uma certa gramática pelos terminais de uma outra base de dados, modificando-se apenas  $\Sigma$ . Infelizmente, alterações no conjunto de terminais podem invalidar tanto símbolos não-terminais em  $N$  como as produções em  $P$  e deixar a gramática em um estado inoperante devido à quebra das regras de produção. Por exemplo, se a base de dados anterior para a qual a gramática foi definida possui

---

<sup>8</sup>As faixas de constantes referem-se ao conjunto de todos os valores declarados por atributos, em particular os nominais e ordinais.

atributos booleanos e a nova não, então todos os símbolos não-terminais e regras de produção relativos, direta ou indiretamente, aos atributos booleanos tornam-se inválidos.

O *learner* soluciona esse problema introduzindo uma meta-linguagem de definição de gramáticas, que a grosso modo funciona assim: (i) as gramáticas predefinidas são descritas sob uma linguagem que constrói *dinamicamente* os elementos da quádrupla  $(N, \Sigma, S, P)$  com base nas informações disponíveis no cabeçalho das bases de dados; (ii) quando uma base de dados é fornecida ao sistema, a descrição do seu cabeçalho é coletada e a gramática é então construída; (iii) finalmente, eventuais inconsistências nas regras de produção são podadas recursivamente.

## 4.2.2 Tratamento dos Tipos de Atributos

Existem três classes fundamentais de tipos de atributos [29, 73]: *lógica*, *numérica* e *categórica*. Dentro da classe numérica tem-se os tipos *inteiro* e *real*, por sua vez, a classe categórica subdivide-se nos tipos *nominal* e *ordinal*. Os tipos básicos, portanto, somam cinco. Cada um desses tipos é tratado apropriadamente pelo *gpclassifier*, como consequência, as relações semânticas são preservadas ao mesmo tempo em que o espaço de busca é reduzido.

O atributo lógico representa informações binárias da forma *verdadeiro/falso* ou *sim/não*. Estes atributos são restritos pelas regras de produção das gramáticas predefinidas a atuarem como operandos somente de operadores booleanos, como *conjunção*, *disjunção* e *negação*.

As gramáticas predefinidas tratam os tipos numéricos de maneira quase idêntica entre si; são intercambiáveis e submetem-se, dependendo da amplitude da gramática, às operações relacionais (abrangendo constantes ou atributos) e aritméticas. São discriminados, contudo, em um detalhe no que concerne à criação de *constantes efêmeras*<sup>9</sup>: enquanto que o tipo inteiro produz constantes numéricas inteiras, o real produz constantes numéricas contínuas—os intervalos destas constantes, que variam

---

<sup>9</sup>A constante efêmera é um terminal especial que torna possível a criação de uma infinidade de constantes numéricas (inteiras ou contínuas) no curso do processo evolutivo; sempre que é inserida na estrutura de um indivíduo (p. ex. no ato de sua iniciação) ela assume um valor aleatório que permanece imutável durante toda a sua existência.

de atributo para atributo, são extraídos automaticamente da base de dados.

Os atributos categóricos são aqueles que especificam uma lista finita de elementos discretos como a faixa admissível de valores. Os subtipos *nominal* e *ordinal* diferem no que diz respeito à existência ou não da noção de *ordem/preferência* entre seus elementos. Os valores de um atributo ordinal são classificados segundo um critério particular, ainda que não exista a ideia de distância absoluta entre seus elementos— não se pode medir quão próximo dois vizinhos estão; exemplos de atributos ordinais incluem o *grau de escolaridade* e *posição hierárquica* de uma pessoa. Por outro lado, os valores dos atributos nominais não exibem qualquer noção preferencial entre eles, e portanto descrevem apenas uma coleção não ordenada de valores categóricos; a *cor* de um objeto e a *cidade natal* de alguém são exemplos de atributos nominais.

Dada a natureza dos atributos ordinais, são admitidas todas as variações dos operadores relacionais, por exemplo  $=$ ,  $\neq$ ,  $<$  e  $>$ , seja qual for a complexidade da gramática predefinida. Diferentemente, aos atributos nominais não há justificativa conceitual para a admissão de comparações que vão além da *igualdade* e *desigualdade* (ou operações derivadas destas). Também não são conceitualmente aceitáveis, em ambos atributos nominais e ordinais, as comparações *entre atributos* e operações *aritméticas*—tais construções são proibidas nas gramáticas predefinidas.

### 4.2.3 Formato da Base de Dados

O formato da base de dados (conjunto de treinamento) empregado no *learner* é derivado do formato ARFF<sup>10</sup> nativo do *software Weka* [29], conseqüentemente semelhante a este, mas torna-o mais conveniente em alguns detalhes e modifica-o de maneira a introduzir informações relevantes ao processo evolucionário acerca da base de dados.

As alterações que visam aprimorar a riqueza de informações extraídas da base de dados constituem na (i) adição do tipo de dados *lógico* e *ordinal*, e (ii) na distinção entre o tipo de dados *inteiro* e *real* (ver Seção 4.2.2).

Como medida de conveniência, o formato da base de dados torna opcional a

---

<sup>10</sup> *Attribute-Relation File Format* – ARFF

declaração explícita dos valores (nomes) dos atributos nominais e da classe; no processo de leitura da base de dados, estes valores são automaticamente extraídos e catalogados<sup>11</sup>.

#### 4.2.4 Evolução Sequencial e Distribuída

Muito embora o *learner* seja um genuíno programa computacional de alto desempenho, muitas vezes é desejável a evolução em sequencial, tanto por questões de interpretabilidade das soluções—normalmente quanto maior o número de classificadores em um comitê mais difícil torna-se a extração de conhecimento—como por eventuais restrições computacionais. O *learner* suporta tanto a evolução sequencial quanto a distribuída/paralela em uma quantidade arbitrária de populações (“ilhas”).<sup>12</sup>

O modo distribuído, por sua vez, configura-se em três aspectos: (i) o número de ilhas; (ii) a topologia de conexão entre elas; e (iii) a intensidade de comunicação, isto é, de troca de material genético.

O número de ilhas define o número de evoluções concorrentes, que então determina o tamanho do comitê<sup>13</sup>. O grau de paralelismo deve ser estabelecido considerando-se a dificuldade do problema em questão e os recursos computacionais disponíveis.

A troca de material genético acontece por migrações de indivíduos (árvores classificadoras) selecionados proporcionalmente em função de suas aptidões, a cada geração, em uma frequência configurável. As comunicações são *assíncronas*, o que significa que o processo evolucionário em uma certa população não é interrompido à espera por imigrantes.

A topologia de conexão entre ilhas, descrita na seção que se segue, demarca a rota e o fluxo das migrações e pode também interferir nas suas intensidades.

---

<sup>11</sup>Na verdade, o Weka possui utilitários de conversão que se encarregam de extrair e catalogar os valores de atributos nominais, no entanto tal operação demanda um passo extra e não é nativa do formato ARFF.

<sup>12</sup>Sempre quando não claramente especificado, o termo *população* assume, neste trabalho, o mesmo significado do termo *ilha*, sendo portanto sinônimos.

<sup>13</sup>Apenas o modo co-evolucionário (algoritmo **PGMC**) e *bagging* são passíveis de evolução paralela; o *boosting* é intrinsecamente sequencial (ver Seção 4.2.6).

#### 4.2.4.1 Topologia de Ilhas

As topologias de conexão entre ilhas implementadas no *learner* são topologias virtuais, e como tais independem da disposição física dos nós/processadores do *hardware* computacional. São atualmente disponíveis em seis variedades, como mostradas nas Figuras 4.4, 4.5 e 4.6.

O tipo de topologia *fila* impõe uma rota encadeada, mas sem conexão entre a última ilha e a primeira. A primeira versão (Figura 4.4a) possui fluxo direcionado, onde uma determinada ilha comunica-se unilateralmente com a ilha posterior. Já a segunda versão, visualizada na Figura 4.4b, não faz restrições quando à direção, e admite comunicação bilateral entre ilhas adjacentes.

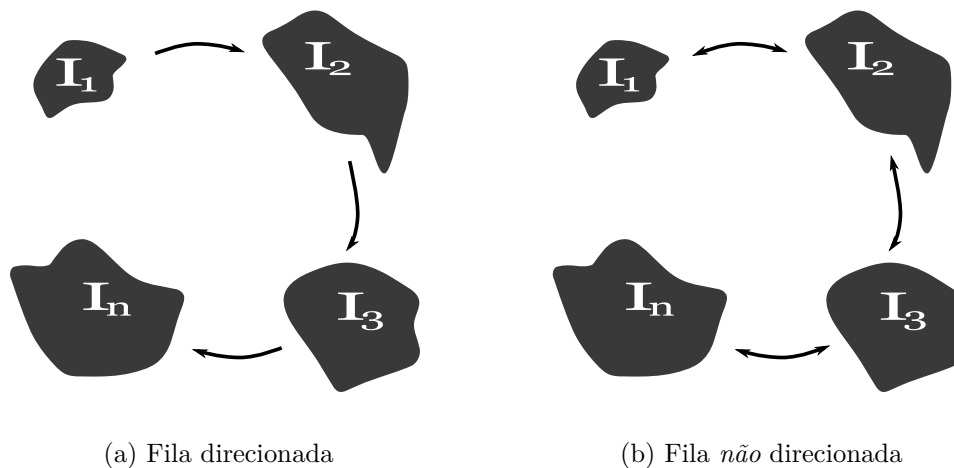


Figura 4.4: Topologia de ilhas I

Em comparação com as demais disposições topológicas, a fila apresenta a menor intensidade de comunicação, especialmente na versão direcionada. Compartilha-se portanto menos material genético e tende-se a preservar melhor a diversidade de cada ilha, ainda que arriscando-se, por outro lado, a retardar o processo evolucionário. Curioso notar que na fila direcionada a primeira ilha em nenhum momento recebe imigrantes, enquanto que as subsequentes recebem, cumulativamente, os materiais provenientes das ilhas anteriores.

Uma outra variedade de topologia é a chamada *anel* que, como se espera, define uma conexão anelar. É uma generalização da fila e permite que as ilhas das extremidades se comuniquem assim como as demais. Analogamente, a topologia

anel subdivide-se nas versões direcionada e não direcionada, como mostram as Figuras 4.5a e 4.5b, respectivamente.

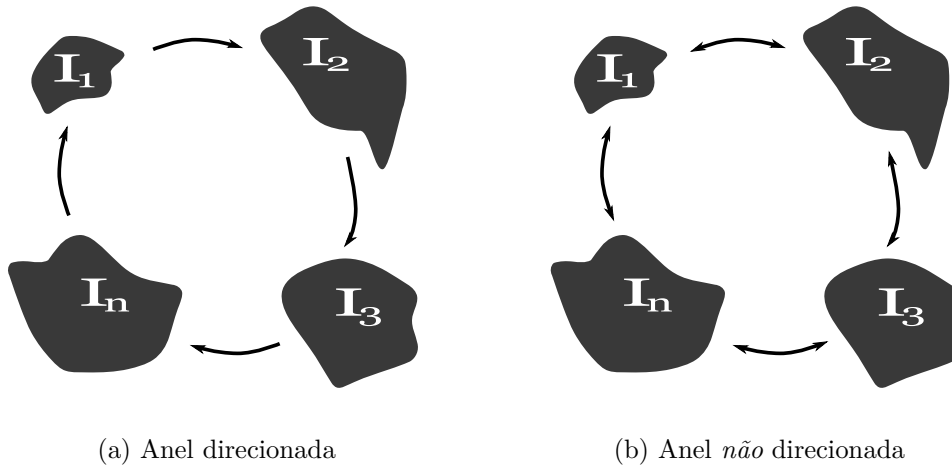


Figura 4.5: Topologia de ilhas II

Em termos de dinâmica evolucionária, a principal diferença entre a topologia fila e a anel é que esta última potencialmente “retorna” à ilha inicial o material genético eventualmente depositado/evoluído da última ilha. Nas versões não direcionadas a diferença prática é pouco perceptível, já que o fluxo bidirecional permite, mesmo que indiretamente, que o material da última retorne à primeira.

Uma topologia *totalmente conectada* também é implementada no *learner*, e nesta não há qualquer restrição quanto à rota e fluxo de comunicação (Figura 4.6a). É uma conexão, no entanto, favorável à indesejável convergência prematura, já que soluções sub-ótimas emergidas localmente em uma ilha propagam-se diretamente para as demais. Dependendo das frequências de migração estabelecidas, esta topologia pode se comportar na prática como uma “única” grande ilha, um arquipélago de fluxo intenso, e anular os benefícios da evolução semi-isolada.

Finalmente, uma sofisticada disposição topológica é apresentada na Figura 4.6b. A topologia *globo* simula a distribuição natural, em que ilhas e continentes são espalhados ao longo do globo terrestre. Esta ilha implementa o conceito *chance de sobrevivência*: quando mais próximo duas ilhas estão, maior a chance/frequência de um dado indivíduo conseguir migrar (“completar a travessia”) de uma ilha a outra.

Muito embora o globo seja uma topologia totalmente conectada e de fluxo bidire-

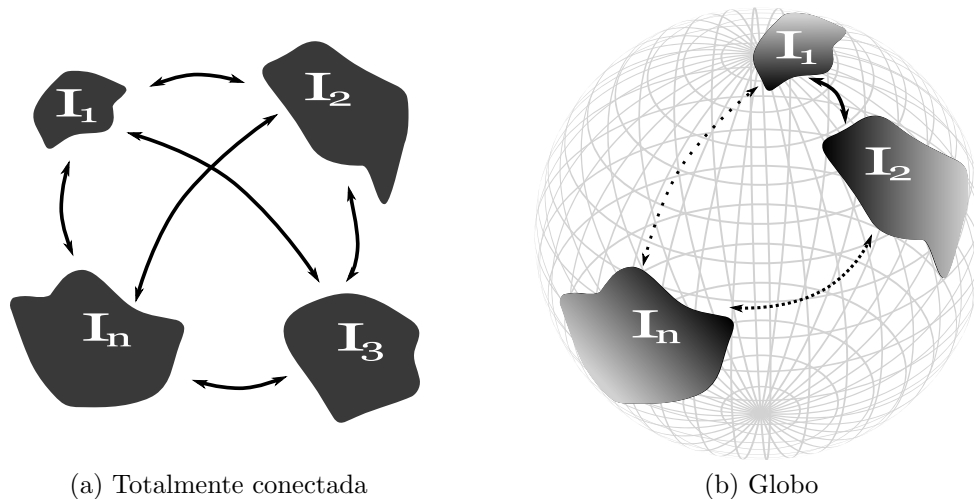


Figura 4.6: Topologia de ilhas III

cional, a distância entre as ilhas—fixadas aleatoriamente a cada execução—delimita e desbalanceia a intensidade das migrações. Esta perturbação tende a introduzir diversidade na dinâmica evolucionária, que induz ilhas a desenvolverem perfis evolucionários próprios.

#### 4.2.5 Evolução Canônica e Co-evolucionária

Como descrito na seção anterior, há dois modos básicos de operação no *learner*: sequencial e distribuído. Não obstante, estes modos operam, cada um, tanto sob a dinâmica evolutiva por programação genética canônica (Capítulo 2) ou co-evolucionária (Seção 3.3.2). Isto significa que, não incluindo os algoritmos específicos de combinação de classificadores (ver Seção 4.2.6), os modos triviais de operação do *learner* totalizam quatro: sequencial canônico, sequencial co-evolucionário, distribuído canônico e distribuído co-evolucionário.

A forma ordinária da evolução distribuída canônica e co-evolucionária implementa o paralelismo à moda tradicional segundo o modelo semi-isolado de ilhas; embora várias populações evoluam simultaneamente, ao final interessa somente uma única solução, em outras palavras, a árvore classificadora mais bem sucedida dentre todas as populações.

## 4.2.6 Método de Combinação de Classificadores

Somando-se aos modos operativos descritos na seção anterior, o *learner* implementa ainda três operações relativas à *formação de comitê* (vide Seção 1.2), cujo classificador final é dado por uma combinação das predições dos membros do comitê; são elas: *bagging*, *boosting* e **PGMC**.

A implementação no *learner* dos algoritmos de combinação de classificadores *bagging* e **PGMC** segue suas respectivas definições; o *bagging* é apresentado na Seção 3.1, enquanto que o algoritmo **PGMC** é descrito no Capítulo 3.

Para o *boosting*, todavia, o *learner* implementa a extensão ao *AdaBoost* denominada *SAMME*, proposta por Ji Zhu *et al.* [74]. O *SAMME* relaxa o rigoroso critério de parada das iterações em problemas de múltiplas classes de  $\epsilon > \frac{1}{2}$  para  $\epsilon > 1 - \frac{1}{K}$ , onde  $\epsilon$  é o erro ponderado do classificador em uma certa iteração e  $K$  é o número de classes do problema. Análises teóricas e experimentais providas pelos autores garantem o funcionamento do *SAMME*.

## 4.2.7 Particionamento da Base de Dados

Normalmente os algoritmos de combinação de classificadores não particionam a base de dados de treinamento, apenas reamostram os dados segundo uma determinada distribuição de probabilidade, portanto, mantém-se a cardinalidade original da base de dados em cada uma das populações. Tipicamente, esta prática garante redução incremental nos erros de treinamento conforme o número de populações cresce, à custa, porém, da multiplicação do esforço computacional.

Uma alternativa é o particionamento efetivo da base de dados, ou seja, se existem  $s$  amostras de dados e  $p$  populações, cada população se responsabilizará pelo treinamento de  $s/p$  amostras de dados. Percebe-se claramente que esta abordagem fraciona o tempo computacional em  $p$  vezes quando executada em um igual número de processadores independentes<sup>14</sup>.

---

<sup>14</sup>Dado que a programação genética exhibe paralelismo natural e o *learner* implementa um modelo eficiente de paralelismo assíncrono de baixa comunicação, a curva de *speedup* [75] aproxima-se da linearidade.

Existe, por outro lado, um sério risco do uso do particionamento comprometer a precisão de classificação, dependendo do grau de paralelismo e características da base de dados em questão, sobretudo se a quantidade de amostras  $s$  é pequena.

#### 4.2.8 Controle de Complexidade – *bloat*

Um dos notáveis desafios da programação genética, e que tem recentemente demandado expressivo esforço de pesquisa, é o problema de *controle de complexidade*<sup>15</sup> das soluções candidatas (programas de computador) no decorrer do processo evolucionário.

Por basear-se em estruturas de tamanho variável e ilimitado, como idealmente são representadas as soluções candidatas na programação genética, existe o risco iminente—na realidade uma tendência [76]—ao crescimento *inútil* e desordenado do tamanho médio destas estruturas no curso da evolução; este fenômeno é conhecido como *bloat*. Várias teorias foram elaboradas na tentativa de prover explicações para o fato [77, 78, 79, 80, 81, 82, 83, 84]; no entanto, muito embora certas explicações soem mais fidedignas que outras, nenhuma das teorias até então propostas foi capaz de elucidar o problema do *bloat* de maneira segura e completa.

Os problemas decorrentes do aumento indesejável dos programas são óbvios; envolvem a explosão da necessidade de *recursos computacionais* em tempo e espaço, e a degradação na *legibilidade* e capacidade de *generalização* das soluções.

A despeito do esforço teórico direcionado ao entendimento do *bloat*, há uma mobilização concomitante em torno do controle prático da complexidade das soluções candidatas. As abordagens propostas podem ser categorizadas em três frentes [18]: (i) controle rígido do tamanho e profundidade das estruturas; (ii) operadores genéticos especificamente desenvolvidos; e (iii) pressão de seleção contrária à complexidade.

No primeiro método [8], o controle rígido de tamanho/profundidade, limites máximos—e às vezes mínimos—são impostos sobre todas as operações que criam

---

<sup>15</sup>Neste contexto, o termo *complexidade* é definido simplesmente como alguma medida objetiva diretamente proporcional ao tamanho da estrutura a qual se refere e é normalmente equivalente ao *número de nós* nas representações por árvore.

ou modificam as estruturas dos programas, isto é, usualmente inicialização, cruzamento e mutação. Estes limites são facilmente implementados na inicialização, porém, são inadequados quando impostos sobre a mutação e, especialmente, o cruzamento; nestes operadores a existência de limites esbarra na natureza estocástica da programação genética, cujas regiões de atuação dos operadores são escolhidas aleatoriamente, culminando no fracasso (desperdício) de diversas tentativas que infringiriam os limites. Ademais, o controle rígido transfere ao especialista a responsabilidade de garantir rigorosamente que o limite máximo estabelecido é suficiente para descrever a solução do problema.

Aplicar operadores genéticos especializados é uma outra variação de controle de complexidade. Estes operadores atuam no âmbito do cruzamento e mutação e asseguram que um limite pré-estabelecido não seja transgredido ou, em alguns casos, que não seja transgredido além de uma certa margem. Incluem, por exemplo, operadores de cruzamento que restringem a recombinação apenas às partes de tamanhos compatíveis [85], ou operadores de mutação que tanto efetivamente encolhem as soluções candidatas ou proíbem alterações que acresçam complexidade [86, 87]. Muito embora o método de especialização dos operadores genéticos ofereça uma alternativa aprimorada ao controle rígido de complexidade, a operação de cruzamento ainda é restrita e induz um viés na dinâmica de recombinação do material genético—os tamanhos das estruturas a serem recombinadas devem ser iguais/semelhantes.

A terceira abordagem adota uma postura mais flexível e natural, e delega o controle de complexidade exclusivamente à função de seleção [8, 88]. Agora, ser um indivíduo apto significa, dentre outros quesitos, ser parcimonioso; a solução ideal é aquela que melhor conjuga *qualidade “bruta”*, que poderia ser a acurácia supondo a tarefa de classificação de dados, com *complexidade*, por exemplo o número total de nós de uma árvore classificadora. Ao invés de se estabelecer diretamente limites de complexidade, especifica-se o peso no qual soluções serão proporcionalmente penalizadas conforme a complexidade—indivíduos complexos tendem a desaparecer sob pressão seletiva, salvo se a complexidade for justificada considerando-se os demais quesitos sendo avaliados. O método por pressão de seleção redefine a função original

de aptidão  $f(x)$  como:

$$f'(x) = f(x) - c \times \ell(x)$$

onde  $x$  é um indivíduo da população,  $c$  o coeficiente (peso) de penalização e  $\ell(x)$  a complexidade (tamanho) do indivíduo  $x$ .

O controle de complexidade por pressão parcimoniosa dispensa quaisquer modificações no que diz respeito aos operadores genéticos, logo, tais operações realizam-se livres de restrições. A real questão, entretanto, incide sobre o ajuste do valor do coeficiente de penalização  $c$ ; mais especificamente, em como defini-lo de maneira que a pressão de seleção atue perspicazmente ao longo de todo o processo evolutivo. Habitualmente  $c$  é ajustado como uma constante e permanece portanto imutável durante toda a evolução. No entanto, a dinâmica incerta do processo evolucionário faz desta aparentemente simples fixação do coeficiente  $c$  um modo de ajuste sensível e instável. Uma constante pouco acima da ideal pode ser o bastante para estagnar a evolução por forte penalização e produzir soluções extremamente compactas mas sem qualidade; por outro lado, um ajuste pouco abaixo do ideal seria suficiente para anular inteiramente o efeito prático da pressão parcimoniosa e assim ser incapaz de controlar o *bloat*.

Recentemente, Poli e McPhee [89] desenvolveram um prático e elegante método de *auto-ajuste dinâmico* do coeficiente  $c$ , método este empregado na implementação do módulo *learner*. Em vez de focarem no coeficiente como a entidade a ser ajustada, os autores ascenderam ao nível da complexidade das soluções e focaram diretamente no controle do tamanho *médio* dos programas da população. Nessa inversão de foco a definição do coeficiente  $c$  passou a ter caráter secundário, pois o seu ajuste passou a equivaler-se ao resultado de um cálculo que mantém sob controle a complexidade média das soluções candidatas. Este cálculo baseia-se na relação entre as aptidões

dos indivíduos e seus tamanhos, e é expresso pela Equação 4.1<sup>16</sup>:

$$c = \begin{cases} \frac{\text{Cov}(\vec{\ell}, \vec{f}) - (\tau - \mu_{\vec{\ell}})\mu_{\vec{f}}}{\text{Var}(\vec{\ell}) - (\tau - \mu_{\vec{\ell}})\mu_{\vec{\ell}}} & \text{se } \mu_{\vec{\ell}} > \tau \\ 0 & \text{caso contrário} \end{cases} \quad (4.1)$$

Na equação,  $\text{Cov}(\vec{\ell}, \vec{f})$  representa a covariância entre o vetor de tamanhos  $\vec{\ell}$  e o de aptidões  $\vec{f}$ ; similarmente,  $\text{Var}(\vec{\ell})$  é a variância do vetor de tamanhos  $\vec{\ell}$ ;  $\tau$  é uma constante que indica o tamanho médio esperado para os indivíduos da população;  $\mu_{\vec{\ell}}$  e  $\mu_{\vec{f}}$  são, respectivamente, o tamanho e aptidão média dos indivíduos da população corrente.

A Equação 4.1 pode ser interpretada como se segue. Quanto maior o numerador  $\text{Cov}(\vec{\ell}, \vec{f})$ , maior é a correlação entre tamanho e aptidão, dessa forma soluções mais complexas tendem a ser mais frequentemente selecionadas, e portanto aumenta-se a probabilidade dos descendentes imediatos virem a ter tamanho médio maior; isso, no entanto, aciona o crescimento de  $c$  que faz a contra-partida e penaliza com maior vigor os indivíduos grandes. O denominador  $\text{Var}(\vec{\ell})$ , por sua vez, age como um normalizador evitando que  $c$  cresça em demasia. Com função parecida, as demais variáveis agem objetivando controlar a amplitude;  $c$  crescerá tendo como referencial a diferença entre a média dos tamanhos atuais e o valor meta  $\tau$ . Quando o tamanho médio fica abaixo de  $\tau$ , o valor de  $c$  é tornado nulo e assim aumenta-se a tolerância às soluções mais complexas. Dessa forma, a cautela em se fixar o sensível coeficiente  $c$  cede lugar à especificação do limite do *tamanho médio esperado*—o parâmetro  $\tau$ —para as soluções candidatas; basta ao especialista, portanto, ter uma ideia da complexidade da solução procurada e adequar  $\tau$  para tolerar esta complexidade.

Eventualmente  $c$  pode tornar-se negativa quando  $\text{Cov}(\vec{\ell}, \vec{f}) < 0$ , isto é, quando a aptidão de indivíduos grandes decresce ou vice-versa; neste caso  $c$  ingenuamente pressionaria a seleção a favor das soluções mais complexas visando restabelecer o patamar estabelecido, no caso o tamanho  $\tau$ —procedimento de controle rigoroso. O *learner*, porém, torna  $c = 0$  nesses casos no intuito de *sempre* estimular soluções

---

<sup>16</sup>A Equação 4.1 apresenta tão somente uma variante das fórmulas de controle de complexidade propostas pelos autores. Embora as demais variantes encontram-se também implementadas no *learner*, está em uso efetivo apenas o método discutido.

mais simples quando suas aptidões são equivalentes.

#### 4.2.8.1 Considerações Técnicas

Nota-se que a constante  $c$  deve ser recalculada idealmente a *cada* modificação que incida no vetor de aptidões ou de tamanhos, em razão da variância, covariância e médias serem alteradas. Estes vetores são modificados sempre quando se muda a aptidão de um indivíduo ou quando indivíduos são introduzidos ou removidos da população.

Realizar o recálculo completo das médias e (co)variâncias dos vetores seria computacionalmente muito dispendioso<sup>17</sup>, sobretudo na dinâmica co-evolucionária, cuja frequência de alteração das aptidões é alta em decorrência da avaliação incremental (Seção 3.3.2). O *learner* contorna este problema por meio da atualização parcial e gradativa desses valores. Os Algoritmos 4.1 e 4.2, que são baseados no eficiente e numericamente estável método de Welford [90, 91] para cálculo de média e desvio padrão, mostram em pseudo-código como as médias, variância e covariância são atualizadas após terem sido inicialmente computadas.

O Algoritmo 4.1 atualiza a média do tamanho ( $\mu_{\vec{\ell}}$ ) e da aptidão ( $\mu_{\vec{f}}$ ) da população, bem como a variância ( $\text{Var}(\vec{\ell})$ ) e covariância ( $\text{Cov}(\vec{\ell}, \vec{f})$ ); toma como argumentos informações do *indivíduo* sendo atualizado, são elas, o tamanho atual,  $\ell$ , e o novo,  $\ell'$ , a aptidão atual,  $f$ , e a nova,  $f'$ . Simplificado e otimizado, o Algoritmo 4.2 atualiza apenas as variáveis dependentes da aptidão do indivíduo, ou seja, a média das aptidões  $\mu_{\vec{f}}$  e a covariância  $\text{Cov}(\vec{\ell}, \vec{f})$ ; aceita como argumentos, respectivamente, o tamanho do indivíduo ( $\ell$ ), sua aptidão atual ( $f$ ) e a nova ( $f'$ ). A especialização do Algoritmo 4.2 é motivada pela intensa sequência de atualizações gradativas ocorridas no cerne do processo co-evolucionário, onde apenas modifica-se a aptidão do indivíduo.

Nos algoritmos, a variável  $n$  representa o número de elementos dos vetores  $\vec{\ell}$  e  $\vec{f}$ , isto é, o número de indivíduos da população. As variáveis intermediárias  $var_{bruta}(\vec{\ell})$  e

---

<sup>17</sup>Rotineiramente adota-se o cálculo da variância e covariância em dois passos/laços: primeiro computam-se as médias dos vetores e em seguida a soma da diferença (ou soma dos produtos das diferenças, para a covariância) de cada elemento em relação à média.

$cov_{bruta}(\vec{\ell}, \vec{f})$  referem-se ao estado “bruto” da variância e covariância, logo, sem divisão por  $n$ .

---

**Algoritmo 4.1** Atualização *online* de ambas variância e covariância

---

**Requer**  $n > 0$

**procedimento** ONLINEVARCOV( $\ell, \ell', f, f'$ )

$$\mu'_{\vec{\ell}} \leftarrow \mu_{\vec{\ell}} + \frac{(\ell' - \ell)}{n}$$

$$var_{bruta}(\vec{\ell}) \leftarrow var_{bruta}(\vec{\ell}) - (\ell - \mu'_{\vec{\ell}}) \times (\ell - \mu_{\vec{\ell}}) + (\ell' - \mu'_{\vec{\ell}}) \times (\ell' - \mu_{\vec{\ell}})$$

$$cov_{bruta}(\vec{\ell}, \vec{f}) \leftarrow cov_{bruta}(\vec{\ell}, \vec{f}) - (\ell - \mu'_{\vec{\ell}}) \times (f - \mu_{\vec{f}}) + (\ell' - \mu'_{\vec{\ell}}) \times (f' - \mu_{\vec{f}})$$

$$\mu_{\vec{f}} \leftarrow \mu_{\vec{f}} + \frac{(f' - f)}{n}$$

▷ Atualização da média das aptidões

$$\mu_{\vec{\ell}} \leftarrow \mu'_{\vec{\ell}}$$

▷ Atualização da média dos tamanhos

$$Var(\vec{\ell}) \leftarrow \frac{var_{bruta}(\vec{\ell})}{n}$$

▷ Atualização da variância

$$Cov(\vec{\ell}, \vec{f}) \leftarrow \frac{cov_{bruta}(\vec{\ell}, \vec{f})}{n}$$

▷ Atualização da covariância

**fim procedimento**

---



---

**Algoritmo 4.2** Atualização *online* da covariância

---

**Requer**  $n > 0$

**procedimento** ONLINECOV( $\ell, f, f'$ )

$$cov_{bruta}(\vec{\ell}, \vec{f}) \leftarrow cov_{bruta}(\vec{\ell}, \vec{f}) + (f' - f) \times (\ell - \mu_{\vec{\ell}})$$

$$\mu_{\vec{f}} \leftarrow \mu_{\vec{f}} + \frac{(f' - f)}{n}$$

▷ Atualização da média das aptidões

$$Cov(\vec{\ell}, \vec{f}) \leftarrow \frac{cov_{bruta}(\vec{\ell}, \vec{f})}{n}$$

▷ Atualização da covariância

**fim procedimento**

---

## 4.2.9 Especificação de Parâmetros

O *learner* exibe um poderoso e flexível leque de opções para especificação de parâmetros em tempo de execução. As opções contemplam os parâmetros: (i) acerca do modelo de treinamento—*bagging*, *boosting* ou algoritmo **PGMC**; (ii) de entrada e saída—por exemplo, o arquivo de base de dados de treinamento e o arquivo de saída do comitê; (iii) de configurações gerais relativas ao processo evolucionário—como os critérios de parada, tamanho da população, probabilidades dos operadores genéticos, pressão de seleção, complexidade das árvores, topologia de paralelismo e gramática dos classificadores; e (iv) que tangem elementos específicos à programação genética canônica e co-evolucionária.

# Capítulo 5

## Experimentos

São realizados neste capítulo experimentos computacionais que visam comparar o desempenho do algoritmo **PGMC** em tarefas de classificação de dados perante os clássicos meta-algoritmos *bagging* e *boosting*, no contexto da programação genética.

A implementação computacional de referência está descrita no Capítulo 4, onde junto com os Capítulos 2 e 3 encontram-se também a explicação dos parâmetros de sistema relacionados aos experimentos.

São efetuados dois lotes de experimentos, cada qual em nove bases de dados com diferentes características e complexidades. No lote principal, o algoritmo **PGMC**, na sua concepção original, é diretamente confrontado com as outras duas abordagens consideradas; já no lote secundário opta-se pela versão *particionada* do **PGMC**, que é então comparada com sua forma original.

Este capítulo discursa, nesta ordem, sobre as configurações do parque computacional usado, metodologias adotadas e bases de dados dos experimentos. Os resultados experimentais são apresentados e discutidos em seguida. Ao final são feitas algumas conclusões e considerações.

## 5.1 Ambiente Computacional de Alto Desempenho

Os experimentos foram realizados, em sua grande maioria, nas instalações do NACAD<sup>1</sup>. O *hardware* usado foi um *SGI Altix ICE 8200* com a seguinte configuração: 64 processadores *Intel Xeon de 2.66GHz* com quatro núcleos cada, portanto 256 núcleos ao todo, e 512 GB de memória distribuída.

Parte dos experimentos também ocorreu com o amparo de recursos computacionais oriundos do LNCC<sup>2</sup>, onde na ocasião foram utilizados quatro processadores *Intel de 2.50GHz* com quatro núcleos cada (16 no total) e uma quantidade de 32 GB de memória distribuída.

Em ambos os ambientes foram usados o sistema operacional *GNU/Linux*, compilador *GCC* e implementação MPI fornecida pela biblioteca *Open MPI*.

## 5.2 Metodologia

### 5.2.1 Objetivo

A escolha dos parâmetros metodológicos foi guiada levando-se em conta um objetivo em particular: a comparação *entre os três algoritmos* objetos de estudo deste capítulo; portanto, os experimentos expõem a diferença *relativa* entre eles. Qualquer tentativa de transportar os resultados aqui obtidos e compará-los com os de outros algoritmos é cientificamente *inválida*.

Tentou-se obter tanta isenção de vícios quanto possível na comparação entre os algoritmos: (i) mesma implementação base e parâmetros; (ii) formação idêntica do conjunto de treinamento e teste; e (iii) mesmo limite de avaliações por população.

---

<sup>1</sup>Núcleo de Atendimento em Computação de Alto Desempenho (COPPE/UFRJ) – [www.nacad.ufrj.br](http://www.nacad.ufrj.br)

<sup>2</sup>Laboratório Nacional de Computação Científica – [www.lncc.br](http://www.lncc.br)

## 5.2.2 Avaliação da Qualidade dos Classificadores

Seguindo as conclusões de Kohavi [28] sobre estimativas adequadas para se medir o desempenho de classificadores<sup>3</sup>, elege-se neste trabalho para fins de avaliação/comparação o método de *validação cruzada estratificada* de fator  $k = 10$ . Neste esquema, o conjunto original de amostras é particionado em  $k$  blocos disjuntos de tamanhos preferencialmente iguais. Por  $k$  sessões o algoritmo é *treinado* usando-se  $k - 1$  blocos e *testado* sobre o remanescente—os blocos são alternados a cada sessão. O adjetivo *estratificado* diz respeito à preservação da proporção das classes em cada partição em relação ao conjunto original de amostras.

Ignorou-se nos experimentos qualquer configuração predefinida do conjunto de teste e treinamento que eventualmente poderia ser indicada por algumas das bases de dados. A formação de cada conjunto completo de validação cruzada (todos os  $k$  blocos) é feita a partir da ordenação aleatória de todas as amostras da base de dados; no caso de haver separação predefinida em conjunto de teste e treinamento, as amostras de ambos os conjuntos são antes reunidas.

### Tamanho do comitê

A fim de traçar a curva de evolução do desempenho dos algoritmos, optou-se por medir a precisão dos algoritmos com o tamanho do comitê variando de 1 até 31 membros.<sup>4</sup> Com um único membro, efetivamente não há combinação de classificadores, porém, ele serve como referencial.

O algoritmo **PGMC** foi executado de forma paralela/distribuída, o *bagging* em modo paralelo e o *boosting* iterativamente de maneira sequencial. Tanto o **PGMC** quanto o *bagging* adotaram múltiplas populações, entretanto, no *bagging* não houve comunicação (inter-populacional) devido à própria definição do algoritmo.

---

<sup>3</sup>Classificadores, neste contexto, significam os *classificadores finais* de cada experimento.

<sup>4</sup>Pode soar estranho 31 membros e não 32 (potência de dois,  $2^5$ ), mas isto deve-se ao fato de que um processador é tido como unidade *mestre*, cuja função é distribuir amostras de dados e coletar os resultados.

### 5.2.2.1 Limitações

#### Validade Estatística

Jonathan *et al.* [92] notam que uma base de dados contendo  $n$  amostras pode ser arranjada de

$$\frac{n!}{k! \times \left(\frac{n}{k}\right)!^k}$$

maneiras distintas com  $k$  partições de mesmo tamanho, e que cada um desses arranjos pode produzir diferentes medidas de avaliação. Segundo os autores, uma forma de aliviar o risco de vício nas avaliações seria executar o treinamento em diferentes arranjos—configurações distintas de conjuntos completos de validação cruzada—e calcular a média dos resultados. Em [29] recomenda-se executar o processo de validação cruzada por 10 vezes; isto significa que, supondo  $k = 10$ , cada algoritmo teria de ser executado 100 vezes ( $k \times 10$ ) em cada base de dados.

Este cenário ideal é, no entanto, um tanto quanto irrealístico perante a dimensão das baterias de experimentos realizados nesta tese. Concentrando-se apenas na medida de desempenho do algoritmo **PGMC**, teria-se para um lote de experimentos (nove bases de dados em cada lote) que executá-lo 27900 vezes ( $9 \times 10 \times 10 \times 31 = 27900$ ). Este expressivo número é, em grande parte, decorrente do fato de que a dinâmica evolucionária é afetada pelo número de populações (“ilhas”); por isso, uma execução com 31 populações não serviria, estritamente falando, para medir o desempenho de execuções com populações menores se fossem descartadas as soluções das populações excedentes.<sup>5</sup> Não obstante, essa demanda computacional é multiplicada se forem considerados o lote secundário e as avaliações dos demais algoritmos.

No entanto, a própria curva de evolução do desempenho é capaz de oferecer um mecanismo de amenização da variabilidade das avaliações. Esta propriedade decorre da tomada independente das medições ao longo das 31 configurações de comitê. Muito embora pontualmente possam surgir perturbações, o feitio da curva, como um todo, revela uma visão suavizada do desempenho geral.

---

<sup>5</sup>Por exemplo, uma execução com 5 populações não pode ter sua qualidade avaliada aproveitando-se uma execução com 31 populações e desconsiderando-se as 26 ( $31 - 5$ ) populações excedentes.

## Tamanho do Comitê

Devido à limitação computacional, não se pôde verificar o desempenho dos algoritmos para tamanhos de comitê além de 31 classificadores. Isto significa que qualquer extrapolação da curva de desempenho para quantidades maiores é mera especulação, e portanto não há garantias reais de que as análises e conclusões realizadas neste trabalho se mantenham para comitês acima de 31 membros.

## 5.3 Parâmetros

Com exceção do número de avaliações, que foi automaticamente computado em função da base de dados em questão, os demais parâmetros *permaneceram constantes* durante todos os experimentos, isto é, independiam do problema. Esta decisão, por outro lado, tem impacto significativo no que tange ao desempenho *absoluto* dos algoritmos, contudo, tal indicador não é relevante frente os objetivos traçados neste capítulo.

Os parâmetros foram estabelecidos com base em recomendações encontradas na literatura, experiências prévias do autor, e no bom senso; não houve nenhuma tentativa sistemática de otimizá-los.

Em particular, a escolha da quantidade total de avaliações deu-se com base no *compromisso* entre a *capacidade de discernimento* e a *disponibilidade de tempo computacional*, em outras palavras, um limite menor de avaliações poderia não produzir discriminante e um limite maior poderia inviabilizar a execução da bateria de experimentos dentro do tempo disponível. Concomitantemente, buscou-se ponderar a alocação dos recursos computacionais de acordo com a aparente “complexidade” do problema; quanto maior a quantidade de amostras de uma base de dados, maior o limite de avaliações. Precisamente, sendo  $t$  e  $p$  o tamanho do conjunto de treinamento e da população, respectivamente, o número de avaliações toleradas para cada população foi definido como:

$$\text{avaliações} = t \times p \times 50,$$

onde 50 é o “coeficiente de compromisso”. Entende-se por *uma* avaliação de um

classificador a sua execução sobre *uma* amostra de dados, isto é, uma predição equivale à uma avaliação. A Tabela 5.1 lista os demais parâmetros (fixos) e seus valores.

Parâmetro	Valor
Parâmetros gerais	
Tamanho médio esperado ( $\tau$ )	100 nós
Tamanho de cada população ( $I$ )	1000 indivíduos
Tamanho do torneio (pressão de seleção)	5 indivíduos
Profundidade máx. das árvores iniciais	10
Profundidade máx. das árvores de mutação	8
Probabilidade de cruzamento	95%
Probabilidade de mutação	5%
Parâmetros específicos ao <b>PGMC</b>	
Topologia de conexão	<i>fila</i> direcionada
Taxa de migração	0.01%, um indivíduo em 10 gerações
Pressão de seleção das amostras ( $\beta$ )	1.8
Número de iterações do ciclo ( $\gamma$ )	50
Tamanho do histórico de confrontos ( $\eta$ )	20 avaliações

Tabela 5.1: Parâmetros fixos dos experimentos.

### 5.3.1 Gramática

A gramática utilizada nos experimentos foi a predefinida do tipo *atributo-valor*, cujas regras proíbem comparações entre atributos ou operações matemáticas (vide Seção 4.2.1.3).

## 5.4 Apresentação das Bases de Dados

Foram usadas nos experimentos nove base de dados, sendo que oito delas foram colhidas do repositório de *Machine Learning* da Universidade da Califórnia [93]; a nona base de dados é uma aplicação em sistemas petrolíferos e tem origem no trabalho de Doraisamy *et al.* [94]. Tentou-se mediante estas seleções de bases de dados obter: (i) uma amostragem razoavelmente representativa do universo de problemas de classificação de dados, que fosse suficiente para expor as diferenças entre os algoritmos avaliados; e (ii) coleção de problemas que fosse computacionalmente gerenciável diante da metodologia de experimentos proposta.

As bases de dados não passaram por nenhum pré-processamento, como seleção e transformação de atributos, normalização, discretização, redução do número de classes, e afins. Todos os registros com valores ausentes, no entanto, foram ignorados.

A Tabela 5.2 sintetiza as características das bases de dados, listando o número de registros, quantidade e tipos de atributos, número de classes e sua distribuição. Na tabela os tipos de atributos são, nesta ordem: **I**nteiro, **R**eal, **L**ógico, **N**ominal e **O**rdinal. Uma breve descrição para cada uma das bases de dados é feita a seguir.

Base	Reg.	Atributos					#	Classes <i>distribuição</i>
		<i>I</i>	<i>R</i>	<i>L</i>	<i>N</i>	<i>O</i>		
<i>abalone</i>	4177	0	7	0	1	0	28	
<i>allhypo</i>	2514	1	6	20	2	4	4	
<i>arcene</i>	200	10 <sup>4</sup>	0	0	0	0	2	
<i>geochemical</i>	165	3	10	0	4	0	2	
<i>kr-vs-kp</i>	3196	0	0	33	3	0	2	
<i>optdigits</i>	5620	64	0	0	0	0	10	
<i>segmentation</i>	2310	0	19	0	0	0	7	
<i>splice</i>	3190	0	0	0	60	1	3	
<i>waveform</i>	5000	0	40	0	0	0	3	

Tabela 5.2: Síntese das características das bases de dados.

### 5.4.1 *abalone*

O objetivo desse problema é, com base em algumas medidas de fácil obtenção, construir um preditor que seja uma alternativa à tediosa e demorada tarefa de determinação da idade dos abalones (tipo de molusco gastrópodes) por meios convencionais. Normalmente, a idade desses moluscos é determinada pelo trabalhoso processo manual de perfuração da concha, tingimento da sua camada interna e, finalmente, contagem de anéis pelo microscópio.

A *abalone* é uma base de dados terminantemente difícil, tipicamente impondo aos mais diversos preditores taxas de erro consideravelmente superiores a 50%. Duas características certamente impactam nessa dificuldade: o elevado número de classes (28) e o grande desbalanceamento da distribuição destas classes.

### 5.4.2 *allhypo*

A *allhypo* é uma base de dados que visa a detecção de hipotireoidismo em quatro níveis: *negativo*, *hipotireoidismo primário*, *secundário* e *compensado*. Os casos negativos dominam a base de dados, culminando em uma severa desproporção na distribuição de classes.

Uma característica marcante da *allhypo* é a presença simultânea dos cinco tipos de atributos (inteiro, real, lógico, nominal e ordinal).

### 5.4.3 *arcene*

Embora apresente uma baixa quantidade de registros, a base de dados *arcene* destaca-se pela alta dimensionalidade de seus atributos; são 10.000 atributos inteiros.

O problema é descrito como a tarefa de diagnóstico de câncer servindo-se de dados obtidos via espectrometria de massa. Entretanto, o grande desafio reside na identificação dos atributos que de fato são relevantes ao diagnóstico—a base de dados é propositalmente poluída com milhares de atributos irrelevantes à predição.

#### 5.4.4 *geochemical*

Este problema almeja descobrir se um ponto da superfície, no qual foram recolhidos do solo amostras de concentração de etano e certos outros dados ambientais, encontra-se dentro ou fora dos limites de um reservatório de gás natural da Pensilvânia, EUA [94, 95].

Em função da baixa quantidade de amostras, a base de dados *geochemical* reúne registros de dois períodos de coleta, que ocorreram em novembro de 1994 e julho de 1995.

#### 5.4.5 *kr-vs-kp*

Dada uma certa jogada (estado de tabuleiro) em uma partida de xadrez em caráter *end-game*<sup>6</sup>, o objetivo desse problema é predizer se há possibilidade do jogador em ataque vencer a partida. Mais precisamente, considera-se nesta base de dados uma jogada de *end-game* particular, a *Rei+Torre versus Rei+Peão*<sup>7</sup>, onde o atacante dispõe de duas peças, um rei e uma torre, e o outro jogador de um rei e um peão.

Cada amostra da base de dados representa uma variação da posição do tabuleiro para esse tipo de jogada.

#### 5.4.6 *optdigits*

Este é um problema de reconhecimento de dígitos escritos à mão. A imagem de um dígito, codificada em  $32 \times 32$  *pixels*, é segmentada em blocos não sobrepostos de  $4 \times 4$  *pixels*, onde então o número de *pixels* referentes ao dígito em cada bloco é contado. Este procedimento dá origem a uma matriz de dimensão  $8 \times 8$ , onde cada elemento é um valor inteiro que representa uma quantidade de *pixels*. Os atributos da base de dados *optdigits* são, assim, todos os 64 elementos da matriz, enquanto que as classes representam os dígitos de 0 à 9.

---

<sup>6</sup>O *end-game* se refere ao estágio do jogo de xadrez que antecede o término da partida e no qual existem várias estratégias de ataque e defesa bem estudadas e conhecidas.

<sup>7</sup>*King+Rook versus King+Pawn (KR vs KP)*.

### 5.4.7 *segmentation*

A base de dados *segmentation* diz respeito à detecção de imagens e envolve o reconhecimento de alguns padrões usualmente encontrados na natureza e construções, por exemplo “folhagem”, “cimento”, entre outros. Os registros da base de dados foram criados, aleatoriamente, a partir de sete imagens, cada qual representando uma região segmentada de  $3 \times 3$  *pixels*. Para cada amostra, 19 atributos contínuos fornecem dados sobre propriedades da região de imagem representada, como por exemplo a saturação, densidade, intensidade, *hue* e outros. Uma dada região pode ser enquadrada em sete classes distintas, a saber: *tijolo*, *céu*, *folhagem*, *cimento*, *janela*, *estrada* e *grama*.

### 5.4.8 *splice*

As junções de *splicing* são pontos em uma sequência de DNA que demarcam o limite entre os éxons (partes da sequência de DNA retidas após o *splicing*) e os íntrons (partes da sequência de DNA que são removidas).

O objetivo da base de dados *splice* é, fornecidos elementos de uma sequência de DNA (codificados pelos atributos), identificar o tipo do limite nos pontos de junção, podendo ser *de íntron para éxon* (“íntron  $\rightarrow$  éxon”), *de éxon para íntron* (“éxon  $\rightarrow$  íntron”) ou *nenhum dos dois*.

### 5.4.9 *waveform*

A base de dados *waveform* conjuga atributos ruidosos com atributos irrelevantes. Dentre os atributos, que são todos contínuos, 21 são verdadeiros, porém, foi adicionado a cada um destes ruído gaussiano—com média 0 e variância 1. Os 19 atributos restantes são supérfluos e visam simplesmente confundir o processo de treinamento.

Os atributos relevantes descrevem uma forma de onda, sendo gerados como combinações de outras formas de onda [96, 97]. O problema busca, portanto, a identificação da forma de onda (três variações) com base nas informações contidas nos atributos.

## 5.5 Resultados

Esta seção apresenta e discute os resultados obtidos nos experimentos de classificação de dados sobre as bases de dados consideradas neste trabalho.<sup>8</sup> São divididos em dois lotes de experimentos. O primeiro compara o desempenho do algoritmo proposto, **PGMC**, com os clássicos *bagging* e *boosting*. O segundo lote, de caráter secundário, apresenta a evolução do desempenho do **PGMC** ao longo do aumento gradativo do número de partições do conjunto de treinamento.

A ordem em que os problemas são abordados é dada alfabeticamente de acordo com o nome da base de dados em questão.

A Figura 5.1 exibe a curva da taxa de erro no conjunto de teste para a base de dados *abalone* à medida que o tamanho do comitê cresce. As taxas de erro deste problema são bastante acentuadas, o que se explica pela dificuldade inerente do problema e por não figurar como meta dos atuais experimentos a otimização do desempenho *absoluto*. Apesar dos baixos índices de classificação, percebe-se que o **PGMC** comportou-se nitidamente melhor do que os demais algoritmos, sustentando queda continuada do erro de classificação até aproximadamente 10 classificadores no comitê. O *bagging* mostrou-se estável, enquanto que o *boosting* apresentou um comportamento bastante anômalo nesta base de dados: ao invés de decrescer a taxa de erro à proporção que o número de classificadores aumentava, sua precisão de classificação degradou.

As curvas para o conjunto de treinamento seguem mais ou menos o mesmo padrão, como visto na Figura 5.2. Embora ainda atípico, o comportamento anômalo do *boosting* mostrou-se menos severo.

O experimento na base de dados *allhypo* (Figuras 5.3 e 5.4) mostra claramente a melhor eficiência de treinamento do algoritmo **PGMC**; embora a queda na taxa de erro tenha sido sensível, já de início nota-se uma expressiva disparidade de precisão de classificação a favor do **PGMC**. O *bagging* procedeu-se de maneira visivelmente ruim e foi incapaz de obter qualquer ganho de classificação. O *boosting*, por outro

---

<sup>8</sup>Encontra-se no Apêndice A a sintetização dos valores das taxas de erro de todos os experimentos oficiais realizados nesta tese.

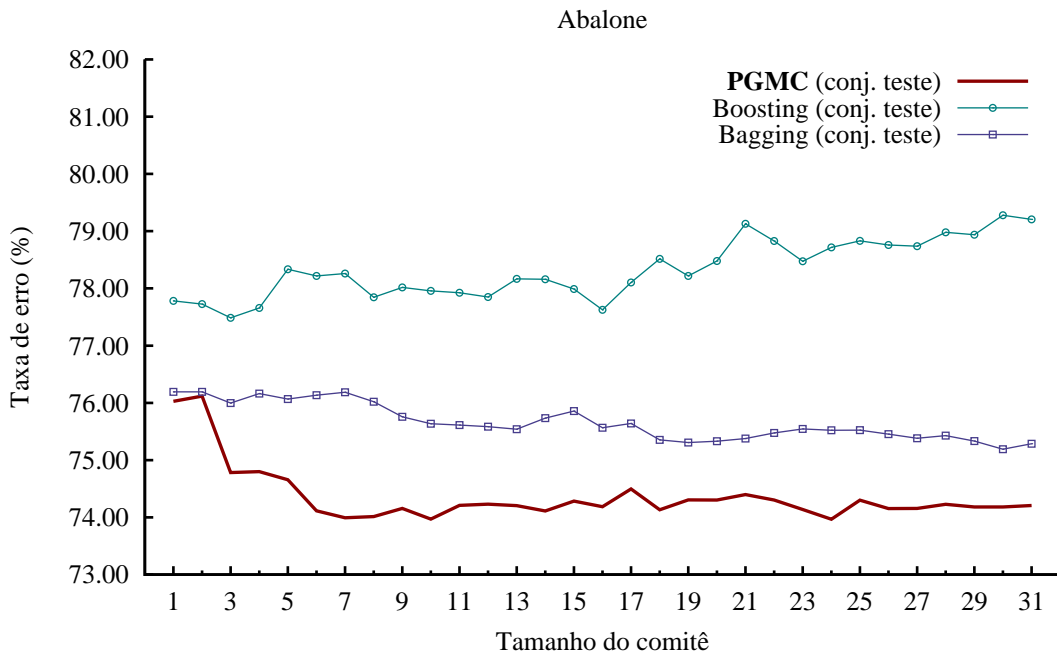


Figura 5.1: *Abalone*: curva de erro no conjunto de teste.

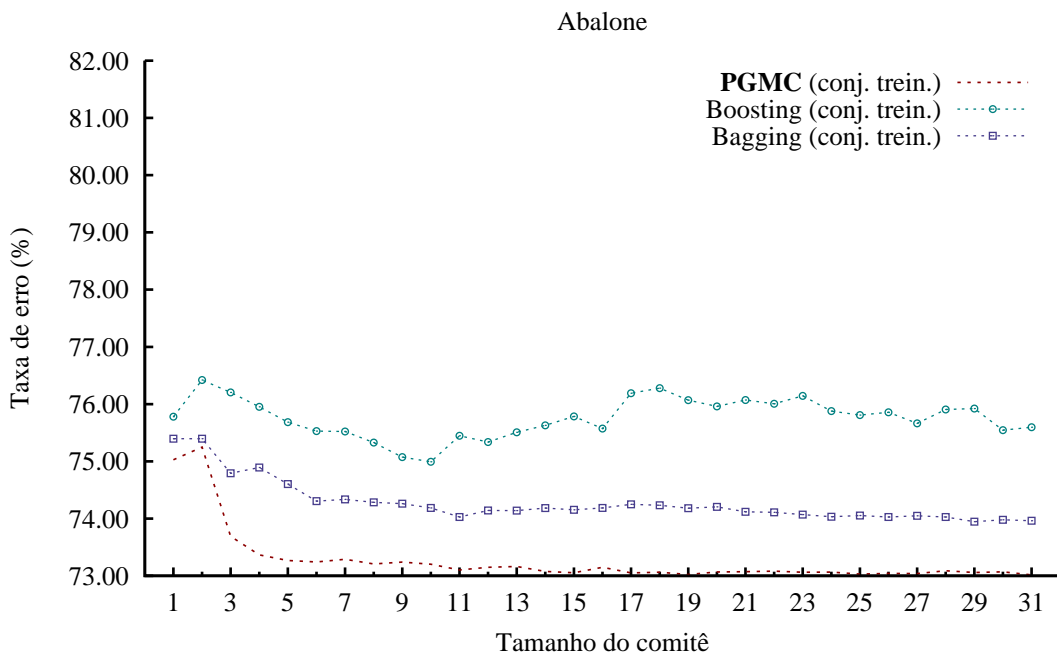


Figura 5.2: *Abalone*: curva de erro no conjunto de treinamento.

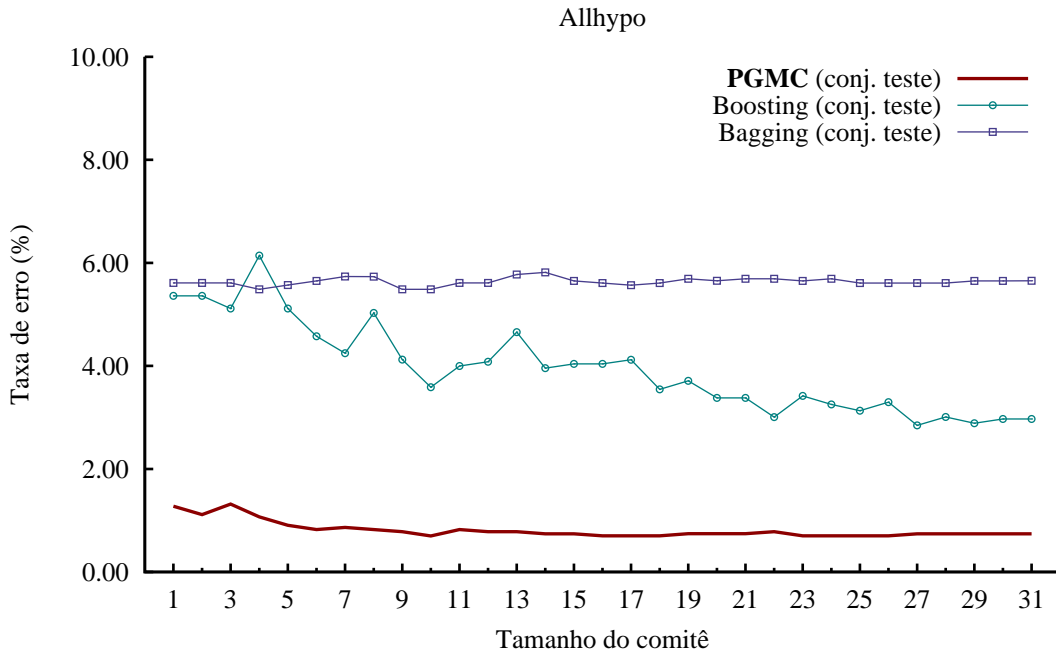


Figura 5.3: *Allhypo*: curva de erro no conjunto de teste.

lado, apresentou-se como tipicamente o faz, obtendo uma significativa queda da taxa de erro conforme o número de classificadores aumentava.

No que tange a base de dados *arcene*, todos os algoritmos foram capazes de apresentar uma tendência esperada de redução de erro, como mostra a Figura 5.5. O **PGMC**, no entanto, exibiu uma evolução no poder de classificação bem mais pronunciada do que os outros algoritmos. O *boosting* veio em segundo lugar, ao passo que o *bagging* apresentou uma curva de decréscimo da taxa de erro bem tímida perante os demais. Os distintos pontos de pico/perturbação visíveis na curva do gráfico podem na verdade ser ruídos decorrentes das limitações metodológicas discutidas na Seção 5.2.2.1; entretanto, não impedem a correta interpretação da tendência da curva.

A avaliação em relação ao conjunto de teste da base de dados *arcene*, visto na Figura 5.6, expõe uma propriedade bastante controversa do *boosting*: a intensidade com que foca as amostras de treinamento. Com apenas sete membros no comitê o *boosting* obteve 100% de precisão de classificação no conjunto de treinamento. Entretanto, tamanha exuberância não contribuiu em nada no real quesito, que é o desempenho no conjunto de teste; aliás, não raramente, a super eficiência no conjunto de treinamento traz sérios efeitos colaterais na capacidade de generalização

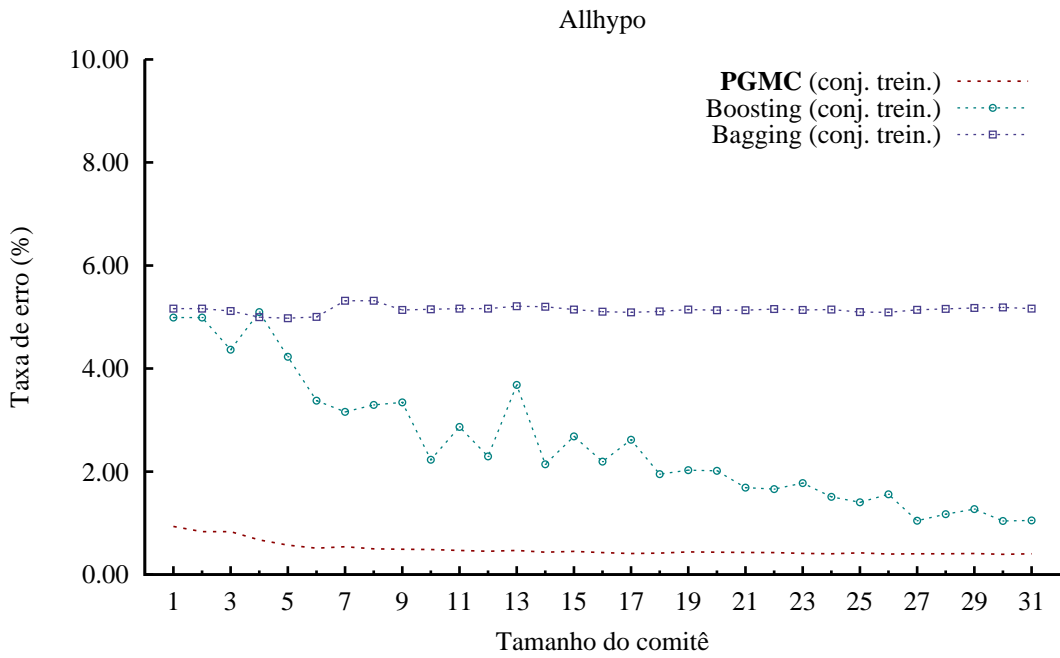


Figura 5.4: *Allhypo*: curva de erro no conjunto de treinamento.

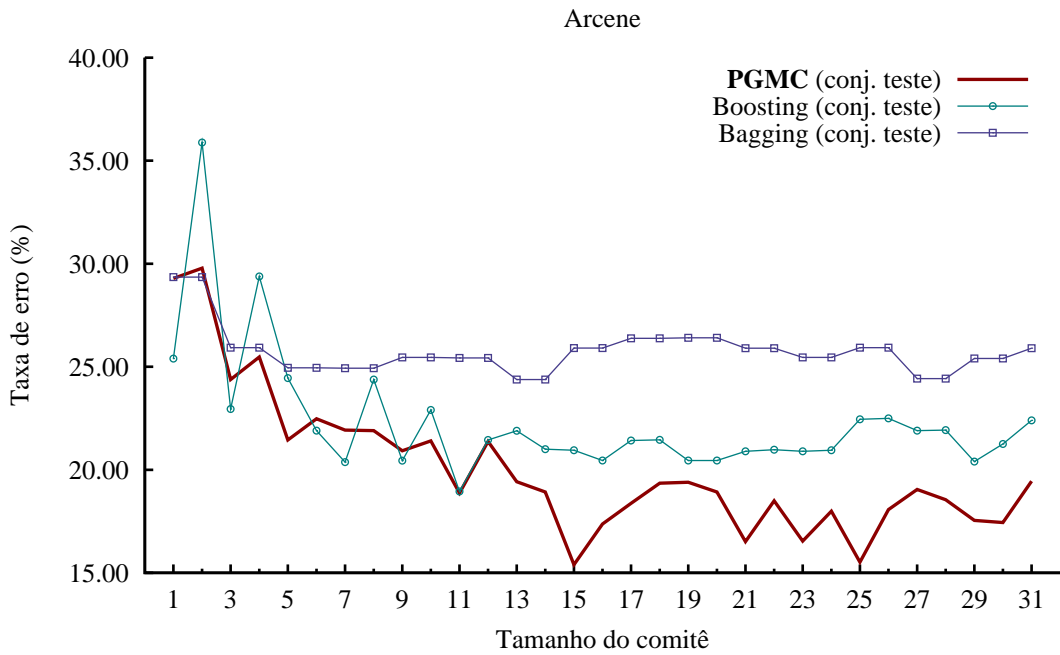


Figura 5.5: *Arcene*: curva de erro no conjunto de teste.

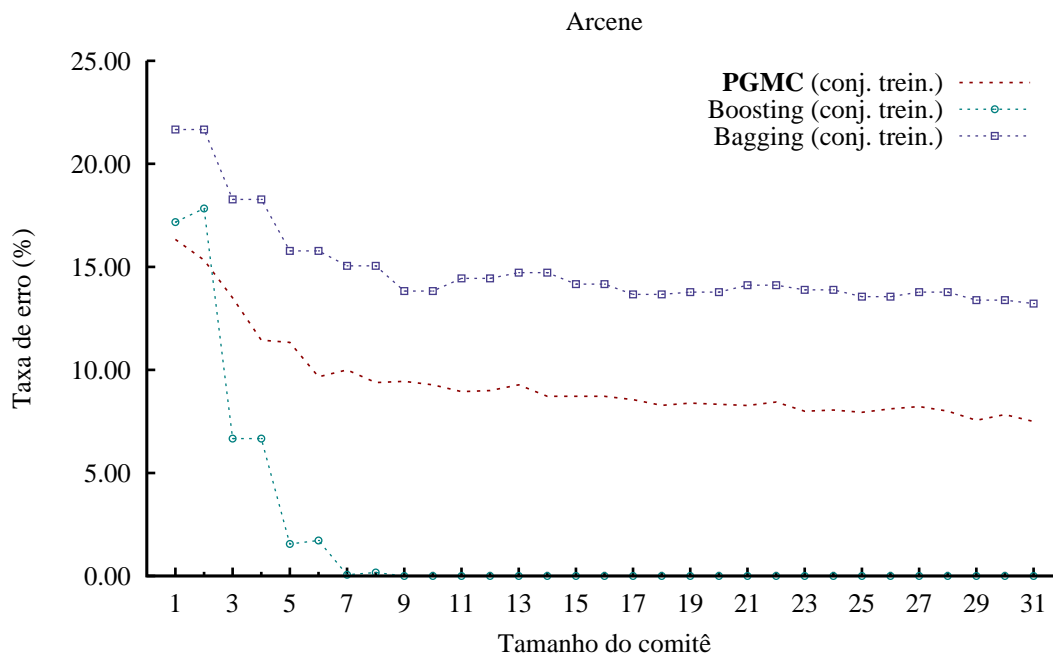


Figura 5.6: *Arcene*: curva de erro no conjunto de treinamento.

(conjunto de teste).

O **PGMC** foi o único algoritmo a ficar abaixo do índice de 2% de erro de teste na base de dados *geochemical* (Figura 5.7), que foi alcançado ao passar da marca de oito membros no comitê. O *boosting* teve a curva estagnada a partir do quinto classificador. Por sua vez, o *bagging* obteve uma expressiva queda logo de início, mas apresentou instabilidades ao longo da formação do comitê.

A Figura 5.8 mostra, como já era de se esperar, que no conjunto de treinamento o **PGMC** obteve índices ainda mais precisos. O *boosting* apresentou o mesmo padrão de desempenho do conjunto de teste; esta estagnação, no entanto, não é um comportamento esperado, e não ficou aparente o motivo. Muito embora o *bagging* tenha mostrado instabilidades no conjunto de teste, sua atuação foi menos grave no conjunto de treinamento; não há indícios seguros de perda de generalização pois ambas as curvas exibiram aproximadamente a mesma evolução, isto é, não divergiram.

No problema *kr-vs-kp*, o *boosting* demonstrou um exímio desempenho, mostrado na Figura 5.9. A atuação do **PGMC** manteve-se superior até o quarto classificador, a partir de então não pôde acompanhar a curva do *boosting*. Por outro lado, ficou

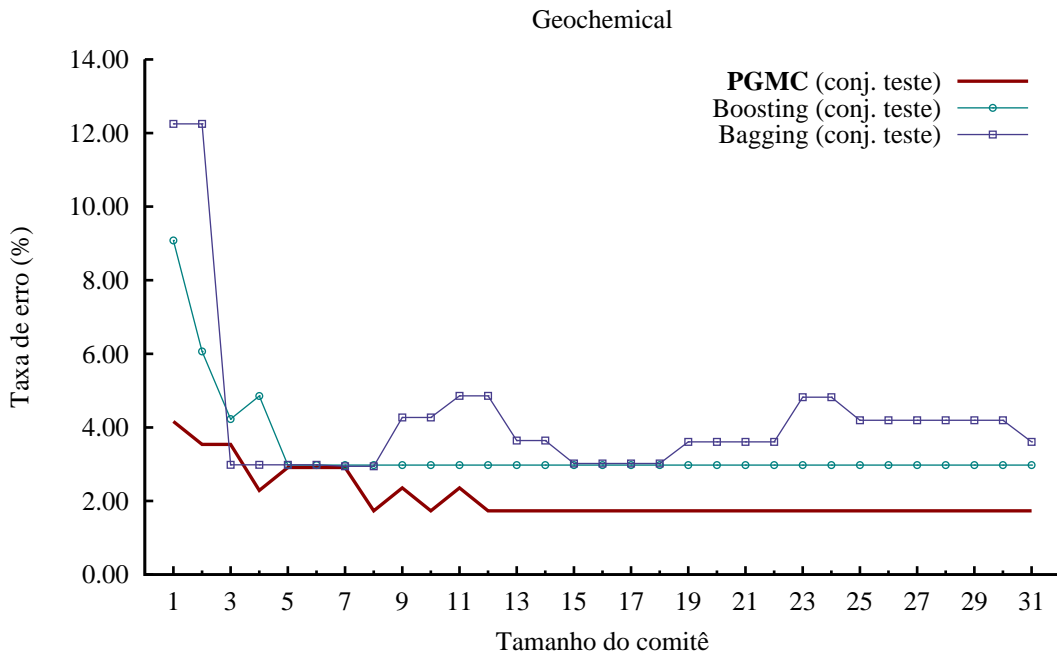


Figura 5.7: *Geochemical*: curva de erro no conjunto de teste.

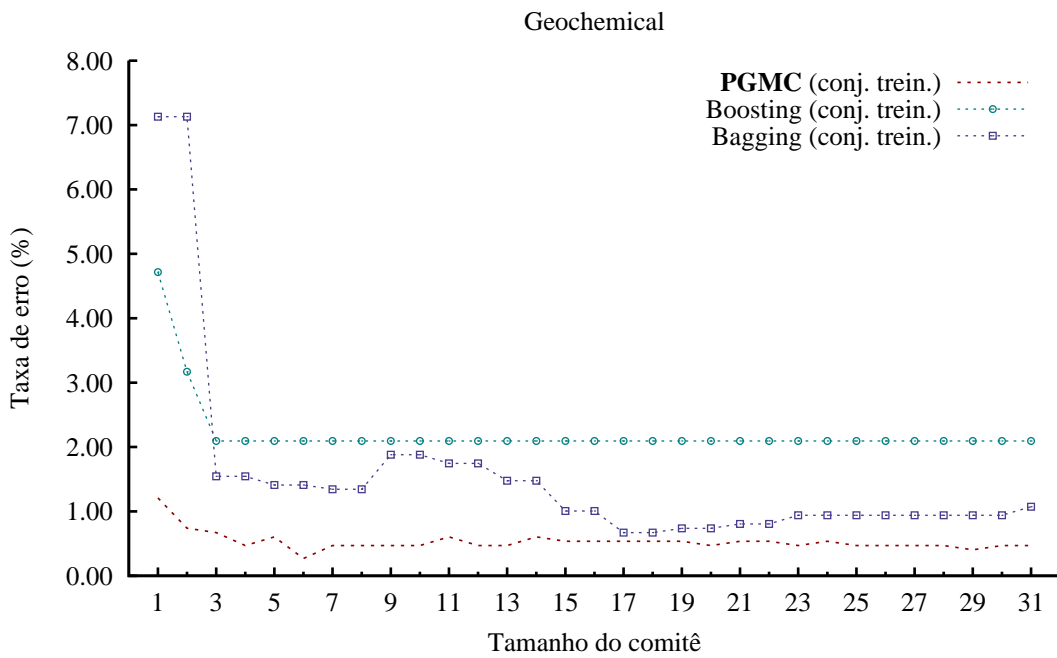


Figura 5.8: *Geochemical*: curva de erro no conjunto de treinamento.

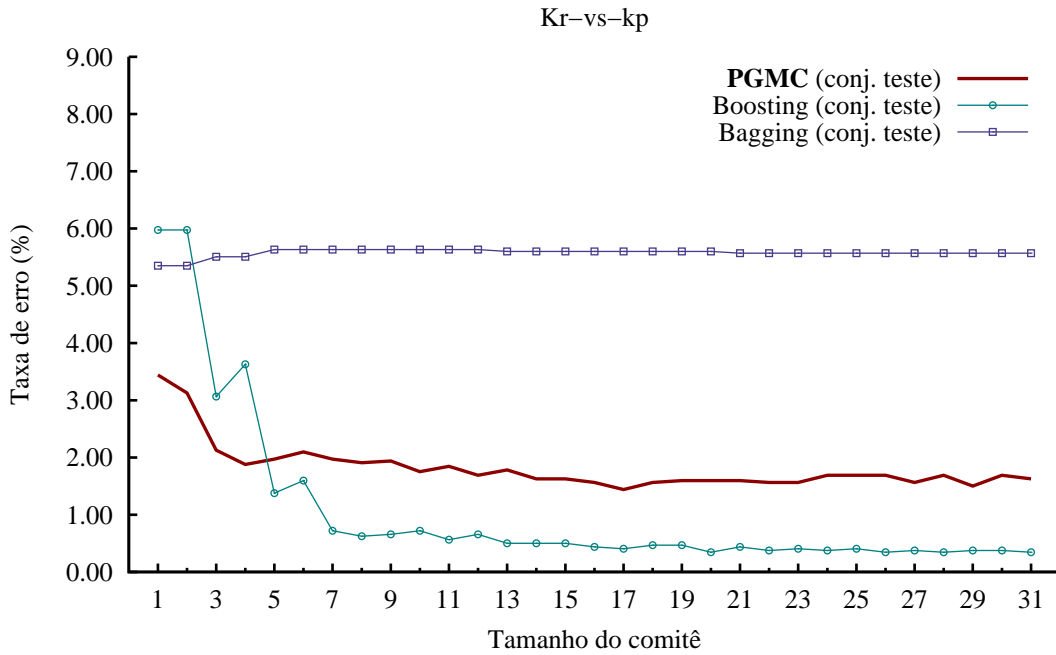


Figura 5.9: *Kr-vs-kp*: curva de erro no conjunto de teste.

evidente a supremacia do desempenho do **PGMC** em relação ao *bagging*, este que foi incapaz de explorar qualquer benefício da coleção de classificadores.

Quanto ao desempenho no conjunto de treinamento (Figura 5.10), o feitiço das curvas permaneceu o mesmo. Cabe notar que, mesmo atingindo 100% de classificação, o *boosting* ainda pôde apresentar sensíveis melhoras na generalização (conjunto de teste).

O **PGMC** apresentou um bom resultado na base de dados *optdigits*, como exposto na Figura 5.11. No entanto, a partir do 19º classificador o *boosting* foi mais agressivo e obteve melhor decaimento na taxa de erro. O *bagging*, mais uma vez, foi o algoritmo menos competitivo entre os três. A curva sobre o conjunto de treinamento resultou em um padrão muito semelhante (Figura 5.12).

A eficiência de treinamento do **PGMC** novamente fez-se presente de forma categórica na base de dados *segmentation*, visto na Figura 5.13. A maior margem para os outros algoritmos deu-se com um comitê reduzido: com oito classificadores o **PGMC** alcançou um índice que o *boosting* só atingiria com vinte classificadores. Enquanto o *boosting*, mesmo exigindo comitês maiores, conseguiu em certo momento se igualar ao **PGMC**, o *bagging* apresentou um afastamento virtualmente constante.

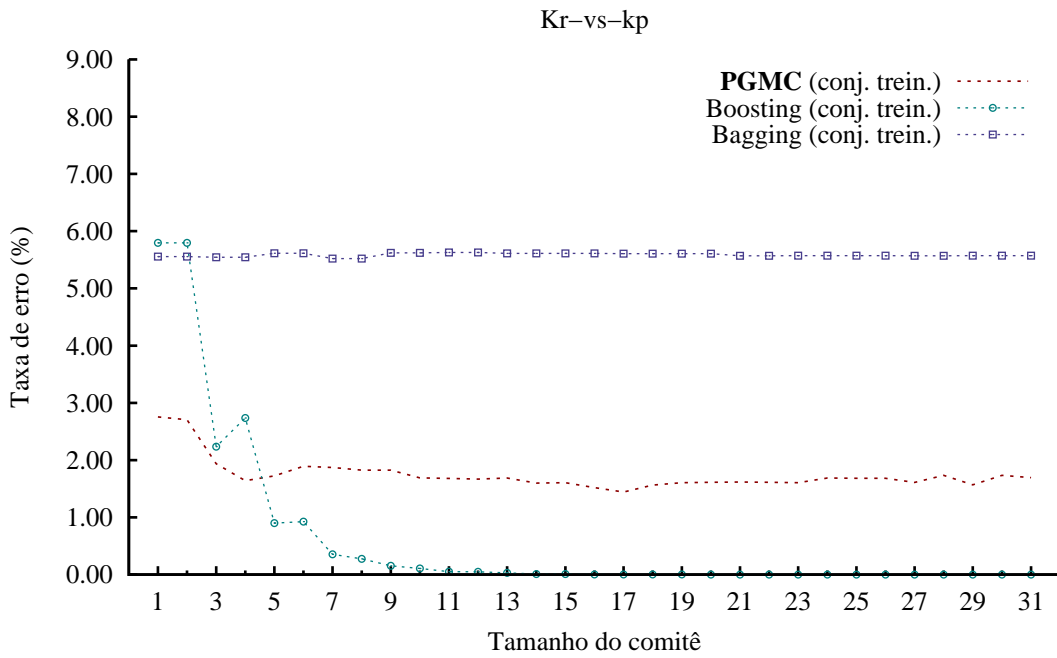


Figura 5.10: *Kr-vs-kp*: curva de erro no conjunto de treinamento.

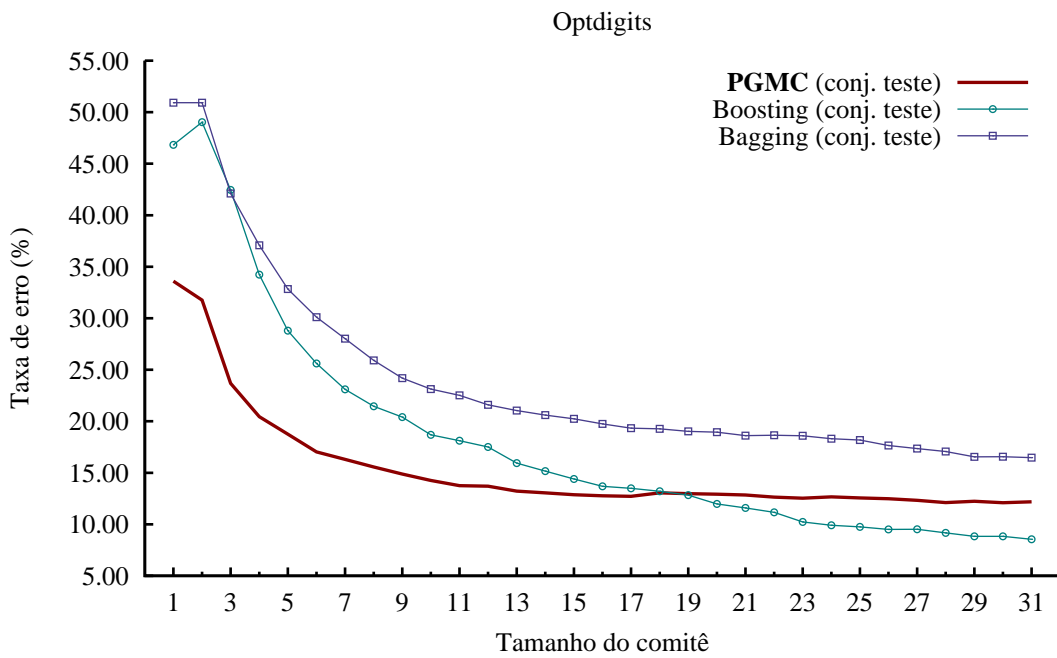


Figura 5.11: *Optdigits*: curva de erro no conjunto de teste.

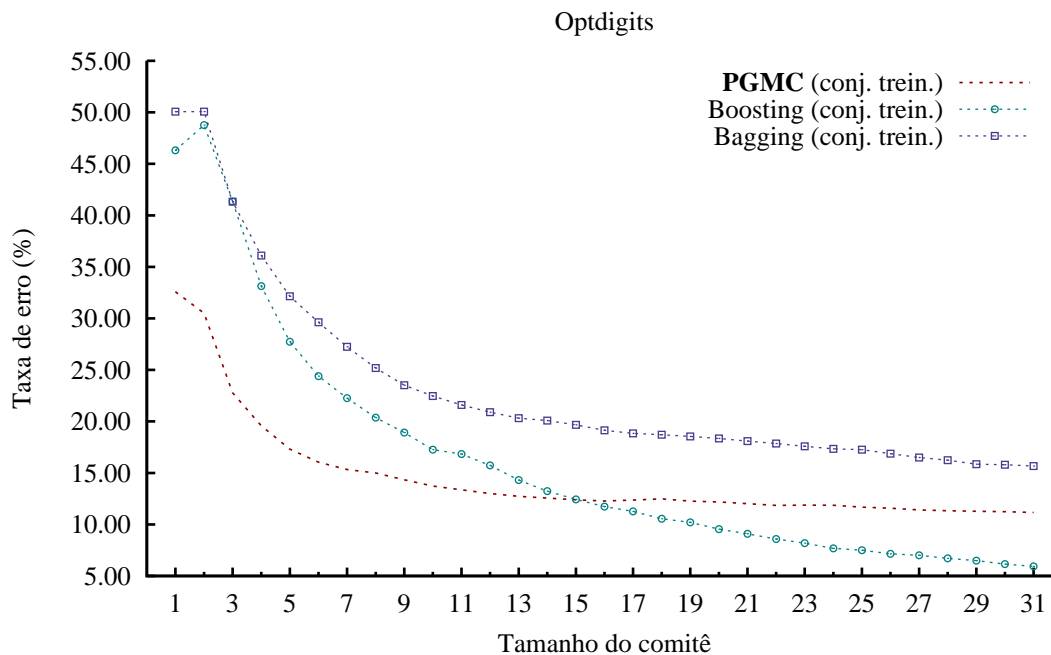


Figura 5.12: *Optdigits*: curva de erro no conjunto de treinamento.

Os desempenhos no conjunto de treinamento não apresentaram nenhuma surpresa (Figura 5.14). Nota-se, contudo, que o *boosting* obteve uma pequena margem de vantagem a partir do vigésimo classificador; margem esta que não teve efeitos, proporcionalmente, no conjunto de teste.

Percebe-se pelas Figuras 5.15 e 5.16 que, dentre todas as outras, a base de dados *splice* foi a que ofereceu menor resistência à classificação. Nota-se ainda neste problema que, mesmo com uma taxa de erro no conjunto de treinamento ligeiramente superior a do *boosting*, o **PGMC** sustentou menor taxa de erro no conjunto de teste. O *bagging* foi o único algoritmo que não conseguiu obter 100% de precisão no conjunto de treinamento dentro do limite de 31 classificadores.

Finalmente, observando as curvas das taxas de erro no conjunto de treinamento da base de dados *waveform*, Figura 5.18, é possível perceber que o *boosting* começa a esboçar uma tendência à perda de generalização—ou melhor à estagnação— a partir do 15º classificador, onde o *bagging* passa à sua frente (Figura 5.17). Neste estágio o *bagging* e, em menor escala o **PGMC**, demonstram melhor robustez. Não coincidentemente, a *waveform* é uma base de dados declaradamente ruidosa. De

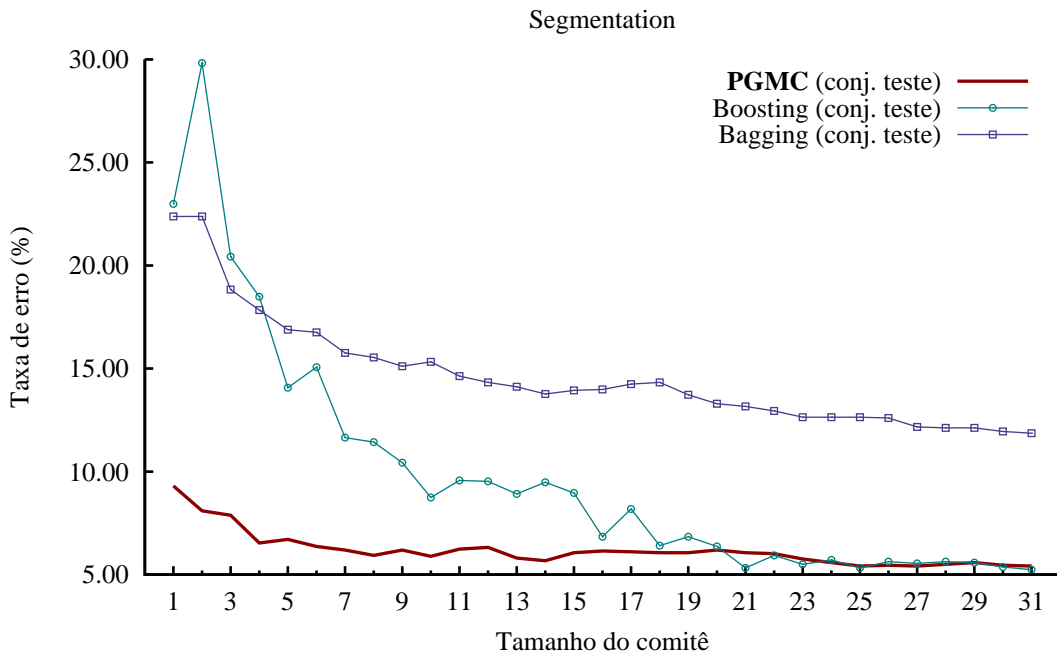


Figura 5.13: *Segmentation*: curva de erro no conjunto de teste.

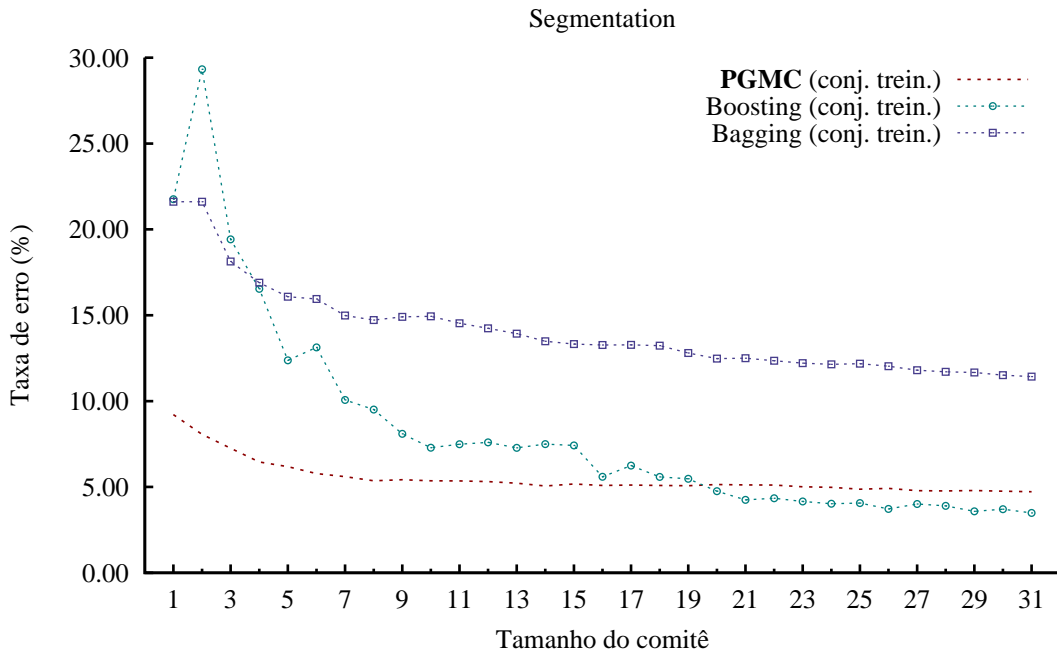


Figura 5.14: *Segmentation*: curva de erro no conjunto de treinamento.

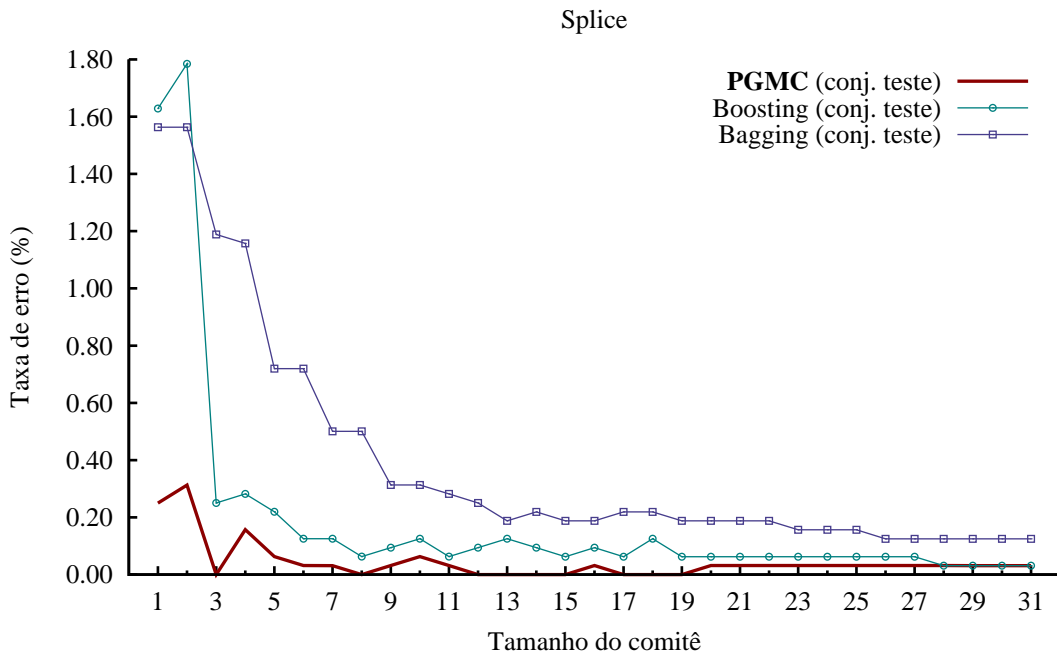


Figura 5.15: *Splice*: curva de erro no conjunto de teste.

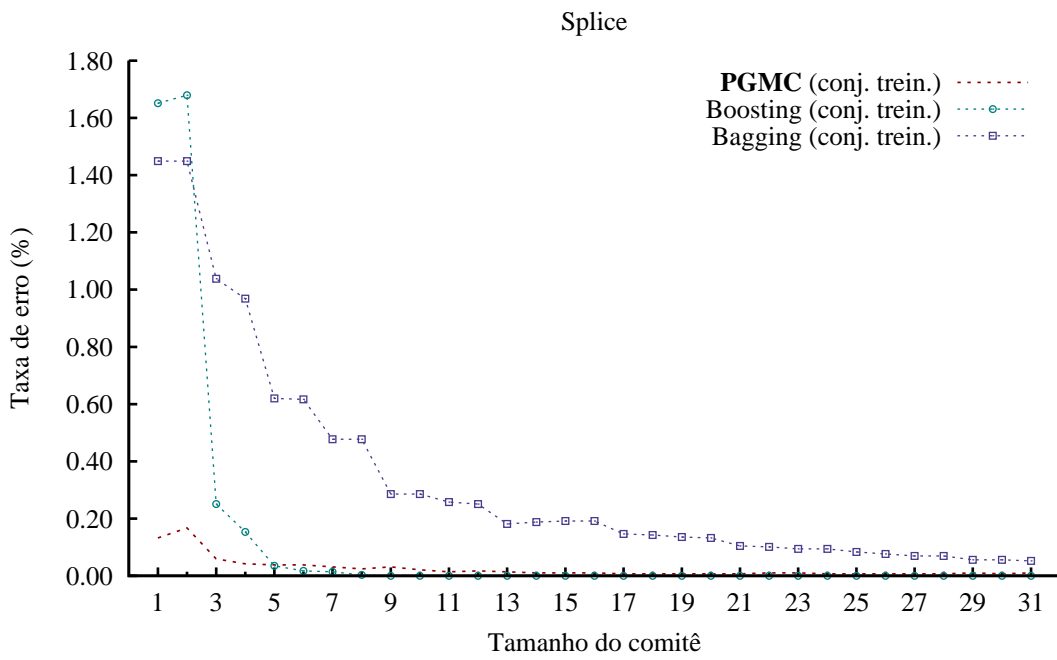


Figura 5.16: *Splice*: curva de erro no conjunto de treinamento.

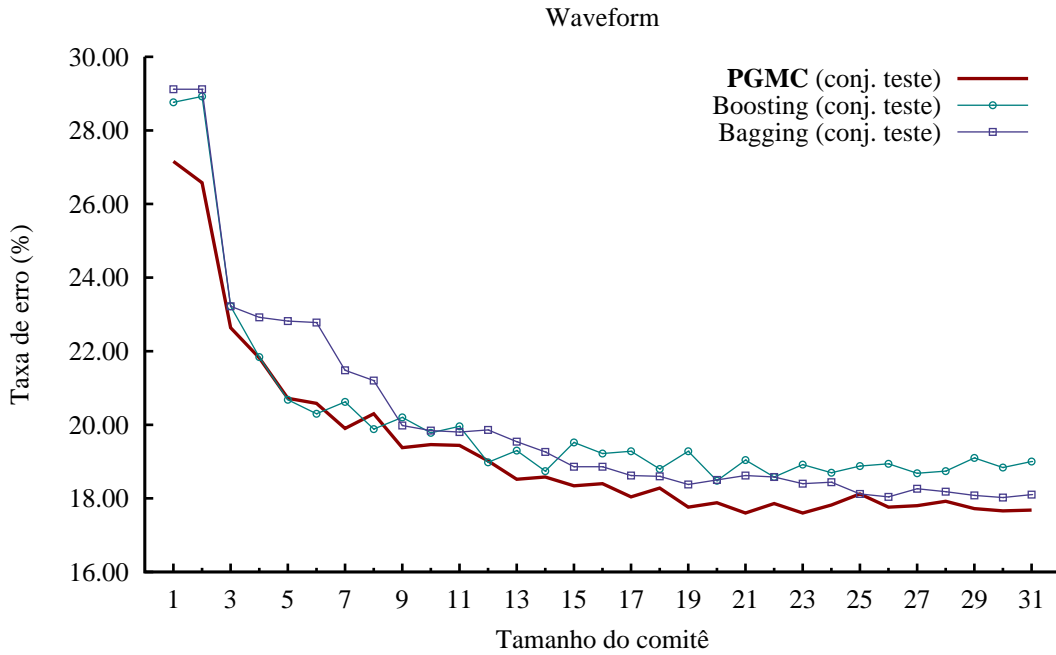


Figura 5.17: *Waveform*: curva de erro no conjunto de teste.

qualquer forma, o **PGMC** exibe, por todo o conjunto de teste desse problema, o melhor desempenho entre os algoritmos.

### 5.5.1 Análise e Discussão

Para que o uso do algoritmo **PGMC** se justifique, primeiramente é *necessário* que ele apresente *melhor desempenho* do que o *boosting*, do contrário seria obviamente preferível utilizar o próprio *boosting*, pois este, entre outras propriedades, também é trivialmente paralelizável. Os experimentos mostraram que em todos os problemas estudados, sem exceção, o **PGMC** comportou-se melhor do que o *boosting*, não raramente por uma boa margem de diferença. Portanto, o **PGMC** satisfaz este requisito fundamental, o que já garante a legitimidade de sua utilidade.

Em segundo lugar, seria terminantemente *desejável* que o algoritmo **PGMC** exibisse desempenho comparável ao do *boosting* no que concerne aos índices de classificação. Ser competitivo ao *boosting* nesses termos poria o **PGMC** na condição de uma alternativa absoluta a este: gozaria da nítida vantagem do paralelismo ao mesmo tempo em que produziria resultados tão bons quanto. Os experimentos confirmaram que não somente o **PGMC** é competitivo, mas superior se for considerado

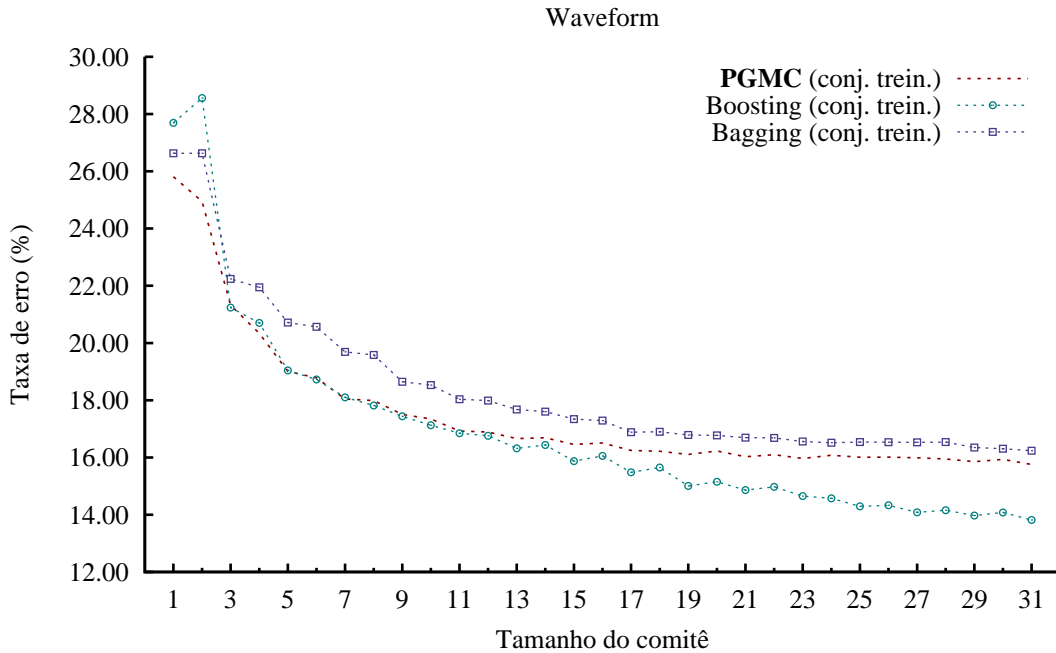


Figura 5.18: *Waveform*: curva de erro no conjunto de treinamento.

o desempenho geral. Com exceção das bases de dados *kr-vs-kp* e *optdigits*, o **PGMC** comportou-se melhor em todas as outras sete bases de dados. Mesmo na *kr-vs-kp* e *optdigits* o **PGMC** exibiu vantagem parcial, sendo melhor do que o *boosting* até certo momento.

## 5.5.2 Particionamento do Conjunto de Treinamento

Esta sessão de experimentos visa estabelecer uma noção acerca dos limites da aplicabilidade do algoritmo **PGMC** quando o conjunto de treinamento é sucessivamente particionado. Esta modalidade de experimentação é motivada pelo extraordinário benefício em termos computacionais quando o esforço é efetivamente dividido entre as populações/ilhas; em outras palavras, nesse modelo, quanto maior o tamanho do comitê (“número de partições”), mais rapidamente se dá o treinamento.

Intuitivamente, espera-se que a taxa de erro cresça monotonicamente à medida que o número de partições aumenta, pois progressivamente o processo evolucionário tem ao seu dispor menor quantidade de informação sobre a base de dados—há tendência ao excesso de ajuste (*overfitting*) localmente em cada população.

São exibidos nos gráficos de curva de erro a comparação entre duas variações de

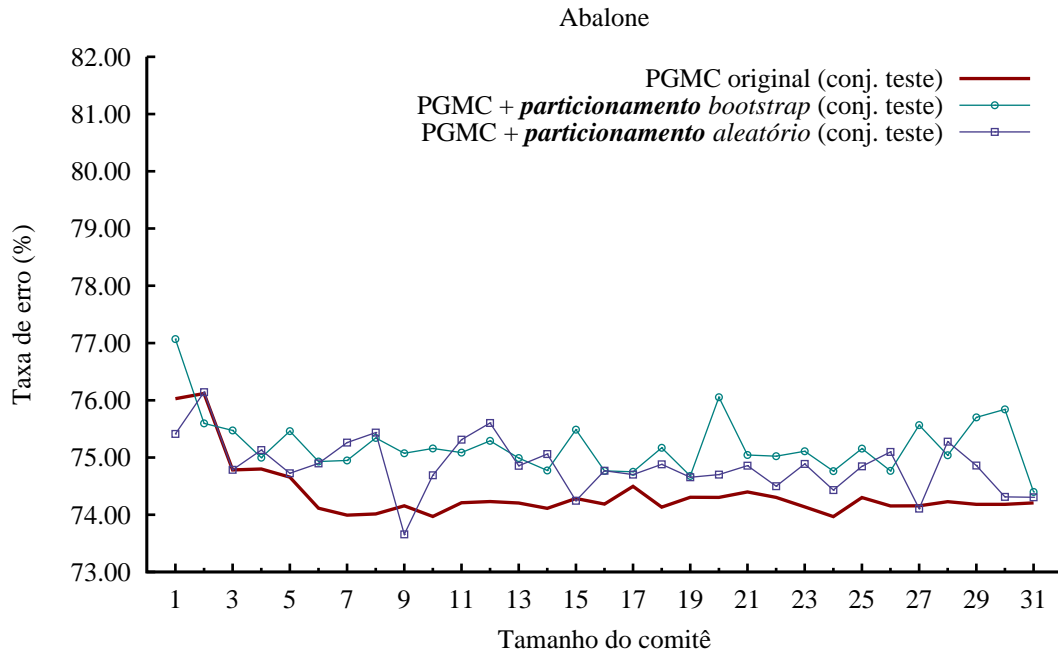


Figura 5.19: *Abalone*: curva de erro no conjunto de teste (versão particionada).

particionamento e a versão original do **PGMC**, como avaliada na seção anterior. As duas variações diferem no que concerne ao mecanismo de amostragem; enquanto uma versão usa amostragem via *bootstrap* (com reposição) a outra usa amostragem *aleatória sem reposição*, isto é, uma certa amostra de dados deve obrigatoriamente ser alocada à uma população uma única vez.

Por apresentarem desempenho bem similar ao do conjunto de teste, não foram incluídos os gráficos relativos ao conjunto de treinamento nesta bateria de experimentos.

Na Figura 5.19 tem-se a evolução da taxa de erro da base de dados *abalone* conforme o tamanho do comitê aumenta. Contrariando a intuição, ambas as versões com particionamento não somente evitaram a degradação como, ainda que sutilmente, apresentaram uma tendência de melhora na precisão de classificação que se sustentou por toda a curva. Não é nítida a diferença entre os modelos de amostragem, mas aparentemente o *aleatório* saiu-se sensivelmente melhor. Curiosamente, estes resultados foram superiores aos do *bagging* e *boosting* dos experimentos anteriores.

O relativo sucesso do experimento não se repetiu no caso da base de dados *allhypo*, como pode ser visto na Figura 5.20. A versão particionada rapidamente se

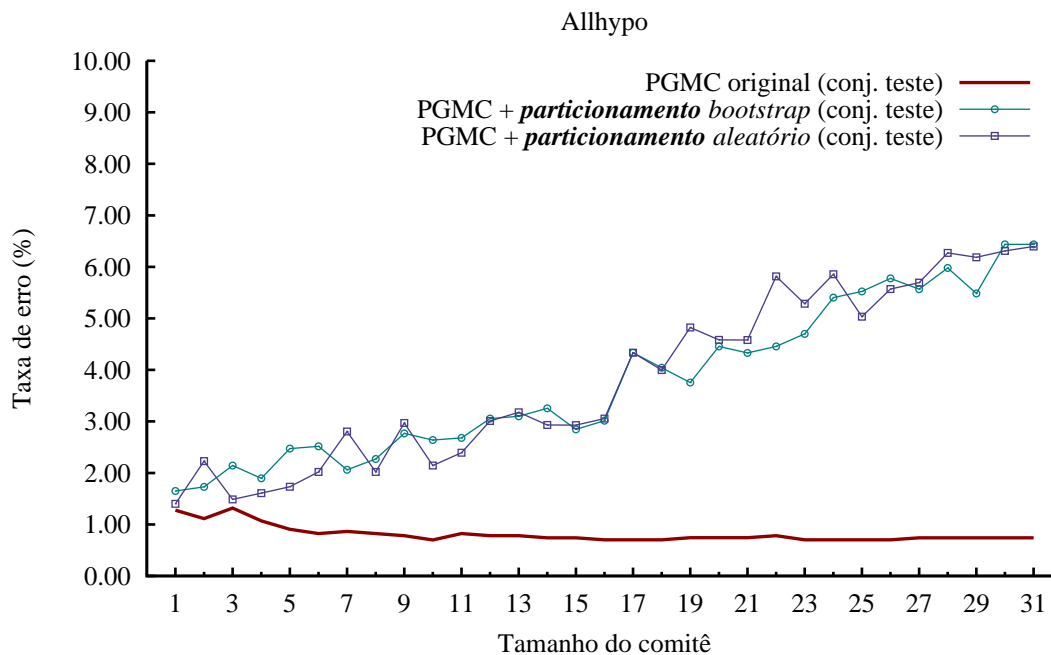


Figura 5.20: *Allhypo*: curva de erro no conjunto de teste (versão particionada).

degradou.

Quanto à base de dados *arcene*, embora em relação ao **PGMC** original o desempenho pareça pífio, percebe-se que os algoritmos com particionamento tiveram aptidão para sustentar a taxa de erro (Figura 5.21).

Na base de dados *geochemical*, cujos desempenhos são mostrados na Figura 5.22, as versões com particionamento exibiram um sucesso parcial, sendo dominantes até três partições. Os resultados são razoáveis, sobretudo se for observado que a *geochemical* é a base de dados de todos os experimentos com menor número de registros.

O desempenho dos algoritmos com particionamento na base de dados *kr-vs-kp* apresenta o mesmo perfil do problema anterior, mas é mais bem-sucedido, sendo competitivo até o 9º particionamento (Figura 5.23).

Os resultados na base de dados *optdigits* (Figura 5.24) são, no entanto, surpreendentes. Os algoritmos com particionamento demonstraram que podem, a despeito do menor número de amostras para treinamento, aumentar continuamente o poder de generalização. Embora tenham exibido um feitio de decaimento mais próximo do linear—enquanto que o do **PGMC** original tem aspecto exponencial—as taxas

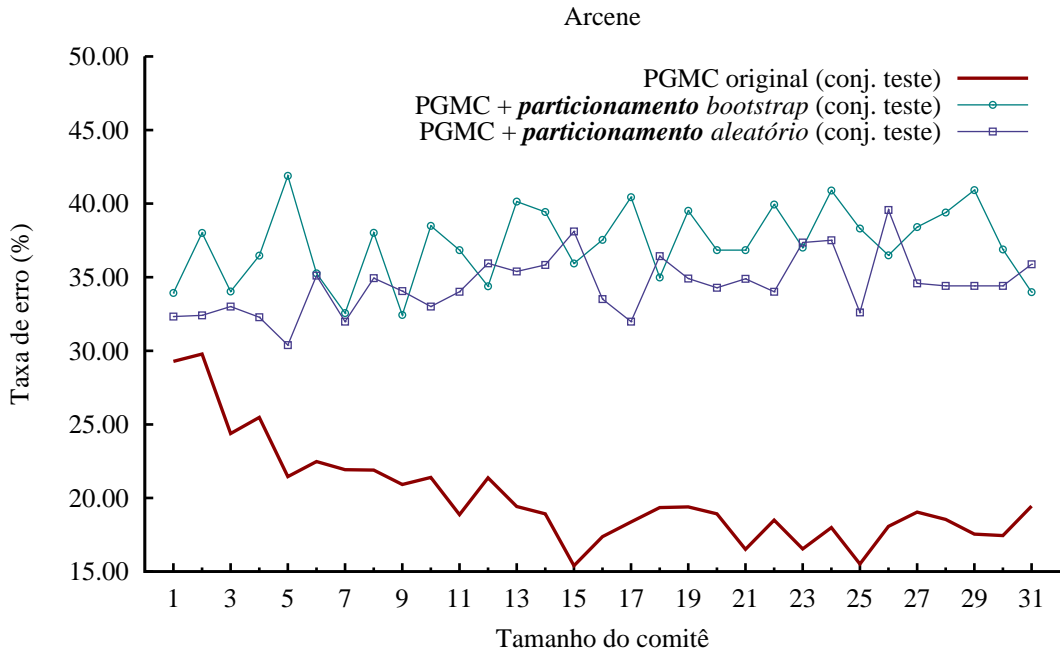


Figura 5.21: *Arcene*: curva de erro no conjunto de teste (versão particionada).

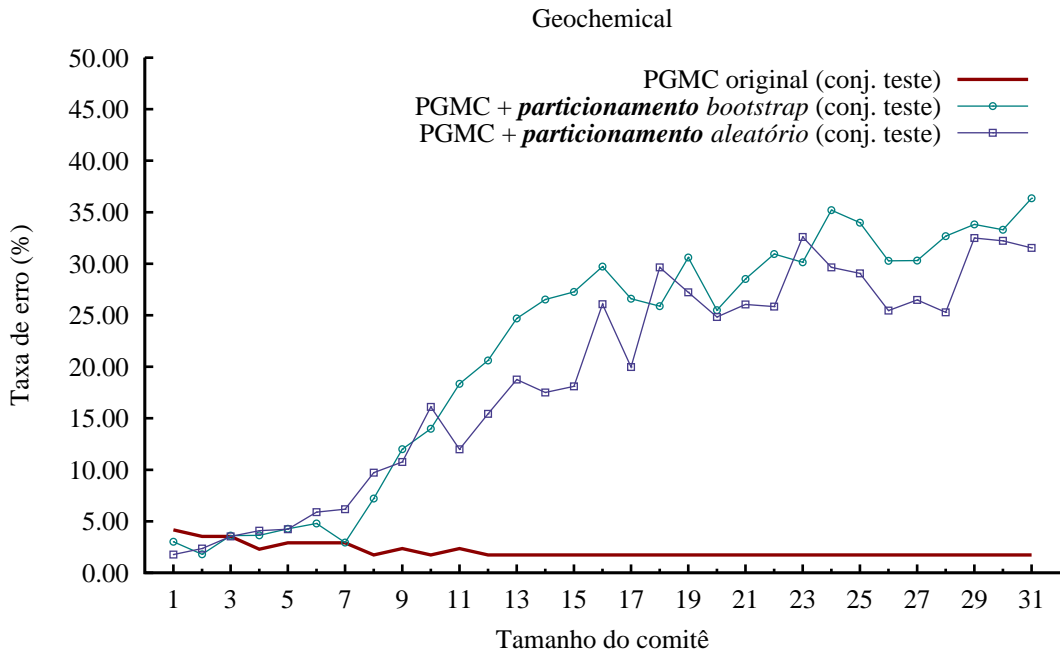


Figura 5.22: *Geochemical*: curva de erro no conjunto de teste (versão particionada).

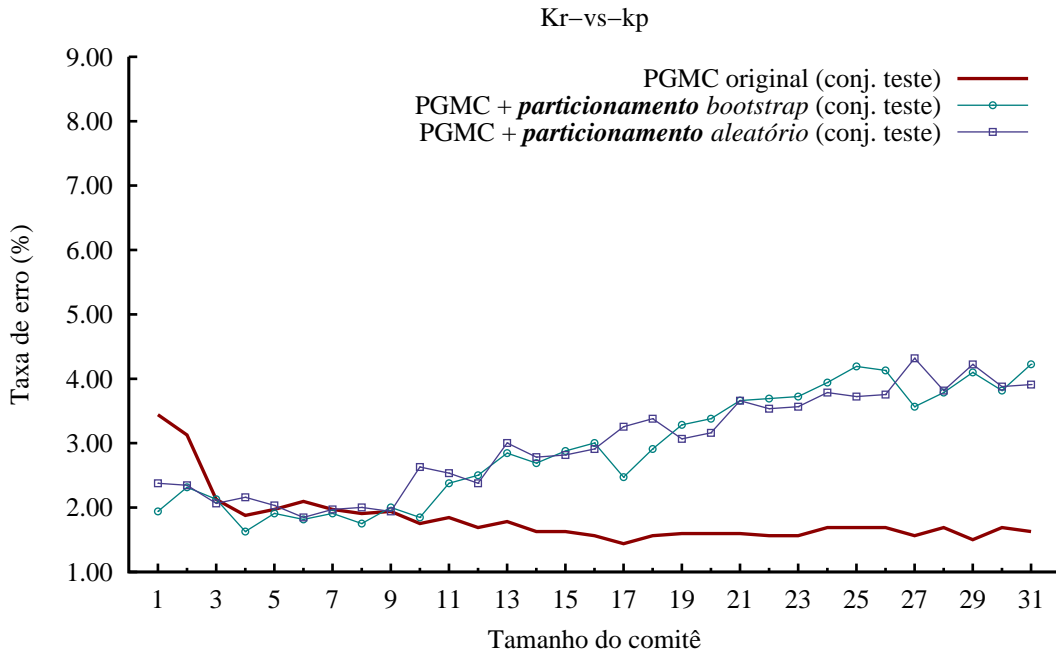


Figura 5.23: *Kr-vs-kp*: curva de erro no conjunto de teste (versão particionada).

finais de erro dos três algoritmos tecnicamente empataram.

Na Figura 5.25, referindo-se à base de dados *segmentation*, são mostrados os desempenhos das três variações do **PGMC**, onde observa-se que a taxa de erro das versões com particionamento mantiveram-se praticamente estáveis. Houve um tímido esboço de redução do erro no início da curva, mas esta tendência não se sustentou.

O desempenho na base de dados *splice*, Figura 5.26, é mais um exemplo de sustentação do poder de generalização à medida que o número de particionamentos aumenta.

Por fim, as curvas das versões com particionamento na base de dados *waveform* comportaram-se extraordinariamente bem (Figura 5.27). Não só conseguiram efeito semelhante ao ocorrido na classificação da base de dados *optdigits*, como também foram efetivamente capazes de superar a taxa de classificação da versão original do **PGMC**. Este desempenho torna-se mais saliente se for observado que a *waveform* é uma base de dados ruidosa.

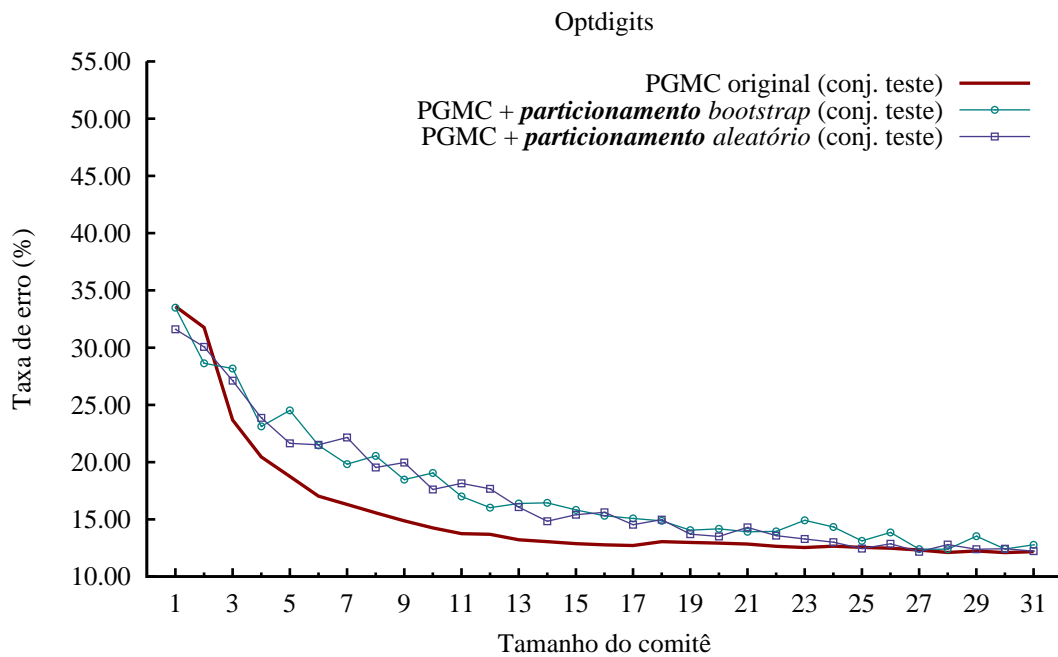


Figura 5.24: *Optdigits*: curva de erro no conjunto de teste (versão particionada).

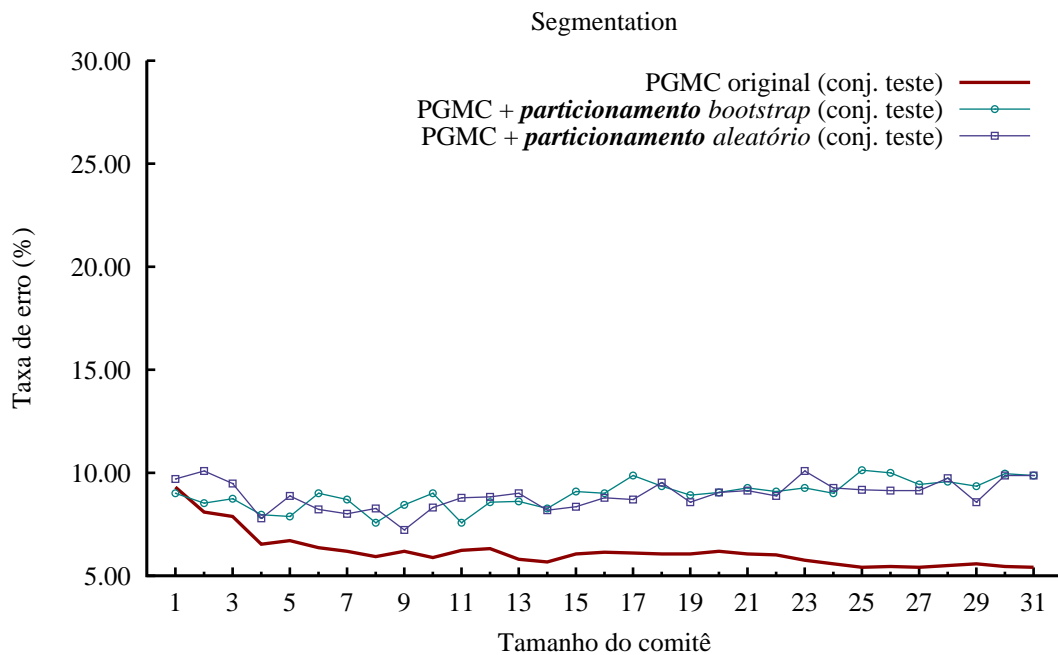


Figura 5.25: *Segmentation*: curva de erro no conjunto de teste (versão particionada).

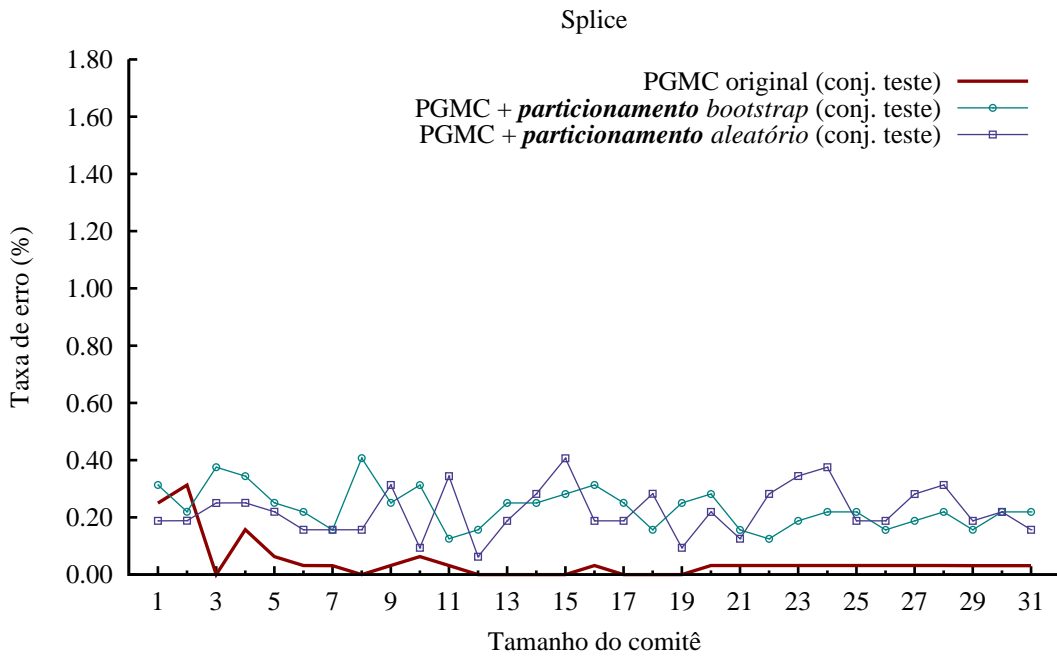


Figura 5.26: *Splice*: curva de erro no conjunto de teste (versão particionada).

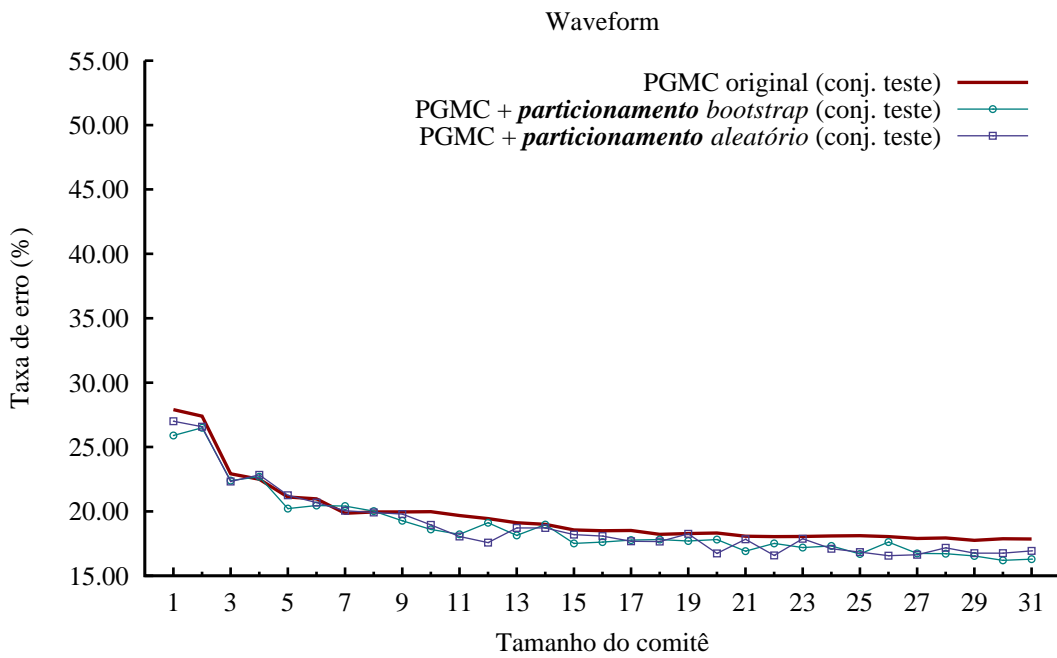


Figura 5.27: *Waveform*: curva de erro no conjunto de teste (versão particionada).

### 5.5.2.1 Análise e Discussão

Essa modalidade de experimentos, no qual o conjunto de treinamento é sucessivamente particionado, oferece apelo incontestável para aplicações em ambiente computacional de alto desempenho em função da redução linear do tempo requerido. Os experimentos mostraram que as versões particionadas do **PGMC** podem, em certos casos, não apenas sustentar a taxa de erro mas, surpreendentemente, diminuí-las à medida que a quantidade de partições cresce.

Analisando as propriedades das bases de dados na qual as versões com particionamento obtiveram desempenhos expressivos, em particular a *optdigits* e *waveform*, nota-se uma forte correlação entre o número de registros e o decaimento da taxa de erro; não à toa, a *optdigits* e *waveform* são as bases de dados com as maiores quantidade de registros. Tal constatação, no entanto, não surpreende, já que é sabido que a disponibilidade de um maior número de amostras por conjunto de treinamento favorece a capacidade de generalização.

O tamanho da base de dados é um indicador importante para se conseguir boa evolução na taxa de erro, mas não é uma característica determinante. Por exemplo, comparando-se a base de dados *segmentation* (2310 registros) com a *allhypo* (2514 registros), percebe-se que o desempenho da *segmentation* foi bem melhor. Um outro fator que parece influenciar é a distribuição das classes; a base de dados *allhypo*, por exemplo, exibe uma distribuição de classes evidentemente desproporcional.

No que concerne aos métodos de amostragem, o modo *aleatório* e o *bootstrap* comportaram-se de forma bastante similar. Em alguns problemas, no entanto, o *aleatório* apresentou sensível vantagem. Quando não existe interseção entre as partições, ou quando esta interseção é baixa, não se justifica o uso de métodos de amostragem que visem criar “pertubações”, como o *bootstrap*. Nos experimentos com particionamento, o *bootstrap* na verdade dificultou a generalização, pois a propriedade de *reposição*, que na prática duplica algumas amostras de dados enquanto ignora outras, diminui o número efetivo de amostras distintas (grau de discriminação) nos conjuntos de treinamento.

## 5.6 Conclusões

Foram realizadas duas baterias de experimentos neste capítulo. A primeira comparando o desempenho do algoritmo **PGMC** com o do *bagging* e do *boosting*. A segunda bateria testou os limites do **PGMC** conforme o conjunto de treinamento era progressivamente particionado.

Ficou demonstrado experimentalmente que, nas bases de dados estudadas, o algoritmo **PGMC** possui melhor poder de classificação do que o *bagging* e, na maioria dos casos, do que o *boosting*. Os experimentos indicaram também que o **PGMC** é capaz de conjugar bem as características do *bagging*, especialmente no que concerne à robustez perante a ruídos e *overfitting*, e as do *boosting*, particularmente a precisão de classificação.

Não obstante, os resultados obtidos nos experimentos com particionamento do conjunto de treinamento são bastante animadores. Foi revelado que a qualidade do desempenho é em geral proporcional ao tamanho da base de dados; em outras palavras, as bases de dados que mais demandam poder computacional são justamente as mais beneficiadas. Contudo, o número de registros de base de dados não é o único fator que impacta na qualidade de classificação; o balanceamento da distribuição de classes, por exemplo, parece ser relevante. A análise rigorosa das condições necessárias e suficientes para que o particionamento comporte-se de maneira apropriada é certamente um importante tópico de pesquisa futura (vide Seção 6.1).

Finalmente, cabe frisar que as análises conduzidas neste capítulo não podem ser extrapoladas para outros domínios além da programação genética. Estritamente, os resultados experimentais obtidos dizem respeito, especificamente, à implementação particular da programação genética e demais algoritmos que foi utilizada nos experimentos. Não obstante, embora tenha se tentado coletar problemas de classificação de dados dos mais diversos perfis, o universo de problemas disponíveis é enorme, e certamente não pôde ser rigorosamente representado nos problemas estudados. Portanto, sob o rigor científico, não há garantias de que o comportamento observado aqui se mantenha em outros domínios, implementações ou aplicações.<sup>9</sup>

---

<sup>9</sup>Entretanto, espera-se que a aplicação do algoritmo **PGMC** em outros domínios e implementações comporte-se de maneira similar à observada nos experimentos conduzidos neste capítulo.

# Capítulo 6

## Conclusões

Este trabalho descreveu a proposta de um novo modelo de algoritmo distribuído para a tarefa de classificação de dados, apoiando-se na noção de “decomposição e recomposição” do problema. Para tanto, adotou-se a efetiva abordagem de combinação de classificadores, a qual foi devidamente integrada à meta-heurística da computação evolutiva denominada programação genética, que dedica-se à evolução simulada de programas de computador em uma certa linguagem. Na proposta, múltiplas populações de programas classificadores—cada uma focada em uma região específica do domínio do problema—evoluem em paralelo em um sistema cooperativo, no qual soluções promissoras descobertas localmente são periodicamente compartilhadas em benefício do processo evolutivo global. No nível intra-populacional, por sua vez, os membros classificadores co-evoluem competitivamente com amostras de treinamento, visando criar tensões bilaterais positivas que estimulam a aceleração da escalada evolucionária. Ao final do processo, os melhores membros das populações são agregados sob a estrutura de um comitê de votação para formarem o classificador final, que, espera-se, seja mais preciso e robusto do que suas partes isoladamente.

A abordagem também se inspirou nos conceitos bem estabelecidos do *bagging* e *boosting*, dois famosos combinadores de soluções, cujas essências foram adaptadas para a proposta no domínio da programação genética. Do *bagging* aproveitou-se o mecanismo de amostragem dos conjuntos de treinamento e a estrutura naturalmente independente/paralela; do *boosting* foram incorporados a ideia de direcionamento da busca às amostras mais dificilmente classificáveis e o esquema de estimativa de confiabilidade dos membros votantes.

## Contribuições

A proposta, nos moldes descritos, deu origem ao algoritmo **PGMC**, que figura como uma das duas distintas contribuições desta tese. A segunda contribuição deve-se à implementação computacional do **PGMC** (e outros algoritmos)—apelidada de *gpclassifier*—que situa-se provavelmente como uma das mais sofisticadas implementações livres/públicas de programação genética para classificação de dados em ambiente computacional de alto desempenho. Sua licença livre beneficia, de forma direta e não-burocrática, instituições, pesquisadores e indústria, e garante ainda que qualquer trabalho derivado, na forma de contribuição ou adaptação, seja distribuído livremente sem qualquer restrição adicional.

## Resultados Experimentais

Foram realizadas duas baterias de experimentos, cada uma em nove problemas de classificação de dados. A primeira bateria comparou o desempenho do **PGMC**, em sua concepção original, em relação ao do *bagging* e *boosting* no contexto da programação genética. Já a segunda bateria mediu o desempenho do **PGMC** à medida que a base de dados foi sendo progressivamente *particionada* entre as populações.

A primeira sessão de experimentos corroborou as expectativas em relação ao **PGMC**, isto é, demonstrou que a integração à programação genética proposta, de fato, conseguiu unir as qualidades do *bagging* e *boosting*. O **PGMC** foi absolutamente superior ao *bagging*, enquanto que dominou o desempenho do *boosting* na maioria dos problemas. Após mostrar-se largamente mais preciso do que o *bagging*, bastaria que o **PGMC** produzisse resultados compatíveis ao do *boosting* para que sua proposta fosse justificada, já que o **PGMC** é um algoritmo intrinsecamente paralelo enquanto que o *boosting* é sequencial.

A sessão de experimentos com a base de dados particionada revelou resultados bastante animadores. Embora em geral tenha havido clara tendência de degradação na precisão de classificação para as bases de dados menores, pôde-se perceber que certas características do problema favorecem a possibilidade de particionamento. Dentre estas características a mais óbvia é o tamanho da base de dados: quanto maior é a base de dados, maior é a chance de aplicação bem sucedida do particiona-

mento. Felizmente, são justamente as bases de dados maciças (de grande porte) que mais demandam processamento, e portanto são as maiores beneficiárias da estratégia de particionamento em ambiente computacional de alto desempenho.

## 6.1 Trabalhos Futuros

Em virtude da necessidade de delimitação do escopo de pesquisa, interessantes e potenciais linhas de trabalho não foram abordadas na presente tese. Apresenta-se a seguir uma síntese de direções para pesquisa futura relacionadas ao tema deste trabalho.

**Exploração do modelo de ilhas** O modelo de paralelismo por ilha é notavelmente poderoso e flexível, ainda que o conceito seja trivial. Nesta tese, entretanto, explorou-se apenas uma singela fração desse modelo. Pode-se destacar duas direções promissoras de trabalho:

1. Estudo aprofundado das diversas topologias e taxas migratórias e como estes afetam a diversidade/qualidade do comitê final.
2. Investigação acerca da comunicação de outras informações além da migração de indivíduos classificadores.

Em relação ao segundo item, pode-se pensar, por exemplo, em troca contínua de informações a respeito das amostras de dados, especialmente no que tange a distribuição corrente de seus pesos (“dificuldade”) nas populações. Dessa forma poderia-se: (i) induzir nichos no arquipélago, onde cada ilha ficaria responsável pela exploração de segmentos de amostras de dados que seriam definidos dinamicamente; ou (ii) aproximar mais fielmente o esquema do *boosting* de “comunicação de pesos”, por meio da propagação entre as ilhas de informações sobre os pesos das amostras de dados.

**Investigação da estratégia de particionamento** Nos experimentos com o particionamento da base de dados, mostrou-se nítida a forte relação entre o tamanho da base de dados e o bom desempenho conforme o número de partições aumentava. Mas, aparentemente, o tamanho não é o único fator que governa

esta relação. Portanto, dada a inegável importância da aplicação da estratégia de particionamento devido à enorme redução do tempo computacional, faz-se de grande relevância a investigação acerca das características/condições exatas que facultam sua aplicabilidade.

**Ponderação dinâmica** Tanto o *boosting* como o **PGMC** adotam um esquema de ponderação de votos do tipo estático, isto é, após estabelecido a confiança de cada membro do comitê—ao final do processo de treinamento—, este valor permanece constante para toda e qualquer nova predição.

A grande limitação desses mecanismos tem origem no fato de ignorarem completamente o perfil da amostra de dados sendo predita. Recorrendo à uma analogia, seria *como se um grupo de médicos de diferentes especialidades fosse diagnosticar um paciente com problemas neurológicos e as opiniões de dermatologistas pudessem ter peso igual ou maior do que as opiniões de neurocirurgiões*.

Como forma de abordar mais sabiamente esta questão, propõe-se como tópico de pesquisa futura o tratamento da limitação descrita via conceito de *ponderação dinâmica*. Neste paradigma surge a figura de um *intermediador* que determina dinamicamente a confiança de um determinado classificador baseando-se nas características da amostra de dados sendo classificada. De volta à analogia, seria *como se o paciente antes de receber o diagnóstico fosse consultado por um clínico geral que avaliaria sua enfermidade e o encaminharia para os especialistas na doença*.

A implementação desta abordagem imporia uma sutil penalização computacional em decorrência do surgimento da figura do intermediador, penalização esta que poderia ser plenamente justificada pelo eventual ganho na precisão das taxas de classificação. Efetivamente, o intermediador nada mais é do que um algoritmo que calcula a similaridade entre a amostra a ser classificada e a região de especialidade de um classificador; quanto mais próximo a amostra estiver da especialidade do classificador, maior é a confiança de voto.

Portanto, em síntese, os componentes de imediata análise deste tópico seriam: (i) as medidas de similaridade que melhor expressem a real proximidade das

entidades; e (ii) os mecanismos exatos de atribuição dinâmica de pesos em função do grau de similaridade. Quanto ao primeiro item, poderia-se sugerir a distância euclidiana como medida de similaridade, todavia, esta medida é bastante sensível à diferença de magnitude dos valores da base de dados e às correlações entre os atributos; uma medida mais robusta e apropriada seria a distância de Mahalanobis [98], em especial sua variação formulada para tipos mistos de dados (incluindo tipos nominal e ordinal) [99].

A ideia da ponderação dinâmica, no entanto, não é inédita. Por exemplo, o algoritmo *iBoost* [100] implementa este conceito valendo-se de um classificador previamente treinado que prediz se um certo membro do comitê é capaz ou não de classificar uma dada amostra. Se a resposta for *sim*, então a confiança dele é acrescida por um certo valor; e decrescida caso contrário. Contudo, no *iBoost* os valores que são acrescidos ou decrescidos são estáticos. Uma medida de similaridade, por outro lado, poderia oferecer valores *contínuos* de confiança, o que parece ser mais adequado.

**Expansão da validação experimental** Sob a intenção de se obter resultados justos e estatisticamente válidos, fez-se necessário controlar o ambiente de experimentação (p. ex., mesma implementação base e parâmetros), e portanto restringiu-se a abrangência dos experimentos. Dessa forma, seria cientificamente relevante—desde que haja rigor metodológico—no que concerne à abordagem proposta:

1. Comparar o seu desempenho com outros métodos similares, especialmente os relativos à programação genética, como os descritos na Seção 3.2.
2. Medir o seu desempenho em outras tarefas de classificação de dados, tanto em problemas típicos de *benchmark* quanto em problemas reais, de grande porte.

# Referências Bibliográficas

- [1] BREIMAN, L., “Bagging Predictors”, *Machine Learning*, v. 24, n. 2, pp. 123–140, 1996.
- [2] SCHAPIRE, R. E., “The Strength of Weak Learnability”, *Mach. Learn.*, v. 5, n. 2, pp. 197–227, 1990.
- [3] FREUND, Y., SCHAPIRE, R. E., “Experiments with a New Boosting Algorithm”. In: *International Conference on Machine Learning*, pp. 148–156, 1996.
- [4] DIETTERICH, T. G., “Ensemble Methods in Machine Learning”, *Lecture Notes in Computer Science*, v. 1857, pp. 1–15, 2000.
- [5] LICHODZIJEWSKI, P., HEYWOOD, M. I., ZINCIR-HEYWOOD, A. N., “CasGP: building cascaded hierarchical models using niching”. In: *Congress on Evolutionary Computation*, pp. 1180–1187, 2005.
- [6] MCINTYRE, A., HEYWOOD, M., “MOGE: GP classification problem decomposition using multi-objective optimization”. In: *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 863–870, New York, NY, USA, ACM Press, 2006.
- [7] LEMCZYK, M., HEYWOOD, M. I., “Training Binary GP Classifiers Efficiently: A Pareto-coevolutionary Approach.” In: *EuroGP*, v. 4445, *Lecture Notes in Computer Science*, pp. 229–240, 2007.
- [8] KOZA, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA, MIT Press, 1992.

- [9] OLTEAN, M., GROSAN, C., DIOSAN, L., et al., “Genetic Programming with Linear Representation: a survey”, *International Journal on Artificial Intelligence Tools*, v. 18, n. 2, pp. 197–238, 2009.
- [10] POLI, R., “Parallel Distributed Genetic Programming”, In: CORNE, D., DORIGO, M., GLOVER, F. (eds), *New Ideas in Optimization*, chap. 27, pp. 403–431, *Advanced Topics in Computer Science*, Maidenhead, Berkshire, England, McGraw-Hill, 1999.
- [11] TELLER, A., “Evolving Programmers: The Co-evolution of Intelligent Recombination Operators”, In: ANGELINE, P. J., KINNEAR, JR., K. E. (eds), *Advances in Genetic Programming 2*, chap. 3, pp. 45–68, Cambridge, MA, USA, MIT Press, 1996.
- [12] MILLER, J. F., SMITH, S. L., “Redundancy and Computational Efficiency in Cartesian Genetic Programming”, *IEEE Transactions on Evolutionary Computation*, v. 10, n. 2, pp. 167–174, April 2006.
- [13] PELIKAN, M., KVASNICKA, V., POSPICHAL, J., “Read’s linear codes and genetic programming”. In: *Genetic Programming 1997: Proceedings of the Second Annual Conference*, p. 268, Stanford University, CA, USA, Morgan Kaufmann, 13-16 July 1997.
- [14] AUGUSTO, D. A., BARBOSA, H. J. C., “Symbolic Regression via Genetic Programming”. In: *SBRN ’00: Proceedings of the VI Brazilian Symposium on Neural Networks (SBRN’00)*, p. 173, Washington, DC, USA, IEEE Computer Society, 2000.
- [15] O’NEILL, M., RYAN, C., “Grammatical Evolution”, *IEEE Transactions on Evolutionary Computation*, v. 5, n. 4, pp. 349–358, 2001.
- [16] MONTANA, D. J., *Strongly Typed Genetic Programming*, Tech. Rep. #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 1994.

- [17] WHIGHAM, P. A., “Grammatically-based Genetic Programming”. In: *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pp. 33–41, Tahoe City, California, USA, 1995.
- [18] POLI, R., LANGDON, W. B., MCPHEE, N. F., *A field guide to genetic programming*. Morrisville, NC, USA, Lulu Press, 2008, (With contributions by J. R. Koza).
- [19] GOLDBERG, D. E., DEB, K., “A comparative analysis of selection schemes used in genetic algorithms”. In: *Foundations of Genetic Algorithms*, pp. 69–93, 1991.
- [20] BRINDLE, A., *Genetic Algorithms for Function Optimization*, Phd thesis, University of Alberta, 1981.
- [21] SYSWERDA, G., “Uniform Crossover in Genetic Algorithms”. In: *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2–9, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1989.
- [22] WHITLEY, D., “The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best”. In: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 116–121, 1989.
- [23] EFRON, B., TIBSHIRANI, R. J., *An Introduction to the Bootstrap*. New York, NY, USA, Chapman & Hall/CRC, May 1994.
- [24] FU, W. J., CARROLL, R. J., WANG, S., “Estimating misclassification error with small samples via bootstrap cross-validation”, *Bioinformatics*, v. 21, n. 9, pp. 1979–1986, May 2005.
- [25] EFRON, B., TIBSHIRANI, R., “Improvements on Cross-Validation: The .632+ Bootstrap Method”, *Journal of the American Statistical Association*, v. 92, n. 438, pp. 548–560, 1997.
- [26] JIANG, W., SIMON, R., “A comparison of bootstrap methods and an adjusted bootstrap approach for estimating the prediction error in microarray classification.” *Stat Med*, July 2007.

- [27] FROMONT, M., “Model selection by bootstrap penalization for classification”, *Mach. Learn.*, v. 66, n. 2-3, pp. 165–207, 2007.
- [28] KOHAVI, R., “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection”. In: *IJCAI*, pp. 1137–1145, 1995.
- [29] WITTEN, I. H., FRANK, E., *Data Mining: Practical Machine Learning Tools and Techniques*. 2 ed. *Morgan Kaufmann Series in Data Management Systems*, San Francisco, CA, USA, Morgan Kaufmann, June 2005.
- [30] FREUND, Y., “Boosting a weak learning algorithm by majority”, *Inf. Comput.*, v. 121, n. 2, pp. 256–285, 1995.
- [31] FREUND, Y., “An Adaptive Version of the Boost By Majority Algorithm”. In: *In Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pp. 102–113, 2000.
- [32] ROSSET, S., “Robust boosting and its relation to bagging”. In: *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 249–255, New York, NY, USA, ACM Press, 2005.
- [33] DEMIRIZ, A., BENNETT, K. P., SHAWE-TAYLOR, J., “Linear Programming Boosting via Column Generation”, *Mach. Learn.*, v. 46, n. 1-3, pp. 225–254, 2002.
- [34] KRAUSE, N., SINGER, Y., “Leveraging the margin more carefully”. In: *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, p. 63, New York, NY, USA, ACM Press, 2004.
- [35] SCHAPIRE, R. E., SINGER, Y., “Improved Boosting Algorithms Using Confidence-rated Predictions”, *Mach. Learn.*, v. 37, n. 3, pp. 297–336, 1999.
- [36] FREUND, Y., SCHAPIRE, R. E., “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pp. 23–37, London, UK, Springer-Verlag, 1995.

- [37] QUINLAN, J. R., “Bagging, Boosting, and C4.5”. In: *AAAI/IAAI, Vol. 1*, pp. 725–730, 1996.
- [38] MACLIN, R., OPITZ, D., “An Empirical Evaluation of Bagging and Boosting”. In: *AAAI/IAAI*, pp. 546–551, 1997.
- [39] SKURICHINA, M., KUNCHEVA, L., DUIN, R. P. W., “Bagging and Boosting for the Nearest Mean Classifier: Effects of Sample Size on Diversity and Accuracy”. In: *MCS '02: Proceedings of the Third International Workshop on Multiple Classifier Systems*, pp. 62–71, London, UK, Springer-Verlag, 2002.
- [40] TAN, A. C., GILBERT, D., “An empirical comparison of supervised machine learning techniques in bioinformatics”. In: *APBC '03: Proceedings of the First Asia-Pacific bioinformatics conference on Bioinformatics 2003*, pp. 219–222, Darlinghurst, Australia, Australia, Australian Computer Society, Inc., 2003.
- [41] KOTSIANTIS, S., PINTELAS, P., “Combining Bagging and Boosting”, *International Journal of Computational Intelligence and Applications*, v. 1, n. 4, pp. 324–333, 2004.
- [42] FERREIRA, A., *Survey on Boosting Algorithms for Supervised and Semi-supervised Learning*, Tech. rep., Instituto Superior de Engenharia de Lisboa, 2007.
- [43] IBA, H., “Bagging, Boosting, and Bloating in Genetic Programming”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, v. 2, pp. 1053–1060, Orlando, Florida, USA, Morgan Kaufmann, 13-17 July 1999.
- [44] PARIS, G., ROBILLIARD, D., FONLUPT, C., “Applying Boosting Techniques to Genetic Programming”. In: *Selected Papers from the 5th European Conference on Artificial Evolution*, pp. 267–280, London, UK, Springer-Verlag, 2002.

- [45] FOLINO, G., PIZZUTI, C., SPEZZANO, G., “Ensemble Techniques for Parallel Genetic Programming Based Classifiers”. In: *Proceedings of the Sixth European Conference on Genetic Programming (EuroGP-2003)*, v. 2610, LNCS, pp. 59–69, Essex, UK, Springer Verlag, 2003.
- [46] FOLINO, G., PIZZUTI, C., SPEZZANO, G., “Boosting technique for Combining Cellular GP Classifiers”. In: *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, v. 3003, LNCS, pp. 47–56, Coimbra, Portugal, Springer-Verlag, 5-7 April 2004.
- [47] PAREDIS, J., “Steps towards Co-evolutionary Classification Neural Networks”. In: *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pp. 102–108, 1994.
- [48] PAREDIS, J., “Coevolutionary computation”, *Artif. Life*, v. 2, n. 4, pp. 355–375, 1995.
- [49] PAREDIS, J., “Coevolutionary Life-Time Learning”. In: *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pp. 72–80, London, UK, Springer-Verlag, 1996.
- [50] PAREDIS, J., “Coevolutionary Process Control”. In: *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*, pp. 394–398, 1997.
- [51] PAREDIS, J., “Coevolution, Memory and Balance”. In: *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pp. 1212–1217, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1999.
- [52] PAREDIS, J., “The evolution of behavior: some experiments”. In: *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pp. 419–426, 1991.
- [53] LOBO, F. G., LIMA, C. F., MICHALEWICZ, Z., (eds), *Parameter Setting in Evolutionary Algorithms*. v. 54. *Studies in Computational Intelligence*, 2007.

- [54] NOLFI, S., FLOREANO, D., “How Co-Evolution can Enhance the Adaptive Power of Artificial Evolution: Implications for Evolutionary Robotics”. In: *Proceedings of the First European Workshop on Evolutionary Robotics*, pp. 22–38, London, UK, Springer-Verlag, 1998.
- [55] AUGUSTO, D. A., *Co-Evolução Amostra-Classificador Integrada à Programação Genética Gramatical para a Classificação de Dados*, Master’s Thesis, Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brasil, Dezembro 2004.
- [56] TOMASSINI, M., *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time (Natural Computing Series)*. Secaucus, NJ, USA, Springer-Verlag New York, Inc., 2005.
- [57] BROWN, G., WYATT, J., HARRIS, R., et al., “Diversity creation methods: A survey and categorisation”, *Journal of Information Fusion*, v. 6, pp. 5–20, 2005.
- [58] KUNCHEVA, L. I., WHITAKER, C. J., “Measures of diversity in classifier ensembles”, *Machine Learning*, v. 51, pp. 181–207, 2003.
- [59] SKURICHINA, M., KUNCHEVA, L., DUIN, R. P. W., “Bagging and Boosting for the Nearest Mean Classifier: Effects of Sample Size on Diversity and Accuracy”. In: *MCS ’02: Proceedings of the Third International Workshop on Multiple Classifier Systems*, pp. 62–71, London, UK, Springer-Verlag, 2002.
- [60] BRAMEIER, M., BANZHAF, W., “Evolving Teams of Predictors with Linear Genetic Programming”, *Genetic Programming and Evolvable Machines*, v. 2, n. 4, pp. 381–407, 2001.
- [61] POPPER, K. R., *The Logic of Scientific Discovery (Routledge Classics)*. London, UK, Routledge, March 2002.
- [62] SONNENBURG, S., BRAUN, M. L., ONG, C. S., et al., “The Need for Open Source Software in Machine Learning”, *J. Mach. Learn. Res.*, v. 8, pp. 2443–2466, 2007.

- [63] LEWINE, D. A., *POSIX programmer's guide: writing portable UNIX programs with the POSIX.1 standard*. Sebastopol, CA, USA, O'Reilly & Associates, Inc., 1991.
- [64] PROVOST, F., KOHAVI, R., "On Applied Research in Machine Learning". In: *Machine learning*, pp. 127–132, 1998.
- [65] FORUM, M. P. I., *MPI-2: Extensions to the Message-Passing Interface*, Tech. rep., University of Tennessee, Knoxville, TN, USA, July 1997.
- [66] FORUM, M. P. I., *MPI: A Message-Passing Interface Standard, Version 2.1*, Tech. rep., University of Tennessee, Knoxville, TN, USA, June 2008.
- [67] CHOMSKY, N., "Three Models for the Description of Language", *iretit*, v. 2, n. 3, pp. 113–124, 1956.
- [68] GRUAU, F., "On using syntactic constraints with genetic programming". In: *Advances in genetic programming: volume 2*, pp. 377–394, Cambridge, MA, USA, MIT Press, 1996.
- [69] HAYNES, T., WAINWRIGHT, R., SEN, S., et al., "Strongly typed genetic programming in evolving cooperation strategies". In: *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pp. 271–278, Pittsburgh, PA, USA, Morgan Kaufmann, 15-19 1995.
- [70] AUGUSTO, D. A., BARBOSA, H. J., EBECKEN, N. F., "Coevolution of data samples and classifiers integrated with grammatically-based genetic programming for data classification". In: *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 1171–1178, New York, NY, USA, ACM, 2008.
- [71] WHIGHAM, P. A., *Grammatical Bias for Evolutionary Learning*, Ph.D. Thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 14 Oct. 1996.
- [72] QUINLAN, J., "Improved use of continuous attributes in C4.5", *Journal of Artificial Intelligence Research*, v. 4, pp. 77–90, 1996.

- [73] HAN, J., KAMBER, M., *Data Mining, Second Edition, Second Edition : Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems) (The Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA, Morgan Kaufmann, January 2006.
- [74] ZHU, J., ROSSET, S., ZOU, H., et al., *Multi-class AdaBoost*, Tech. rep., University of Michigan, 2006.
- [75] GRAMA, A., KARYPIS, G., GUPTA, A., et al., *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Reading, MA, Addison-Wesley, 2003.
- [76] LUKE, S., PANAIT, L., “A comparison of bloat control methods for genetic programming”, *Evol. Comput.*, v. 14, n. 3, pp. 309–344, 2006.
- [77] MCPHEE, N. F., MILLER, J. D., “Accurate Replication in Genetic Programming”. In: *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 303–309, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1995.
- [78] LANGDON, W. B., POLI, R., “Fitness Causes Bloat”. In: *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pp. 13–22, July 1997.
- [79] SOULE, T., FOSTER, J. A., “Removal Bias: a New Cause of Code Growth in Tree Based Evolutionary Programming”. In: *1998 IEEE International Conference on Evolutionary Computation*, pp. 781–186, Anchorage, Alaska, USA, IEEE Press, 5-9 May 1998.
- [80] LANGDON, W. B., SOULE, T., POLI, R., et al., “The evolution of size and shape”, In: *Advances in genetic programming: volume 3*, chap. 8, pp. 163–190, Cambridge, MA, USA, MIT Press, 1999.
- [81] BANZHAF, W., LANGDON, W. B., “Some Considerations on the Reason for Bloat”, *Genetic Programming and Evolvable Machines*, v. 3, n. 1, pp. 81–91, 2002.

- [82] ROSCA, J., “A Probabilistic Model of Size Drift”, In: RIOLO, R. L., WORZEL, B. (eds), *Genetic Programming Theory and Practice*, chap. 8, pp. 119–136, Hingham, MA, USA, Kluwer, 2003.
- [83] DIGNUM, S., POLI, R., “Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat”. In: *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, v. 2, pp. 1588–1595, London, ACM Press, 7-11 July 2007.
- [84] POLI, R., LANGDON, W. B., DIGNUM, S., “On the Limiting Distribution of Program Sizes in Tree-based Genetic Programming”. In: *Proceedings of the 10th European Conference on Genetic Programming*, v. 4445, *Lecture Notes in Computer Science*, pp. 193–204, Valencia, Spain, Springer, 11 - 13 April 2007.
- [85] CRAWFORD-MARKS, R., SPECTOR, L., “Size Control Via Size Fair Genetic Operators In The PushGP Genetic Programming System”. In: *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 733–739, New York, Morgan Kaufmann Publishers, 9-13 July 2002.
- [86] JR, K. E. K., “Evolving a Sort: Lessons in Genetic Programming”. In: *in Proceedings of the 1993 International Conference on Neural Networks*, pp. 881–888, 1993.
- [87] SINGLETON, A., “Genetic Programming with C++”, *BYTE*, pp. 171–176, February 1994.
- [88] ZHANG, B.-T., OHM, P., MÜHLENBEIN, H., “Evolutionary Induction of Sparse Neural Trees”, *Evolutionary Computation*, v. 5, n. 2, pp. 213–236, 1997.
- [89] POLI, R., MCPHEE, N. F., *Covariant Parsimony Pressure for Genetic Programming*, Tech. Rep. CES-480, Department of Computing and Electronic Systems, University of Essex, UK, 2008.

- [90] WELFORD, B. P., “Note on a Method for Calculating Corrected Sums of Squares and Products”, *Technometrics*, v. 4, n. 3, pp. 419–420, 1962.
- [91] KNUTH, D. E., *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition) (Art of Computer Programming Volume 2)*. 3 ed. Reading, MA, Addison-Wesley Professional, November 1997.
- [92] JONATHAN, P., KRZANOWSKI, W. J., MCCARTHY, W. V., “On the use of cross-validation to assess performance in multivariate prediction”, *Statistics and Computing*, v. 10, n. 3, pp. 209–229, 2000.
- [93] ASUNCION, A., NEWMAN, D., “UCI Machine Learning Repository”, 2007.
- [94] DORAISAMY, H., VICE, D. H., HALLECK, P. M., “Detection of hydrocarbon reservoir boundaries using neural network analysis of surface geochemical data”, *AAPG Bulletin*, v. 84, n. 12, pp. 1893–1904, 2000.
- [95] ESPÍNDOLA, R. P., *Sistema Inteligente para Classificação de Dados*, Ph.D. Thesis, Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brasil, Dezembro 2004.
- [96] BREIMAN, L., FRIEDMAN, J., STONE, C. J., et al., *Classification and Regression Trees*. New York, NY, USA, Chapman & Hall/CRC, January 1984.
- [97] LIU, H., MOTODA, H., *Computational Methods of Feature Selection (Chapman & Hall/Crc Data Mining and Knowledge Discovery Series)*. New York, NY, USA, Chapman & Hall/CRC, 2007.
- [98] MAHALANOBIS, P. C., “On the generalised distance in statistics”. In: *Proceedings National Institute of Science, India*, v. 2, n. 1, pp. 49–55, April 1936.
- [99] DE LEON, A. R., CARRIÈRE, K. C., “A generalized Mahalanobis distance for mixed data”, *J. Multivar. Anal.*, v. 92, n. 1, pp. 174–185, 2005.
- [100] KARMAKER, A., YOON, K., NGUYEN, C., et al., “iBoost: Boosting using an instance-based exponential weighting scheme”, *Int. J. Hybrid Intell. Syst.*, v. 4, n. 4, pp. 243–254, 2007.

# Apêndice A

## Síntese dos Experimentos

### A.1 Experimentos *Sem* Particionamento

#### A.1.1 Taxas de Erro sobre o Conjunto de Teste

SEM PARTICIONAMENTO — CONJ. DE TESTE				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	<i>Bagging</i> (%)	<i>Boosting</i> (%)
<i>abalone</i>	1	76.03 ± 3.01	76.19 ± 1.83	77.78 ± 3.17
	2	76.12 ± 3.26	76.07 ± 1.56	77.73 ± 2.28
	3	74.78 ± 1.19	75.99 ± 1.58	77.49 ± 2.74
	4	74.80 ± 1.26	76.26 ± 1.77	77.66 ± 2.90
	5	74.66 ± 1.15	76.23 ± 1.94	78.33 ± 2.45
	6	74.11 ± 1.19	76.20 ± 2.27	78.22 ± 2.78
	7	73.99 ± 1.13	75.94 ± 2.41	78.26 ± 2.16
	8	74.01 ± 1.30	75.93 ± 1.93	77.85 ± 2.01
	9	74.16 ± 1.58	75.66 ± 2.22	78.02 ± 2.07
	10	73.97 ± 0.89	75.61 ± 2.05	77.96 ± 1.37
	11	74.21 ± 1.03	75.45 ± 1.88	77.92 ± 2.01
	12	74.23 ± 1.18	75.63 ± 1.87	77.85 ± 2.36
	13	74.20 ± 1.18	75.64 ± 1.93	78.17 ± 1.98
	14	74.11 ± 1.06	75.59 ± 1.87	78.16 ± 1.55
	15	74.28 ± 0.97	75.81 ± 1.83	77.99 ± 1.83
	16	74.19 ± 1.07	75.35 ± 1.50	77.63 ± 1.92
	17	74.50 ± 0.93	75.28 ± 1.34	78.10 ± 1.88
	18	74.13 ± 1.21	75.33 ± 1.61	78.52 ± 2.36
	19	74.30 ± 1.18	75.09 ± 1.48	78.22 ± 2.41
	20	74.30 ± 1.25	75.50 ± 1.57	78.48 ± 2.19
	21	74.40 ± 1.30	75.38 ± 1.51	79.13 ± 2.33

## SEM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>abalone</i>	22	74.30 ± 1.17	75.43 ± 1.42	78.83 ± 2.39
	23	74.14 ± 1.22	75.36 ± 1.42	78.47 ± 2.56
	24	73.97 ± 1.40	75.52 ± 1.70	78.72 ± 2.51
	25	74.30 ± 1.69	75.48 ± 1.46	78.83 ± 2.50
	26	74.15 ± 1.67	75.48 ± 1.51	78.76 ± 1.96
	27	74.16 ± 1.59	75.47 ± 1.65	78.74 ± 2.46
	28	74.23 ± 1.62	75.50 ± 1.66	78.98 ± 1.98
	29	74.18 ± 1.39	75.36 ± 1.50	78.94 ± 2.39
	30	74.18 ± 1.36	75.28 ± 1.68	79.28 ± 2.17
	31	74.21 ± 1.48	75.24 ± 1.41	79.21 ± 2.23
	<i>allhypo</i>	1	1.28 ± 0.96	5.61 ± 0.85
2		1.11 ± 0.85	4.78 ± 1.17	5.36 ± 1.82
3		1.32 ± 0.77	5.44 ± 1.25	5.11 ± 2.69
4		1.07 ± 0.85	4.99 ± 1.18	6.14 ± 4.45
5		0.91 ± 0.79	5.53 ± 0.94	5.11 ± 3.45
6		0.82 ± 0.33	5.40 ± 1.14	4.57 ± 4.27
7		0.87 ± 0.41	5.69 ± 0.95	4.25 ± 3.12
8		0.82 ± 0.43	5.40 ± 1.06	5.03 ± 4.77
9		0.78 ± 0.30	5.49 ± 1.05	4.12 ± 3.66
10		0.70 ± 0.58	5.57 ± 1.02	3.59 ± 2.08
11		0.82 ± 0.51	5.61 ± 1.00	4.00 ± 2.63
12		0.78 ± 0.56	5.61 ± 0.96	4.08 ± 2.38
13		0.78 ± 0.49	5.82 ± 0.88	4.66 ± 4.12
14		0.74 ± 0.61	5.65 ± 1.03	3.96 ± 2.00
15		0.74 ± 0.61	5.49 ± 1.26	4.04 ± 2.79
16		0.70 ± 0.58	5.40 ± 1.07	4.04 ± 2.39
17		0.70 ± 0.58	5.53 ± 1.03	4.12 ± 2.88
18		0.70 ± 0.48	5.40 ± 1.24	3.55 ± 1.59
19		0.74 ± 0.54	5.65 ± 1.01	3.71 ± 2.58
20		0.74 ± 0.47	5.57 ± 1.07	3.38 ± 1.38
21		0.74 ± 0.54	5.61 ± 0.99	3.38 ± 1.68
22		0.78 ± 0.63	5.57 ± 1.07	3.01 ± 1.78
23		0.70 ± 0.52	5.65 ± 0.99	3.42 ± 2.64
24		0.70 ± 0.52	5.61 ± 1.03	3.25 ± 1.96
25		0.70 ± 0.52	5.57 ± 1.07	3.13 ± 1.87
26		0.70 ± 0.52	5.53 ± 1.08	3.30 ± 1.66
27		0.74 ± 0.54	5.61 ± 1.03	2.84 ± 1.32
28		0.74 ± 0.54	5.61 ± 1.03	3.01 ± 1.60
29		0.74 ± 0.54	5.65 ± 1.01	2.89 ± 2.04
30		0.74 ± 0.54	5.53 ± 1.08	2.97 ± 1.86
31		0.74 ± 0.54	5.61 ± 0.99	2.97 ± 1.99
<i>arcene</i>	1	29.28 ± 12.86	29.35 ± 11.91	25.40 ± 5.63
	2	29.78 ± 11.87	32.85 ± 14.04	35.88 ± 12.50
	3	24.38 ± 12.47	25.92 ± 7.10	22.95 ± 7.69

## SEM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC (%)	Bagging (%)	Boosting (%)	
<i>arcene</i>	4	25.47 ± 12.04	22.37 ± 8.18	29.38 ± 10.93	
	5	21.45 ± 11.35	24.95 ± 7.63	24.45 ± 7.08	
	6	22.47 ± 8.77	24.40 ± 7.19	21.90 ± 9.11	
	7	21.93 ± 10.30	24.93 ± 8.83	20.37 ± 8.53	
	8	21.90 ± 8.92	21.47 ± 9.33	24.38 ± 11.60	
	9	20.92 ± 7.44	25.45 ± 10.52	20.45 ± 5.84	
	10	21.40 ± 8.09	23.43 ± 10.01	22.90 ± 10.62	
	11	18.87 ± 7.96	25.43 ± 10.18	18.95 ± 8.61	
	12	21.37 ± 7.65	24.38 ± 10.75	21.45 ± 8.71	
	13	19.42 ± 6.51	24.38 ± 10.49	21.90 ± 11.47	
	14	18.92 ± 6.19	25.35 ± 10.23	21.00 ± 11.28	
	15	15.40 ± 6.76	25.91 ± 10.45	20.95 ± 12.36	
	16	17.37 ± 6.57	26.41 ± 9.98	20.45 ± 12.47	
	17	18.37 ± 6.92	26.38 ± 11.40	21.42 ± 12.79	
	18	19.35 ± 6.93	25.86 ± 11.81	21.45 ± 13.63	
	19	19.40 ± 6.73	26.41 ± 10.99	20.45 ± 15.31	
	20	18.92 ± 5.22	23.90 ± 12.00	20.45 ± 14.53	
	21	16.52 ± 5.24	25.90 ± 10.55	20.90 ± 13.70	
	22	18.50 ± 3.25	24.93 ± 10.18	20.97 ± 13.49	
	23	16.54 ± 4.18	25.45 ± 9.46	20.90 ± 11.49	
	24	17.99 ± 5.65	24.93 ± 10.18	20.95 ± 12.52	
	25	15.52 ± 4.91	25.93 ± 10.00	22.45 ± 13.00	
	26	18.07 ± 6.48	24.93 ± 10.18	22.50 ± 12.53	
	27	19.05 ± 6.27	24.43 ± 11.07	21.90 ± 13.94	
	28	18.55 ± 5.94	25.90 ± 10.55	21.92 ± 12.31	
	29	17.54 ± 5.98	25.40 ± 10.71	20.40 ± 9.76	
	30	17.44 ± 5.12	25.40 ± 10.97	21.25 ± 11.77	
	31	19.45 ± 5.72	25.90 ± 10.55	22.40 ± 8.22	
	<i>geochemical</i>	1	4.16 ± 4.78	12.25 ± 14.76	9.08 ± 8.05
		2	3.54 ± 4.89	1.76 ± 5.58	6.07 ± 6.94
		3	3.54 ± 5.71	2.98 ± 3.15	4.23 ± 6.35
4		2.29 ± 4.83	3.61 ± 4.17	4.85 ± 6.19	
5		2.91 ± 4.90	2.98 ± 4.20	2.98 ± 5.74	
6		2.91 ± 4.90	2.36 ± 4.12	2.98 ± 5.74	
7		2.91 ± 4.90	2.95 ± 4.17	2.98 ± 5.74	
8		1.73 ± 3.93	2.36 ± 4.12	2.98 ± 5.74	
9		2.36 ± 4.12	4.27 ± 5.11	2.98 ± 5.74	
10		1.73 ± 3.93	2.39 ± 3.10	2.98 ± 5.74	
11		2.36 ± 4.12	4.86 ± 3.91	2.98 ± 5.74	
12		1.73 ± 3.93	3.64 ± 4.31	2.98 ± 5.74	
13		1.73 ± 3.93	3.64 ± 4.31	2.98 ± 5.74	
14		1.73 ± 3.93	3.02 ± 4.34	2.98 ± 5.74	
15		1.73 ± 3.93	3.02 ± 4.34	2.98 ± 5.74	
16		1.73 ± 3.93	2.39 ± 4.27	2.98 ± 5.74	

SEM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>geochemical</i>	17	1.73 ± 3.93	3.02 ± 4.34	2.98 ± 5.74
	18	1.73 ± 3.93	3.02 ± 4.34	2.98 ± 5.74
	19	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	20	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	21	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	22	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	23	1.73 ± 3.93	4.20 ± 4.98	2.98 ± 5.74
	24	1.73 ± 3.93	3.02 ± 4.34	2.98 ± 5.74
	25	1.73 ± 3.93	4.20 ± 4.98	2.98 ± 5.74
	26	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	27	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	28	1.73 ± 3.93	4.20 ± 4.98	2.98 ± 5.74
	29	1.73 ± 3.93	4.20 ± 4.98	2.98 ± 5.74
	30	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
	31	1.73 ± 3.93	3.61 ± 4.29	2.98 ± 5.74
<i>kr-vs-kp</i>	1	3.44 ± 2.31	5.35 ± 1.03	5.98 ± 1.57
	2	3.13 ± 2.40	5.26 ± 1.01	5.98 ± 1.57
	3	2.13 ± 0.86	5.51 ± 1.00	3.07 ± 1.05
	4	1.88 ± 1.03	5.48 ± 1.12	3.63 ± 1.18
	5	1.97 ± 1.07	5.63 ± 1.07	1.38 ± 0.72
	6	2.10 ± 0.74	5.63 ± 1.07	1.60 ± 1.40
	7	1.97 ± 0.77	5.63 ± 1.07	0.72 ± 0.63
	8	1.91 ± 0.85	5.66 ± 1.07	0.63 ± 0.53
	9	1.94 ± 0.83	5.63 ± 1.07	0.66 ± 0.37
	10	1.75 ± 0.80	5.66 ± 1.07	0.72 ± 0.42
	11	1.85 ± 0.88	5.63 ± 1.07	0.56 ± 0.51
	12	1.69 ± 0.69	5.63 ± 1.07	0.66 ± 0.43
	13	1.78 ± 0.79	5.60 ± 1.09	0.50 ± 0.45
	14	1.63 ± 0.92	5.60 ± 1.09	0.50 ± 0.34
	15	1.63 ± 0.92	5.60 ± 1.09	0.50 ± 0.45
	16	1.56 ± 0.87	5.60 ± 1.09	0.44 ± 0.42
	17	1.44 ± 0.96	5.60 ± 1.09	0.41 ± 0.42
	18	1.56 ± 0.82	5.60 ± 1.09	0.47 ± 0.45
	19	1.60 ± 0.96	5.60 ± 1.09	0.47 ± 0.45
	20	1.60 ± 0.96	5.60 ± 1.09	0.34 ± 0.45
	21	1.60 ± 0.96	5.57 ± 1.11	0.44 ± 0.42
	22	1.56 ± 1.01	5.57 ± 1.11	0.38 ± 0.44
	23	1.56 ± 1.01	5.57 ± 1.11	0.41 ± 0.42
	24	1.69 ± 0.91	5.57 ± 1.11	0.38 ± 0.44
	25	1.69 ± 0.91	5.57 ± 1.11	0.41 ± 0.44
	26	1.69 ± 0.91	5.57 ± 1.11	0.34 ± 0.43
	27	1.56 ± 1.05	5.57 ± 1.11	0.38 ± 0.41
	28	1.69 ± 0.91	5.57 ± 1.11	0.34 ± 0.43
	29	1.50 ± 0.96	5.57 ± 1.11	0.38 ± 0.41

SEM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>kr-vs-kp</i>	30	1.69 ± 0.91	5.57 ± 1.11	0.38 ± 0.41
	31	1.63 ± 0.82	5.57 ± 1.11	0.34 ± 0.40
<i>optdigits</i>	1	33.60 ± 4.64	50.93 ± 4.43	46.83 ± 4.56
	2	31.76 ± 5.85	46.03 ± 3.94	49.03 ± 6.53
	3	23.68 ± 2.32	39.27 ± 4.35	42.45 ± 5.24
	4	20.45 ± 1.72	35.23 ± 3.56	34.24 ± 5.38
	5	18.76 ± 1.19	31.02 ± 3.01	28.79 ± 3.26
	6	17.03 ± 1.37	29.50 ± 2.80	25.61 ± 3.51
	7	16.30 ± 1.41	26.88 ± 2.73	23.10 ± 3.10
	8	15.57 ± 1.41	24.81 ± 2.33	21.46 ± 2.29
	9	14.87 ± 1.87	23.48 ± 2.40	20.41 ± 2.27
	10	14.25 ± 1.81	23.07 ± 2.61	18.69 ± 1.46
	11	13.75 ± 1.78	21.84 ± 2.61	18.12 ± 1.82
	12	13.70 ± 1.90	21.77 ± 2.45	17.51 ± 2.53
	13	13.22 ± 1.56	21.02 ± 2.65	15.94 ± 1.98
	14	13.06 ± 1.87	20.54 ± 2.42	15.16 ± 2.51
	15	12.88 ± 1.75	20.13 ± 2.32	14.40 ± 1.68
	16	12.77 ± 1.56	19.69 ± 1.82	13.68 ± 2.54
	17	12.72 ± 1.74	19.57 ± 1.94	13.49 ± 2.19
	18	13.05 ± 1.68	19.50 ± 2.04	13.20 ± 2.08
	19	12.98 ± 1.79	19.05 ± 2.28	12.83 ± 2.32
	20	12.93 ± 1.71	19.09 ± 2.33	11.98 ± 1.78
	21	12.84 ± 1.80	18.89 ± 2.24	11.58 ± 1.74
	22	12.64 ± 1.64	18.79 ± 2.32	11.16 ± 1.71
	23	12.54 ± 1.76	18.36 ± 2.64	10.23 ± 1.42
	24	12.66 ± 1.81	18.18 ± 2.65	9.91 ± 1.51
	25	12.56 ± 1.76	18.04 ± 2.73	9.75 ± 1.46
	26	12.48 ± 1.72	17.29 ± 2.54	9.50 ± 1.41
	27	12.32 ± 1.71	17.29 ± 2.57	9.52 ± 1.72
	28	12.11 ± 1.77	16.85 ± 2.14	9.16 ± 1.72
	29	12.24 ± 1.69	16.21 ± 2.06	8.83 ± 2.00
	30	12.09 ± 1.59	16.49 ± 1.84	8.83 ± 2.11
	31	12.18 ± 1.58	16.23 ± 1.96	8.54 ± 1.49
<i>segmentation</i>	1	9.31 ± 2.14	22.38 ± 6.05	22.99 ± 7.68
	2	8.10 ± 2.57	19.39 ± 6.62	29.83 ± 16.03
	3	7.88 ± 1.90	17.75 ± 5.14	20.43 ± 7.23
	4	6.54 ± 1.80	15.67 ± 5.41	18.48 ± 7.13
	5	6.71 ± 2.04	16.45 ± 5.97	14.07 ± 2.18
	6	6.36 ± 2.18	14.85 ± 3.99	15.06 ± 6.11
	7	6.19 ± 1.91	15.71 ± 4.87	11.65 ± 2.89
	8	5.93 ± 2.28	14.98 ± 4.14	11.43 ± 3.52
	9	6.19 ± 2.00	13.55 ± 3.13	10.43 ± 2.88
	10	5.89 ± 2.29	13.85 ± 3.20	8.74 ± 1.73
	11	6.23 ± 2.46	14.20 ± 3.33	9.57 ± 2.62

SEM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>segmentation</i>	12	6.32 ± 2.33	13.94 ± 3.26	9.52 ± 2.10
	13	5.80 ± 2.66	13.77 ± 3.64	8.92 ± 2.50
	14	5.67 ± 2.17	13.55 ± 2.47	9.48 ± 2.13
	15	6.06 ± 2.29	13.46 ± 2.66	8.96 ± 2.32
	16	6.15 ± 2.35	13.64 ± 2.66	6.84 ± 1.81
	17	6.10 ± 2.00	13.85 ± 2.74	8.18 ± 2.67
	18	6.06 ± 2.37	13.68 ± 2.59	6.41 ± 1.11
	19	6.06 ± 2.24	13.12 ± 2.68	6.84 ± 1.63
	20	6.19 ± 2.24	12.99 ± 2.55	6.36 ± 1.99
	21	6.06 ± 2.17	12.60 ± 2.29	5.32 ± 1.29
	22	6.02 ± 2.11	12.47 ± 2.08	5.93 ± 1.62
	23	5.76 ± 1.95	12.29 ± 2.18	5.50 ± 1.04
	24	5.58 ± 1.89	12.34 ± 1.84	5.71 ± 1.04
	25	5.41 ± 1.56	12.29 ± 1.98	5.32 ± 1.21
	26	5.45 ± 1.56	12.12 ± 2.05	5.63 ± 1.41
	27	5.41 ± 1.67	11.86 ± 1.73	5.54 ± 1.44
	28	5.50 ± 1.38	11.99 ± 1.76	5.63 ± 1.66
	29	5.58 ± 1.35	11.90 ± 1.91	5.58 ± 1.42
	30	5.45 ± 1.61	11.69 ± 1.90	5.37 ± 1.19
	31	5.41 ± 1.53	11.95 ± 1.75	5.24 ± 1.33
<i>splice</i>	1	0.25 ± 0.35	1.56 ± 1.61	1.63 ± 2.22
	2	0.31 ± 0.36	1.25 ± 1.62	1.78 ± 2.18
	3	0.00 ± 0.00	1.16 ± 1.20	0.25 ± 0.41
	4	0.16 ± 0.22	0.59 ± 0.71	0.28 ± 0.40
	5	0.06 ± 0.13	0.72 ± 0.94	0.22 ± 0.26
	6	0.03 ± 0.10	0.44 ± 0.42	0.13 ± 0.22
	7	0.03 ± 0.10	0.47 ± 0.45	0.13 ± 0.16
	8	0.00 ± 0.00	0.34 ± 0.34	0.06 ± 0.13
	9	0.03 ± 0.10	0.31 ± 0.26	0.09 ± 0.15
	10	0.06 ± 0.13	0.25 ± 0.32	0.13 ± 0.16
	11	0.03 ± 0.10	0.25 ± 0.29	0.06 ± 0.13
	12	0.00 ± 0.00	0.22 ± 0.33	0.09 ± 0.15
	13	0.00 ± 0.00	0.19 ± 0.30	0.13 ± 0.16
	14	0.00 ± 0.00	0.22 ± 0.33	0.09 ± 0.15
	15	0.00 ± 0.00	0.19 ± 0.30	0.06 ± 0.13
	16	0.03 ± 0.10	0.19 ± 0.22	0.09 ± 0.15
	17	0.00 ± 0.00	0.19 ± 0.30	0.06 ± 0.13
	18	0.00 ± 0.00	0.13 ± 0.16	0.13 ± 0.16
	19	0.00 ± 0.00	0.19 ± 0.22	0.06 ± 0.13
	20	0.03 ± 0.10	0.09 ± 0.15	0.06 ± 0.13
	21	0.03 ± 0.10	0.19 ± 0.22	0.06 ± 0.13
	22	0.03 ± 0.10	0.16 ± 0.22	0.06 ± 0.13
	23	0.03 ± 0.10	0.16 ± 0.22	0.06 ± 0.13
	24	0.03 ± 0.10	0.19 ± 0.22	0.06 ± 0.13

SEM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>splice</i>	25	0.03 ± 0.10	0.16 ± 0.22	0.06 ± 0.13
	26	0.03 ± 0.10	0.09 ± 0.21	0.06 ± 0.13
	27	0.03 ± 0.10	0.13 ± 0.22	0.06 ± 0.13
	28	0.03 ± 0.10	0.06 ± 0.13	0.03 ± 0.10
	29	0.03 ± 0.10	0.13 ± 0.22	0.03 ± 0.10
	30	0.03 ± 0.10	0.06 ± 0.13	0.03 ± 0.10
	31	0.03 ± 0.10	0.13 ± 0.22	0.03 ± 0.10
<i>waveform</i>	1	27.16 ± 2.16	29.12 ± 3.02	28.76 ± 3.63
	2	26.58 ± 2.97	27.98 ± 2.68	28.92 ± 3.96
	3	22.64 ± 2.12	23.42 ± 3.13	23.22 ± 2.23
	4	21.82 ± 1.63	23.74 ± 2.80	21.84 ± 1.72
	5	20.72 ± 1.83	22.88 ± 2.79	20.68 ± 1.08
	6	20.58 ± 2.12	22.36 ± 2.90	20.30 ± 1.19
	7	19.90 ± 2.23	21.46 ± 2.65	20.62 ± 1.64
	8	20.30 ± 1.31	20.98 ± 2.18	19.88 ± 1.10
	9	19.38 ± 1.17	19.90 ± 2.07	20.20 ± 1.56
	10	19.46 ± 1.35	20.30 ± 2.27	19.78 ± 1.45
	11	19.44 ± 1.64	19.92 ± 1.77	19.96 ± 1.37
	12	19.02 ± 1.60	19.74 ± 1.59	18.98 ± 1.37
	13	18.52 ± 1.55	19.40 ± 1.77	19.30 ± 1.58
	14	18.58 ± 1.58	19.38 ± 1.81	18.74 ± 1.22
	15	18.34 ± 1.28	18.98 ± 1.45	19.52 ± 1.01
	16	18.40 ± 1.18	18.76 ± 1.71	19.22 ± 1.12
	17	18.04 ± 1.37	18.68 ± 1.65	19.28 ± 1.17
	18	18.28 ± 1.38	18.50 ± 1.71	18.80 ± 1.07
	19	17.76 ± 1.41	18.48 ± 1.45	19.28 ± 1.48
	20	17.88 ± 1.29	18.62 ± 1.77	18.48 ± 1.15
	21	17.60 ± 1.27	18.56 ± 1.63	19.04 ± 1.27
	22	17.86 ± 1.16	18.14 ± 2.00	18.58 ± 1.05
	23	17.60 ± 1.38	18.40 ± 1.10	18.92 ± 1.33
	24	17.82 ± 1.23	18.30 ± 1.51	18.70 ± 1.46
	25	18.12 ± 1.43	18.10 ± 1.57	18.88 ± 1.32
	26	17.76 ± 1.53	18.10 ± 1.68	18.94 ± 1.26
	27	17.80 ± 1.50	18.28 ± 1.53	18.68 ± 1.59
	28	17.92 ± 1.39	18.08 ± 1.63	18.74 ± 1.18
	29	17.72 ± 1.50	18.08 ± 1.31	19.10 ± 1.15
	30	17.66 ± 1.77	17.90 ± 1.57	18.84 ± 0.81
	31	17.68 ± 1.65	18.18 ± 1.44	19.00 ± 0.94

## A.1.2 Taxas de Erro sobre o Conjunto de Treinamento

SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>abalone</i>	1	75.03 ± 2.60	75.39 ± 1.31	75.78 ± 2.07
	2	75.25 ± 2.89	74.92 ± 0.88	76.42 ± 1.73
	3	73.69 ± 0.80	74.50 ± 1.18	76.20 ± 2.19
	4	73.37 ± 0.85	74.59 ± 0.79	75.95 ± 2.12
	5	73.27 ± 0.93	74.39 ± 1.16	75.68 ± 1.63
	6	73.24 ± 0.72	74.33 ± 0.98	75.53 ± 1.75
	7	73.29 ± 0.74	74.46 ± 0.89	75.52 ± 1.39
	8	73.21 ± 0.78	74.05 ± 0.76	75.33 ± 1.34
	9	73.24 ± 0.89	74.16 ± 0.79	75.07 ± 1.27
	10	73.20 ± 0.95	73.82 ± 0.96	74.99 ± 1.24
	11	73.10 ± 0.86	73.91 ± 0.88	75.45 ± 1.22
	12	73.15 ± 0.91	74.08 ± 0.85	75.34 ± 1.49
	13	73.16 ± 0.84	74.01 ± 0.81	75.51 ± 1.16
	14	73.08 ± 0.82	74.14 ± 0.81	75.63 ± 1.07
	15	73.05 ± 0.85	74.13 ± 1.02	75.78 ± 1.25
	16	73.15 ± 0.78	74.06 ± 0.68	75.57 ± 1.18
	17	73.05 ± 0.89	74.17 ± 0.45	76.19 ± 1.22
	18	73.06 ± 0.84	74.01 ± 0.67	76.28 ± 1.49
	19	73.02 ± 0.83	74.14 ± 0.64	76.07 ± 1.50
	20	73.07 ± 0.85	74.08 ± 0.63	75.96 ± 1.44
	21	73.07 ± 0.87	74.09 ± 0.52	76.07 ± 1.47
	22	73.08 ± 0.90	74.03 ± 0.54	76.01 ± 1.14
	23	73.06 ± 0.84	74.01 ± 0.55	76.14 ± 0.99
	24	73.07 ± 0.89	73.91 ± 0.62	75.88 ± 0.90
	25	73.03 ± 0.89	74.03 ± 0.56	75.81 ± 0.79
	26	73.04 ± 0.89	73.94 ± 0.51	75.86 ± 0.91
	27	73.04 ± 0.88	73.97 ± 0.52	75.66 ± 0.89
	28	73.09 ± 0.85	73.95 ± 0.45	75.91 ± 1.40
	29	73.06 ± 0.88	73.93 ± 0.45	75.92 ± 1.21
	30	73.07 ± 0.91	73.86 ± 0.40	75.55 ± 1.10
	31	73.02 ± 0.89	73.94 ± 0.37	75.60 ± 1.07
<i>allhypo</i>	1	0.94 ± 0.56	5.16 ± 0.66	4.99 ± 1.05
	2	0.83 ± 0.41	4.34 ± 0.84	4.99 ± 1.05
	3	0.83 ± 0.40	4.99 ± 0.94	4.37 ± 1.77
	4	0.67 ± 0.37	4.55 ± 1.06	5.10 ± 3.12
	5	0.57 ± 0.25	4.92 ± 0.87	4.23 ± 2.69
	6	0.51 ± 0.14	4.81 ± 0.97	3.38 ± 2.88
	7	0.54 ± 0.15	5.31 ± 0.65	3.16 ± 2.10
	8	0.50 ± 0.18	5.03 ± 1.01	3.29 ± 2.89

## SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC (%)	Bagging (%)	Boosting (%)	
<i>alldo</i>	9	0.49 ± 0.12	5.09 ± 0.93	3.34 ± 3.46	
	10	0.49 ± 0.11	5.12 ± 0.93	2.23 ± 1.47	
	11	0.47 ± 0.12	5.09 ± 0.97	2.87 ± 1.97	
	12	0.45 ± 0.12	5.10 ± 0.88	2.29 ± 1.41	
	13	0.47 ± 0.11	5.15 ± 0.81	3.68 ± 3.49	
	14	0.44 ± 0.10	5.14 ± 0.83	2.14 ± 1.25	
	15	0.45 ± 0.12	5.03 ± 0.78	2.68 ± 2.09	
	16	0.43 ± 0.10	4.91 ± 0.95	2.19 ± 1.56	
	17	0.41 ± 0.10	5.06 ± 0.76	2.62 ± 2.25	
	18	0.42 ± 0.12	4.98 ± 0.80	1.95 ± 1.15	
	19	0.44 ± 0.12	5.09 ± 0.66	2.03 ± 1.56	
	20	0.44 ± 0.12	5.02 ± 0.74	2.01 ± 1.03	
	21	0.43 ± 0.14	5.06 ± 0.65	1.69 ± 0.97	
	22	0.43 ± 0.14	5.10 ± 0.71	1.66 ± 0.98	
	23	0.41 ± 0.12	5.11 ± 0.71	1.78 ± 1.48	
	24	0.40 ± 0.11	5.06 ± 0.77	1.51 ± 1.07	
	25	0.42 ± 0.12	5.05 ± 0.81	1.40 ± 0.89	
	26	0.40 ± 0.11	5.06 ± 0.80	1.56 ± 1.32	
	27	0.40 ± 0.12	5.14 ± 0.65	1.05 ± 0.79	
	28	0.40 ± 0.11	5.15 ± 0.63	1.17 ± 0.93	
	29	0.41 ± 0.14	5.13 ± 0.59	1.27 ± 1.38	
	30	0.39 ± 0.13	5.13 ± 0.68	1.04 ± 0.94	
	31	0.40 ± 0.13	5.15 ± 0.71	1.05 ± 1.07	
	<i>arcene</i>	1	16.33 ± 4.24	21.67 ± 3.82	17.17 ± 2.86
		2	15.33 ± 3.51	20.34 ± 2.71	17.83 ± 2.75
		3	13.49 ± 3.11	18.28 ± 3.20	6.67 ± 2.37
		4	11.44 ± 1.80	17.50 ± 3.16	6.67 ± 2.61
		5	11.33 ± 1.85	15.77 ± 3.66	1.56 ± 0.90
		6	9.67 ± 1.80	15.38 ± 3.96	1.72 ± 1.52
		7	10.00 ± 1.92	15.05 ± 3.91	0.06 ± 0.18
		8	9.39 ± 1.90	14.22 ± 3.82	0.17 ± 0.27
9		9.44 ± 1.83	13.83 ± 3.69	0.00 ± 0.00	
10		9.28 ± 2.54	13.66 ± 3.77	0.00 ± 0.00	
11		8.94 ± 1.74	14.44 ± 3.61	0.00 ± 0.00	
12		9.00 ± 1.79	14.66 ± 3.21	0.00 ± 0.00	
13		9.28 ± 1.55	14.72 ± 3.38	0.00 ± 0.00	
14		8.72 ± 1.83	13.72 ± 2.48	0.00 ± 0.00	
15		8.72 ± 1.59	14.16 ± 3.40	0.00 ± 0.00	
16		8.72 ± 1.86	13.89 ± 2.64	0.00 ± 0.00	
17		8.56 ± 1.31	13.66 ± 3.24	0.00 ± 0.00	
18		8.28 ± 1.44	13.39 ± 3.03	0.00 ± 0.00	
19		8.39 ± 1.55	13.78 ± 3.19	0.00 ± 0.00	
20		8.33 ± 1.95	13.72 ± 2.83	0.00 ± 0.00	
21		8.28 ± 1.92	14.11 ± 3.44	0.00 ± 0.00	

## SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>arcene</i>	22	8.44 ± 1.99	14.17 ± 3.36	0.00 ± 0.00
	23	8.00 ± 2.13	13.89 ± 3.71	0.00 ± 0.00
	24	8.05 ± 1.56	13.39 ± 3.56	0.00 ± 0.00
	25	7.94 ± 1.59	13.55 ± 3.56	0.00 ± 0.00
	26	8.11 ± 1.75	13.39 ± 3.50	0.00 ± 0.00
	27	8.22 ± 1.80	13.77 ± 3.44	0.00 ± 0.00
	28	8.00 ± 1.79	12.77 ± 3.87	0.00 ± 0.00
	29	7.55 ± 1.94	13.39 ± 3.63	0.00 ± 0.00
	30	7.83 ± 1.84	13.11 ± 3.70	0.00 ± 0.00
	31	7.50 ± 1.98	13.22 ± 3.49	0.00 ± 0.00
	<i>geochemical</i>	1	1.21 ± 1.33	7.13 ± 6.36
2		0.74 ± 0.97	1.68 ± 1.49	3.17 ± 5.53
3		0.67 ± 1.27	1.55 ± 1.18	2.09 ± 5.74
4		0.47 ± 0.55	1.08 ± 1.01	2.09 ± 5.74
5		0.60 ± 0.74	1.41 ± 1.02	2.09 ± 5.74
6		0.27 ± 0.35	1.21 ± 0.82	2.09 ± 5.74
7		0.47 ± 0.64	1.34 ± 0.89	2.09 ± 5.74
8		0.47 ± 0.55	1.28 ± 1.02	2.09 ± 5.74
9		0.47 ± 0.55	1.88 ± 1.54	2.09 ± 5.74
10		0.47 ± 0.55	1.14 ± 0.90	2.09 ± 5.74
11		0.60 ± 0.67	1.75 ± 1.62	2.09 ± 5.74
12		0.47 ± 0.55	1.34 ± 1.48	2.09 ± 5.74
13		0.47 ± 0.55	1.48 ± 1.58	2.09 ± 5.74
14		0.60 ± 0.67	0.94 ± 0.96	2.09 ± 5.74
15		0.54 ± 0.62	1.01 ± 1.01	2.09 ± 5.74
16		0.54 ± 0.62	0.67 ± 0.84	2.09 ± 5.74
17		0.54 ± 0.62	0.67 ± 0.84	2.09 ± 5.74
18		0.54 ± 0.62	0.67 ± 0.84	2.09 ± 5.74
19		0.54 ± 0.62	0.74 ± 0.86	2.09 ± 5.74
20		0.47 ± 0.55	0.67 ± 0.84	2.09 ± 5.74
21		0.54 ± 0.62	0.81 ± 0.82	2.09 ± 5.74
22		0.54 ± 0.62	0.81 ± 0.82	2.09 ± 5.74
23		0.47 ± 0.55	0.87 ± 0.84	2.09 ± 5.74
24		0.54 ± 0.62	0.87 ± 1.05	2.09 ± 5.74
25		0.47 ± 0.55	0.87 ± 1.05	2.09 ± 5.74
26		0.47 ± 0.55	0.87 ± 1.05	2.09 ± 5.74
27		0.47 ± 0.55	0.87 ± 1.05	2.09 ± 5.74
28		0.47 ± 0.55	0.87 ± 0.84	2.09 ± 5.74
29		0.40 ± 0.47	0.81 ± 0.88	2.09 ± 5.74
30		0.47 ± 0.55	0.87 ± 0.90	2.09 ± 5.74
31		0.47 ± 0.55	0.87 ± 0.90	2.09 ± 5.74
<i>kr-vs-kp</i>	1	2.75 ± 1.62	5.56 ± 0.43	5.80 ± 1.27
	2	2.70 ± 1.66	5.42 ± 0.44	5.80 ± 1.27
	3	1.94 ± 0.53	5.55 ± 0.32	2.24 ± 1.14

## SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC (%)	Bagging (%)	Boosting (%)	
<i>kr-vs-kp</i>	4	1.64 ± 0.59	5.39 ± 0.40	2.74 ± 0.79	
	5	1.72 ± 0.52	5.61 ± 0.20	0.90 ± 0.82	
	6	1.89 ± 0.38	5.50 ± 0.37	0.92 ± 0.59	
	7	1.87 ± 0.34	5.52 ± 0.34	0.35 ± 0.24	
	8	1.83 ± 0.44	5.51 ± 0.36	0.27 ± 0.22	
	9	1.83 ± 0.42	5.62 ± 0.18	0.15 ± 0.11	
	10	1.69 ± 0.43	5.64 ± 0.13	0.11 ± 0.06	
	11	1.68 ± 0.48	5.63 ± 0.18	0.05 ± 0.04	
	12	1.67 ± 0.33	5.61 ± 0.20	0.05 ± 0.04	
	13	1.69 ± 0.43	5.61 ± 0.22	0.03 ± 0.03	
	14	1.60 ± 0.57	5.61 ± 0.22	0.01 ± 0.02	
	15	1.61 ± 0.52	5.61 ± 0.22	0.01 ± 0.02	
	16	1.52 ± 0.45	5.61 ± 0.22	0.00 ± 0.01	
	17	1.44 ± 0.56	5.61 ± 0.22	0.00 ± 0.01	
	18	1.56 ± 0.48	5.61 ± 0.22	0.00 ± 0.01	
	19	1.61 ± 0.52	5.61 ± 0.22	0.00 ± 0.01	
	20	1.61 ± 0.53	5.61 ± 0.22	0.00 ± 0.00	
	21	1.62 ± 0.53	5.57 ± 0.33	0.00 ± 0.00	
	22	1.61 ± 0.53	5.57 ± 0.32	0.00 ± 0.00	
	23	1.61 ± 0.53	5.57 ± 0.32	0.00 ± 0.00	
	24	1.69 ± 0.38	5.57 ± 0.32	0.00 ± 0.00	
	25	1.68 ± 0.38	5.57 ± 0.32	0.00 ± 0.00	
	26	1.68 ± 0.38	5.57 ± 0.32	0.00 ± 0.00	
	27	1.61 ± 0.49	5.57 ± 0.33	0.00 ± 0.00	
	28	1.73 ± 0.34	5.57 ± 0.33	0.00 ± 0.00	
	29	1.57 ± 0.46	5.57 ± 0.32	0.00 ± 0.00	
	30	1.73 ± 0.34	5.57 ± 0.32	0.00 ± 0.00	
	31	1.69 ± 0.30	5.57 ± 0.32	0.00 ± 0.00	
	<i>optdigits</i>	1	32.59 ± 4.67	50.06 ± 4.12	46.31 ± 3.92
		2	30.50 ± 4.64	46.02 ± 4.26	48.76 ± 6.65
		3	22.77 ± 1.55	38.51 ± 4.33	41.34 ± 5.16
4		19.58 ± 1.24	34.48 ± 3.16	33.14 ± 4.58	
5		17.32 ± 1.37	30.54 ± 3.22	27.74 ± 2.86	
6		16.04 ± 1.33	28.49 ± 2.82	24.39 ± 2.88	
7		15.34 ± 1.31	26.53 ± 3.04	22.25 ± 2.42	
8		14.99 ± 1.43	24.43 ± 2.53	20.37 ± 1.44	
9		14.33 ± 1.66	23.20 ± 3.24	18.92 ± 2.08	
10		13.73 ± 1.53	22.34 ± 3.02	17.27 ± 1.79	
11		13.37 ± 1.52	21.33 ± 2.81	16.84 ± 1.76	
12		13.00 ± 1.35	20.87 ± 2.54	15.74 ± 1.69	
13		12.73 ± 1.27	20.35 ± 2.98	14.31 ± 1.56	
14		12.56 ± 1.30	20.29 ± 2.85	13.24 ± 1.74	
15		12.40 ± 1.16	19.61 ± 2.73	12.43 ± 1.57	
16		12.28 ± 1.23	19.13 ± 2.69	11.73 ± 1.12	

## SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>optdigits</i>	17	12.36 ± 1.22	18.98 ± 2.80	11.27 ± 0.82
	18	12.48 ± 1.32	18.87 ± 3.09	10.55 ± 1.18
	19	12.27 ± 1.23	18.55 ± 3.09	10.20 ± 1.07
	20	12.18 ± 1.16	18.40 ± 2.62	9.55 ± 1.02
	21	12.01 ± 1.17	18.03 ± 2.36	9.09 ± 1.11
	22	11.85 ± 1.16	18.01 ± 1.96	8.59 ± 0.94
	23	11.87 ± 1.09	17.62 ± 2.05	8.19 ± 0.89
	24	11.87 ± 1.08	17.34 ± 1.87	7.68 ± 0.70
	25	11.68 ± 1.27	17.17 ± 1.66	7.50 ± 1.06
	26	11.58 ± 1.24	16.55 ± 1.49	7.14 ± 0.93
	27	11.41 ± 1.07	16.29 ± 1.28	7.00 ± 0.97
	28	11.33 ± 1.11	15.95 ± 1.41	6.71 ± 0.91
	29	11.27 ± 1.06	15.53 ± 1.36	6.49 ± 0.99
	30	11.24 ± 1.17	15.75 ± 1.34	6.15 ± 0.95
	31	11.16 ± 1.13	15.65 ± 1.58	5.94 ± 0.68
<i>segmentation</i>	1	9.22 ± 1.75	21.61 ± 5.74	21.76 ± 7.47
	2	8.07 ± 1.37	18.85 ± 5.98	29.32 ± 16.06
	3	7.26 ± 1.55	16.99 ± 5.72	19.42 ± 6.73
	4	6.46 ± 1.24	15.32 ± 5.20	16.54 ± 7.94
	5	6.18 ± 0.82	15.75 ± 5.63	12.37 ± 2.69
	6	5.78 ± 0.71	14.45 ± 4.60	13.13 ± 5.32
	7	5.60 ± 0.60	14.81 ± 4.96	10.07 ± 2.30
	8	5.36 ± 0.68	14.58 ± 4.13	9.50 ± 2.32
	9	5.42 ± 0.68	13.89 ± 3.89	8.10 ± 2.23
	10	5.36 ± 0.79	13.61 ± 3.42	7.29 ± 1.36
	11	5.35 ± 0.88	14.26 ± 3.63	7.49 ± 1.98
	12	5.31 ± 0.82	13.61 ± 3.77	7.60 ± 1.73
	13	5.22 ± 0.73	13.47 ± 3.87	7.28 ± 2.57
	14	5.06 ± 0.80	12.99 ± 2.98	7.50 ± 2.25
	15	5.18 ± 0.68	12.66 ± 2.87	7.42 ± 2.08
	16	5.09 ± 0.67	12.76 ± 2.94	5.59 ± 1.55
	17	5.11 ± 0.59	12.73 ± 3.05	6.25 ± 2.84
	18	5.08 ± 0.65	12.71 ± 2.95	5.58 ± 2.19
	19	5.07 ± 0.74	12.01 ± 2.93	5.48 ± 1.41
	20	5.14 ± 0.69	12.18 ± 2.94	4.76 ± 1.58
	21	5.12 ± 0.61	11.99 ± 2.66	4.25 ± 1.44
	22	5.11 ± 0.54	11.90 ± 2.57	4.34 ± 1.26
	23	5.01 ± 0.54	11.76 ± 2.53	4.16 ± 1.08
	24	4.98 ± 0.57	11.88 ± 2.65	4.02 ± 1.00
	25	4.87 ± 0.50	11.84 ± 2.71	4.06 ± 0.91
	26	4.92 ± 0.43	11.74 ± 2.73	3.72 ± 0.85
	27	4.79 ± 0.36	11.60 ± 2.54	4.01 ± 1.05
	28	4.77 ± 0.38	11.37 ± 2.44	3.90 ± 0.76
	29	4.79 ± 0.38	11.46 ± 2.42	3.58 ± 0.72

## SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	Bagging (%)	Boosting (%)
<i>segmentation</i>	30	4.75 ± 0.46	11.27 ± 2.45	3.71 ± 0.81
	31	4.72 ± 0.40	11.28 ± 2.47	3.49 ± 1.02
<i>splice</i>	1	0.13 ± 0.05	1.45 ± 1.14	1.65 ± 2.37
	2	0.17 ± 0.11	1.15 ± 1.18	1.68 ± 2.35
	3	0.06 ± 0.09	0.97 ± 0.89	0.25 ± 0.40
	4	0.04 ± 0.03	0.57 ± 0.42	0.15 ± 0.29
	5	0.04 ± 0.03	0.62 ± 0.47	0.03 ± 0.07
	6	0.04 ± 0.03	0.50 ± 0.37	0.02 ± 0.03
	7	0.03 ± 0.03	0.48 ± 0.34	0.01 ± 0.03
	8	0.02 ± 0.02	0.34 ± 0.31	0.00 ± 0.01
	9	0.03 ± 0.03	0.29 ± 0.26	0.00 ± 0.00
	10	0.02 ± 0.02	0.24 ± 0.26	0.00 ± 0.00
	11	0.01 ± 0.02	0.25 ± 0.23	0.00 ± 0.00
	12	0.02 ± 0.02	0.19 ± 0.20	0.00 ± 0.00
	13	0.01 ± 0.02	0.18 ± 0.21	0.00 ± 0.00
	14	0.01 ± 0.02	0.14 ± 0.08	0.00 ± 0.00
	15	0.01 ± 0.02	0.19 ± 0.20	0.00 ± 0.00
	16	0.01 ± 0.02	0.13 ± 0.09	0.00 ± 0.00
	17	0.01 ± 0.01	0.14 ± 0.09	0.00 ± 0.00
	18	0.01 ± 0.01	0.12 ± 0.07	0.00 ± 0.00
	19	0.01 ± 0.01	0.14 ± 0.08	0.00 ± 0.00
	20	0.01 ± 0.01	0.10 ± 0.06	0.00 ± 0.00
	21	0.01 ± 0.01	0.11 ± 0.06	0.00 ± 0.00
	22	0.01 ± 0.02	0.10 ± 0.07	0.00 ± 0.00
	23	0.01 ± 0.02	0.09 ± 0.07	0.00 ± 0.00
	24	0.01 ± 0.01	0.10 ± 0.07	0.00 ± 0.00
	25	0.01 ± 0.01	0.08 ± 0.07	0.00 ± 0.00
	26	0.01 ± 0.01	0.08 ± 0.08	0.00 ± 0.00
	27	0.01 ± 0.01	0.07 ± 0.07	0.00 ± 0.00
	28	0.01 ± 0.01	0.07 ± 0.06	0.00 ± 0.00
	29	0.01 ± 0.02	0.06 ± 0.05	0.00 ± 0.00
	30	0.01 ± 0.01	0.06 ± 0.06	0.00 ± 0.00
	31	0.01 ± 0.02	0.05 ± 0.05	0.00 ± 0.00
<i>waveform</i>	1	25.81 ± 1.33	26.63 ± 1.63	27.69 ± 1.70
	2	24.95 ± 1.93	26.12 ± 1.24	28.56 ± 3.35
	3	21.32 ± 0.60	22.25 ± 1.21	21.24 ± 1.06
	4	20.34 ± 0.90	21.87 ± 1.53	20.70 ± 1.06
	5	19.01 ± 0.66	20.73 ± 1.35	19.04 ± 0.63
	6	18.78 ± 0.79	20.43 ± 1.20	18.73 ± 0.51
	7	18.05 ± 0.64	19.69 ± 1.22	18.10 ± 0.82
	8	17.99 ± 0.76	19.34 ± 1.30	17.82 ± 0.56
	9	17.50 ± 0.83	18.61 ± 1.24	17.44 ± 0.60
	10	17.35 ± 0.38	18.38 ± 1.21	17.13 ± 0.79
	11	16.92 ± 0.41	18.07 ± 1.10	16.85 ± 0.50

SEM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC (%)	<i>Bagging</i> (%)	<i>Boosting</i> (%)
<i>waveform</i>	12	16.89 ± 0.38	18.16 ± 0.98	16.76 ± 0.85
	13	16.66 ± 0.34	17.65 ± 1.08	16.32 ± 0.69
	14	16.69 ± 0.44	17.51 ± 1.09	16.44 ± 0.85
	15	16.46 ± 0.40	17.34 ± 0.97	15.87 ± 0.65
	16	16.51 ± 0.36	17.17 ± 0.82	16.05 ± 0.74
	17	16.25 ± 0.36	16.91 ± 0.84	15.48 ± 0.66
	18	16.22 ± 0.31	16.95 ± 0.81	15.65 ± 0.38
	19	16.11 ± 0.31	16.84 ± 0.78	15.00 ± 0.52
	20	16.23 ± 0.42	16.88 ± 0.65	15.15 ± 0.49
	21	16.03 ± 0.34	16.72 ± 0.62	14.86 ± 0.52
	22	16.10 ± 0.29	16.72 ± 0.62	14.97 ± 0.61
	23	15.97 ± 0.28	16.57 ± 0.64	14.65 ± 0.56
	24	16.08 ± 0.42	16.64 ± 0.53	14.57 ± 0.51
	25	16.01 ± 0.40	16.60 ± 0.51	14.29 ± 0.41
	26	16.01 ± 0.40	16.57 ± 0.54	14.33 ± 0.49
	27	15.99 ± 0.46	16.59 ± 0.47	14.08 ± 0.48
	28	15.95 ± 0.50	16.53 ± 0.54	14.16 ± 0.44
	29	15.86 ± 0.44	16.37 ± 0.60	13.97 ± 0.43
	30	15.93 ± 0.42	16.37 ± 0.57	14.08 ± 0.46
	31	15.76 ± 0.42	16.28 ± 0.55	13.82 ± 0.46

## A.2 Experimentos *Com* Particionamento

### A.2.1 Taxas de Erro sobre o Conjunto de Teste

COM PARTICIONAMENTO — CONJ. DE TESTE

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>abalone</i>	1	76.03 ± 3.01	77.07 ± 1.09	75.41 ± 1.96
	2	76.12 ± 3.26	75.60 ± 1.72	76.14 ± 2.06
	3	74.78 ± 1.19	75.47 ± 1.83	74.79 ± 1.56
	4	74.80 ± 1.26	75.00 ± 1.58	75.13 ± 2.10
	5	74.66 ± 1.15	75.46 ± 2.43	74.72 ± 1.87
	6	74.11 ± 1.19	74.93 ± 2.13	74.90 ± 1.48
	7	73.99 ± 1.13	74.95 ± 1.81	75.26 ± 1.85
	8	74.01 ± 1.30	75.34 ± 1.71	75.43 ± 1.49
	9	74.16 ± 1.58	75.07 ± 1.99	73.66 ± 1.88
	10	73.97 ± 0.89	75.16 ± 1.91	74.69 ± 1.62
	11	74.21 ± 1.03	75.08 ± 1.89	75.31 ± 2.07
	12	74.23 ± 1.18	75.29 ± 1.46	75.60 ± 1.02

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>abalone</i>	13	74.20 ± 1.18	74.99 ± 1.67	74.86 ± 1.55
	14	74.11 ± 1.06	74.77 ± 1.92	75.06 ± 2.53
	15	74.28 ± 0.97	75.49 ± 1.40	74.24 ± 1.76
	16	74.19 ± 1.07	74.77 ± 1.89	74.77 ± 2.41
	17	74.50 ± 0.93	74.75 ± 1.58	74.70 ± 2.33
	18	74.13 ± 1.21	75.17 ± 2.04	74.88 ± 1.36
	19	74.30 ± 1.18	74.68 ± 1.07	74.65 ± 1.64
	20	74.30 ± 1.25	76.05 ± 1.31	74.70 ± 2.12
	21	74.40 ± 1.30	75.04 ± 1.97	74.86 ± 1.79
	22	74.30 ± 1.17	75.02 ± 1.88	74.50 ± 1.96
	23	74.14 ± 1.22	75.11 ± 1.40	74.89 ± 2.68
	24	73.97 ± 1.40	74.76 ± 1.41	74.43 ± 2.42
	25	74.30 ± 1.69	75.16 ± 1.61	74.85 ± 1.62
	26	74.15 ± 1.67	74.77 ± 1.03	75.10 ± 1.69
	27	74.16 ± 1.59	75.57 ± 2.34	74.11 ± 2.15
	28	74.23 ± 1.62	75.04 ± 2.09	75.28 ± 2.11
	29	74.18 ± 1.39	75.70 ± 1.54	74.86 ± 1.74
	30	74.18 ± 1.36	75.84 ± 1.24	74.31 ± 1.98
	31	74.21 ± 1.48	74.40 ± 1.71	74.31 ± 1.65
<i>allhypo</i>	1	1.28 ± 0.96	1.65 ± 0.75	1.40 ± 0.65
	2	1.11 ± 0.85	1.73 ± 1.31	2.23 ± 1.46
	3	1.32 ± 0.77	2.14 ± 0.66	1.49 ± 1.00
	4	1.07 ± 0.85	1.90 ± 0.67	1.61 ± 0.84
	5	0.91 ± 0.79	2.47 ± 1.07	1.73 ± 1.26
	6	0.82 ± 0.33	2.52 ± 1.02	2.02 ± 0.80
	7	0.87 ± 0.41	2.06 ± 0.79	2.81 ± 2.15
	8	0.82 ± 0.43	2.27 ± 0.68	2.02 ± 1.00
	9	0.78 ± 0.30	2.77 ± 0.94	2.97 ± 1.29
	10	0.70 ± 0.58	2.64 ± 1.36	2.14 ± 0.84
	11	0.82 ± 0.51	2.68 ± 1.56	2.39 ± 1.10
	12	0.78 ± 0.56	3.06 ± 1.62	3.01 ± 1.61
	13	0.78 ± 0.49	3.10 ± 1.46	3.18 ± 1.65
	14	0.74 ± 0.61	3.26 ± 1.82	2.93 ± 1.27
	15	0.74 ± 0.61	2.84 ± 1.45	2.93 ± 1.19
	16	0.70 ± 0.58	3.01 ± 1.17	3.06 ± 1.57
	17	0.70 ± 0.58	4.33 ± 1.58	4.33 ± 1.87
	18	0.70 ± 0.48	4.04 ± 1.29	4.00 ± 1.97
	19	0.74 ± 0.54	3.75 ± 1.26	4.82 ± 1.60
	20	0.74 ± 0.47	4.46 ± 1.54	4.58 ± 2.02
	21	0.74 ± 0.54	4.33 ± 1.51	4.58 ± 1.45
	22	0.78 ± 0.63	4.46 ± 0.85	5.82 ± 1.22
	23	0.70 ± 0.52	4.70 ± 2.05	5.28 ± 1.28
	24	0.70 ± 0.52	5.40 ± 1.37	5.86 ± 1.44
	25	0.70 ± 0.52	5.52 ± 1.70	5.03 ± 2.00

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>allhypo</i>	26	0.70 ± 0.52	5.78 ± 0.88	5.57 ± 1.17
	27	0.74 ± 0.54	5.57 ± 1.46	5.69 ± 1.42
	28	0.74 ± 0.54	5.98 ± 1.48	6.27 ± 0.89
	29	0.74 ± 0.54	5.48 ± 1.60	6.19 ± 1.15
	30	0.74 ± 0.54	6.44 ± 0.52	6.31 ± 0.88
	31	0.74 ± 0.54	6.44 ± 0.47	6.40 ± 0.67
<i>arcene</i>	1	29.28 ± 12.86	33.93 ± 12.89	32.33 ± 10.98
	2	29.78 ± 11.87	38.01 ± 12.06	32.41 ± 8.79
	3	24.38 ± 12.47	34.03 ± 8.45	33.01 ± 8.03
	4	25.47 ± 12.04	36.47 ± 12.07	32.28 ± 10.28
	5	21.45 ± 11.35	41.89 ± 8.93	30.38 ± 10.11
	6	22.47 ± 8.77	35.26 ± 14.67	35.11 ± 7.44
	7	21.93 ± 10.30	32.56 ± 8.79	31.98 ± 7.40
	8	21.90 ± 8.92	38.02 ± 7.59	34.94 ± 11.82
	9	20.92 ± 7.44	32.44 ± 10.10	34.06 ± 9.38
	10	21.40 ± 8.09	38.49 ± 8.02	33.01 ± 8.23
	11	18.87 ± 7.96	36.84 ± 12.80	34.01 ± 7.74
	12	21.37 ± 7.65	34.38 ± 8.03	35.94 ± 11.39
	13	19.42 ± 6.51	40.14 ± 15.01	35.39 ± 10.35
	14	18.92 ± 6.19	39.43 ± 11.75	35.84 ± 8.51
	15	15.40 ± 6.76	35.93 ± 10.07	38.11 ± 12.10
	16	17.37 ± 6.57	37.54 ± 11.44	33.51 ± 4.74
	17	18.37 ± 6.92	40.44 ± 10.66	31.98 ± 5.66
	18	19.35 ± 6.93	34.98 ± 6.95	36.43 ± 9.41
	19	19.40 ± 6.73	39.51 ± 7.61	34.91 ± 9.57
	20	18.92 ± 5.22	36.84 ± 8.52	34.29 ± 12.78
	21	16.52 ± 5.24	36.84 ± 14.64	34.89 ± 9.41
	22	18.50 ± 3.25	39.94 ± 9.47	34.02 ± 13.33
	23	16.54 ± 4.18	37.01 ± 13.40	37.36 ± 10.90
	24	17.99 ± 5.65	40.89 ± 14.12	37.51 ± 10.76
	25	15.52 ± 4.91	38.31 ± 11.63	32.61 ± 13.97
	26	18.07 ± 6.48	36.48 ± 9.01	39.57 ± 8.26
	27	19.05 ± 6.27	38.41 ± 11.42	34.58 ± 10.36
	28	18.55 ± 5.94	39.39 ± 10.84	34.41 ± 9.18
	29	17.54 ± 5.98	40.92 ± 8.74	34.41 ± 7.00
	30	17.44 ± 5.12	36.89 ± 12.60	34.41 ± 8.48
	31	19.45 ± 5.72	33.99 ± 6.80	35.88 ± 8.57
<i>geochemical</i>	1	4.16 ± 4.78	3.02 ± 3.19	1.77 ± 2.85
	2	3.54 ± 4.89	1.80 ± 2.90	2.36 ± 3.05
	3	3.54 ± 5.71	3.61 ± 3.11	3.53 ± 5.69
	4	2.29 ± 4.83	3.64 ± 5.74	4.09 ± 5.50
	5	2.91 ± 4.90	4.27 ± 7.80	4.23 ± 7.79
	6	2.91 ± 4.90	4.79 ± 5.54	5.89 ± 7.79
	7	2.91 ± 4.90	2.95 ± 4.17	6.18 ± 7.80

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)	
<i>geochemical</i>	8	1.73 ± 3.93	7.21 ± 8.89	9.72 ± 10.56	
	9	2.36 ± 4.12	12.00 ± 6.16	10.76 ± 8.89	
	10	1.73 ± 3.93	13.98 ± 11.36	16.11 ± 17.07	
	11	2.36 ± 4.12	18.35 ± 14.09	11.99 ± 12.75	
	12	1.73 ± 3.93	20.61 ± 6.27	15.43 ± 13.34	
	13	1.73 ± 3.93	24.69 ± 8.75	18.76 ± 11.00	
	14	1.73 ± 3.93	26.53 ± 11.30	17.51 ± 10.03	
	15	1.73 ± 3.93	27.26 ± 8.77	18.10 ± 6.76	
	16	1.73 ± 3.93	29.73 ± 8.41	26.08 ± 6.32	
	17	1.73 ± 3.93	26.61 ± 12.89	19.97 ± 8.41	
	18	1.73 ± 3.93	25.87 ± 11.37	29.65 ± 9.16	
	19	1.73 ± 3.93	30.60 ± 11.92	27.23 ± 4.71	
	20	1.73 ± 3.93	25.50 ± 8.75	24.84 ± 4.23	
	21	1.73 ± 3.93	28.52 ± 8.03	26.05 ± 5.54	
	22	1.73 ± 3.93	30.94 ± 6.84	25.84 ± 10.42	
	23	1.73 ± 3.93	30.14 ± 8.81	32.60 ± 11.35	
	24	1.73 ± 3.93	35.20 ± 8.47	29.65 ± 8.50	
	25	1.73 ± 3.93	33.99 ± 8.44	29.06 ± 7.42	
	26	1.73 ± 3.93	30.28 ± 6.73	25.46 ± 7.57	
	27	1.73 ± 3.93	30.31 ± 7.35	26.50 ± 8.05	
	28	1.73 ± 3.93	32.68 ± 7.04	25.29 ± 9.56	
	29	1.73 ± 3.93	33.82 ± 9.07	32.50 ± 12.06	
	30	1.73 ± 3.93	33.30 ± 7.82	32.23 ± 8.77	
	31	1.73 ± 3.93	36.35 ± 6.88	31.54 ± 13.56	
	<i>kr-vs-kp</i>	1	3.44 ± 2.31	1.94 ± 0.75	2.38 ± 0.83
		2	3.13 ± 2.40	2.32 ± 1.72	2.35 ± 1.05
		3	2.13 ± 0.86	2.13 ± 1.08	2.07 ± 0.75
		4	1.88 ± 1.03	1.63 ± 0.94	2.16 ± 1.82
		5	1.97 ± 1.07	1.91 ± 1.10	2.03 ± 1.07
		6	2.10 ± 0.74	1.82 ± 0.98	1.85 ± 0.94
		7	1.97 ± 0.77	1.91 ± 0.70	1.97 ± 0.65
8		1.91 ± 0.85	1.75 ± 0.66	2.00 ± 1.05	
9		1.94 ± 0.83	2.00 ± 1.05	1.94 ± 0.72	
10		1.75 ± 0.80	1.85 ± 0.84	2.63 ± 0.75	
11		1.85 ± 0.88	2.38 ± 0.77	2.53 ± 0.90	
12		1.69 ± 0.69	2.50 ± 1.25	2.38 ± 1.03	
13		1.78 ± 0.79	2.85 ± 1.13	3.00 ± 1.23	
14		1.63 ± 0.92	2.69 ± 1.18	2.78 ± 0.84	
15		1.63 ± 0.92	2.88 ± 1.01	2.82 ± 1.06	
16		1.56 ± 0.87	3.00 ± 0.90	2.91 ± 0.83	
17		1.44 ± 0.96	2.47 ± 0.94	3.25 ± 0.88	
18		1.56 ± 0.82	2.91 ± 0.64	3.38 ± 0.91	
19		1.60 ± 0.96	3.29 ± 0.90	3.07 ± 0.86	
20		1.60 ± 0.96	3.38 ± 1.07	3.16 ± 0.81	

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>kr-vs-kp</i>	21	1.60 ± 0.96	3.66 ± 1.64	3.66 ± 1.15
	22	1.56 ± 1.01	3.69 ± 1.00	3.54 ± 0.96
	23	1.56 ± 1.01	3.72 ± 1.46	3.57 ± 1.28
	24	1.69 ± 0.91	3.94 ± 1.03	3.79 ± 1.17
	25	1.69 ± 0.91	4.19 ± 1.38	3.72 ± 0.96
	26	1.69 ± 0.91	4.13 ± 0.88	3.75 ± 1.09
	27	1.56 ± 1.05	3.57 ± 1.37	4.32 ± 1.03
	28	1.69 ± 0.91	3.79 ± 1.16	3.82 ± 1.22
	29	1.50 ± 0.96	4.10 ± 1.13	4.22 ± 1.15
	30	1.69 ± 0.91	3.82 ± 1.48	3.88 ± 0.96
	31	1.63 ± 0.82	4.22 ± 1.02	3.91 ± 0.94
<i>optdigits</i>	1	33.60 ± 4.64	33.49 ± 3.13	31.60 ± 2.19
	2	31.76 ± 5.85	28.64 ± 1.84	30.06 ± 3.17
	3	23.68 ± 2.32	28.18 ± 3.27	27.12 ± 2.43
	4	20.45 ± 1.72	23.12 ± 2.80	23.87 ± 2.89
	5	18.76 ± 1.19	24.52 ± 3.21	21.64 ± 3.44
	6	17.03 ± 1.37	21.47 ± 3.00	21.52 ± 2.15
	7	16.30 ± 1.41	19.82 ± 3.10	22.16 ± 3.67
	8	15.57 ± 1.41	20.55 ± 1.97	19.53 ± 3.05
	9	14.87 ± 1.87	18.47 ± 3.78	19.96 ± 3.49
	10	14.25 ± 1.81	19.05 ± 2.07	17.61 ± 2.31
	11	13.75 ± 1.78	17.01 ± 2.72	18.15 ± 2.62
	12	13.70 ± 1.90	16.03 ± 3.01	17.67 ± 2.57
	13	13.22 ± 1.56	16.38 ± 2.12	16.07 ± 1.96
	14	13.06 ± 1.87	16.44 ± 2.65	14.84 ± 1.71
	15	12.88 ± 1.75	15.82 ± 1.80	15.40 ± 2.14
	16	12.77 ± 1.56	15.32 ± 2.25	15.62 ± 1.49
	17	12.72 ± 1.74	15.08 ± 1.84	14.54 ± 0.61
	18	13.05 ± 1.68	14.88 ± 2.28	14.98 ± 2.60
	19	12.98 ± 1.79	14.05 ± 2.02	13.70 ± 1.88
	20	12.93 ± 1.71	14.18 ± 1.95	13.51 ± 1.96
	21	12.84 ± 1.80	13.91 ± 2.39	14.30 ± 1.81
	22	12.64 ± 1.64	13.95 ± 2.08	13.58 ± 2.08
	23	12.54 ± 1.76	14.92 ± 3.13	13.29 ± 1.95
	24	12.66 ± 1.81	14.34 ± 1.80	13.01 ± 1.85
	25	12.56 ± 1.76	13.12 ± 1.93	12.45 ± 1.69
	26	12.48 ± 1.72	13.86 ± 2.10	12.86 ± 1.78
	27	12.32 ± 1.71	12.40 ± 2.16	12.17 ± 1.39
	28	12.11 ± 1.77	12.38 ± 1.86	12.79 ± 2.13
	29	12.24 ± 1.69	13.54 ± 2.02	12.38 ± 1.37
	30	12.09 ± 1.59	12.42 ± 1.98	12.44 ± 1.84
	31	12.18 ± 1.58	12.78 ± 2.04	12.22 ± 1.63
<i>segmentation</i>	1	9.31 ± 2.14	9.00 ± 3.40	9.70 ± 1.70
	2	8.10 ± 2.57	8.53 ± 2.53	10.09 ± 3.56

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)	
<i>segmentation</i>	3	7.88 ± 1.90	8.74 ± 2.37	9.48 ± 3.19	
	4	6.54 ± 1.80	7.97 ± 2.36	7.79 ± 1.67	
	5	6.71 ± 2.04	7.88 ± 1.62	8.87 ± 1.99	
	6	6.36 ± 2.18	9.00 ± 2.67	8.23 ± 1.35	
	7	6.19 ± 1.91	8.70 ± 1.81	8.01 ± 2.60	
	8	5.93 ± 2.28	7.58 ± 1.91	8.27 ± 2.57	
	9	6.19 ± 2.00	8.44 ± 3.23	7.23 ± 1.68	
	10	5.89 ± 2.29	9.00 ± 1.87	8.31 ± 2.40	
	11	6.23 ± 2.46	7.58 ± 1.78	8.79 ± 2.12	
	12	6.32 ± 2.33	8.57 ± 1.85	8.83 ± 2.35	
	13	5.80 ± 2.66	8.61 ± 2.19	9.00 ± 2.88	
	14	5.67 ± 2.17	8.27 ± 2.29	8.18 ± 2.04	
	15	6.06 ± 2.29	9.09 ± 1.58	8.35 ± 1.98	
	16	6.15 ± 2.35	9.00 ± 1.99	8.79 ± 1.76	
	17	6.10 ± 2.00	9.87 ± 2.17	8.70 ± 1.86	
	18	6.06 ± 2.37	9.35 ± 2.29	9.52 ± 2.53	
	19	6.06 ± 2.24	8.92 ± 1.19	8.57 ± 1.35	
	20	6.19 ± 2.24	9.05 ± 1.91	9.05 ± 1.86	
	21	6.06 ± 2.17	9.26 ± 2.34	9.13 ± 1.93	
	22	6.02 ± 2.11	9.09 ± 2.10	8.87 ± 2.01	
	23	5.76 ± 1.95	9.26 ± 2.67	10.09 ± 1.34	
	24	5.58 ± 1.89	9.00 ± 2.29	9.26 ± 1.76	
	25	5.41 ± 1.56	10.13 ± 1.52	9.18 ± 1.60	
	26	5.45 ± 1.56	10.00 ± 1.36	9.13 ± 2.39	
	27	5.41 ± 1.67	9.44 ± 2.40	9.13 ± 1.45	
	28	5.50 ± 1.38	9.57 ± 1.35	9.74 ± 2.27	
	29	5.58 ± 1.35	9.35 ± 2.46	8.57 ± 0.93	
	30	5.45 ± 1.61	9.96 ± 1.31	9.87 ± 1.76	
	31	5.41 ± 1.53	9.87 ± 1.54	9.87 ± 0.91	
	<i>splice</i>	1	0.25 ± 0.35	0.31 ± 0.29	0.19 ± 0.22
		2	0.31 ± 0.36	0.22 ± 0.26	0.19 ± 0.26
3		0.00 ± 0.00	0.38 ± 0.32	0.25 ± 0.32	
4		0.16 ± 0.22	0.34 ± 0.27	0.25 ± 0.25	
5		0.06 ± 0.13	0.25 ± 0.29	0.22 ± 0.33	
6		0.03 ± 0.10	0.22 ± 0.21	0.16 ± 0.17	
7		0.03 ± 0.10	0.16 ± 0.22	0.16 ± 0.34	
8		0.00 ± 0.00	0.41 ± 0.26	0.16 ± 0.22	
9		0.03 ± 0.10	0.25 ± 0.32	0.31 ± 0.33	
10		0.06 ± 0.13	0.31 ± 0.39	0.09 ± 0.15	
11		0.03 ± 0.10	0.13 ± 0.22	0.34 ± 0.31	
12		0.00 ± 0.00	0.16 ± 0.17	0.06 ± 0.13	
13		0.00 ± 0.00	0.25 ± 0.32	0.19 ± 0.30	
14		0.00 ± 0.00	0.25 ± 0.25	0.28 ± 0.27	
15		0.00 ± 0.00	0.28 ± 0.34	0.41 ± 0.53	

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>splice</i>	16	0.03 ± 0.10	0.31 ± 0.26	0.19 ± 0.26
	17	0.00 ± 0.00	0.25 ± 0.29	0.19 ± 0.22
	18	0.00 ± 0.00	0.16 ± 0.17	0.28 ± 0.31
	19	0.00 ± 0.00	0.25 ± 0.29	0.09 ± 0.15
	20	0.03 ± 0.10	0.28 ± 0.23	0.22 ± 0.26
	21	0.03 ± 0.10	0.16 ± 0.30	0.13 ± 0.16
	22	0.03 ± 0.10	0.13 ± 0.22	0.28 ± 0.35
	23	0.03 ± 0.10	0.19 ± 0.22	0.34 ± 0.40
	24	0.03 ± 0.10	0.22 ± 0.15	0.38 ± 0.38
	25	0.03 ± 0.10	0.22 ± 0.33	0.19 ± 0.22
	26	0.03 ± 0.10	0.16 ± 0.27	0.19 ± 0.22
	27	0.03 ± 0.10	0.19 ± 0.22	0.28 ± 0.37
	28	0.03 ± 0.10	0.22 ± 0.21	0.31 ± 0.26
	29	0.03 ± 0.10	0.16 ± 0.22	0.19 ± 0.26
	30	0.03 ± 0.10	0.22 ± 0.33	0.22 ± 0.26
	31	0.03 ± 0.10	0.22 ± 0.26	0.16 ± 0.22
<i>waveform</i>	1	27.16 ± 2.16	25.90 ± 3.18	27.00 ± 2.12
	2	26.58 ± 2.97	26.50 ± 2.71	26.58 ± 2.42
	3	22.64 ± 2.12	22.40 ± 2.17	22.32 ± 2.41
	4	21.82 ± 1.63	22.70 ± 2.14	22.84 ± 1.75
	5	20.72 ± 1.83	20.22 ± 1.96	21.26 ± 1.58
	6	20.58 ± 2.12	20.46 ± 1.70	20.68 ± 2.11
	7	19.90 ± 2.23	20.42 ± 1.61	20.06 ± 1.53
	8	20.30 ± 1.31	20.02 ± 2.17	19.92 ± 1.08
	9	19.38 ± 1.17	19.28 ± 1.74	19.80 ± 2.12
	10	19.46 ± 1.35	18.60 ± 1.66	18.96 ± 0.88
	11	19.44 ± 1.64	18.22 ± 1.80	18.06 ± 1.57
	12	19.02 ± 1.60	19.12 ± 1.34	17.58 ± 2.11
	13	18.52 ± 1.55	18.14 ± 2.15	18.72 ± 1.48
	14	18.58 ± 1.58	18.98 ± 1.06	18.72 ± 1.39
	15	18.34 ± 1.28	17.52 ± 1.50	18.20 ± 1.55
	16	18.40 ± 1.18	17.62 ± 1.19	18.08 ± 2.11
	17	18.04 ± 1.37	17.78 ± 1.43	17.68 ± 1.48
	18	18.28 ± 1.38	17.82 ± 1.10	17.66 ± 1.56
	19	17.76 ± 1.41	17.70 ± 1.21	18.26 ± 2.42
	20	17.88 ± 1.29	17.82 ± 1.47	16.74 ± 0.78
	21	17.60 ± 1.27	16.92 ± 1.04	17.84 ± 1.48
	22	17.86 ± 1.16	17.52 ± 1.32	16.58 ± 0.79
	23	17.60 ± 1.38	17.20 ± 1.28	17.88 ± 1.69
	24	17.82 ± 1.23	17.32 ± 1.42	17.10 ± 1.38
	25	18.12 ± 1.43	16.70 ± 2.14	16.84 ± 1.40
	26	17.76 ± 1.53	17.62 ± 1.54	16.56 ± 0.90
	27	17.80 ± 1.50	16.74 ± 1.95	16.64 ± 1.60
	28	17.92 ± 1.39	16.72 ± 1.59	17.18 ± 1.38

COM PARTICIONAMENTO — CONJ. DE TESTE (CONT.)				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>waveform</i>	29	17.72 ± 1.50	16.54 ± 1.02	16.76 ± 1.26
	30	17.66 ± 1.77	16.20 ± 1.15	16.76 ± 1.00
	31	17.68 ± 1.65	16.30 ± 1.16	16.94 ± 1.82

### A.2.2 Taxas de Erro sobre o Conjunto de Treinamento

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO				
Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>abalone</i>	1	75.03 ± 2.60	75.91 ± 2.16	74.47 ± 1.37
	2	75.25 ± 2.89	74.53 ± 1.16	75.03 ± 1.57
	3	73.69 ± 0.80	74.88 ± 1.31	74.06 ± 1.05
	4	73.37 ± 0.85	74.48 ± 1.29	73.76 ± 0.70
	5	73.27 ± 0.93	74.54 ± 0.97	73.99 ± 0.57
	6	73.24 ± 0.72	74.43 ± 0.76	73.84 ± 0.55
	7	73.29 ± 0.74	74.23 ± 0.81	74.08 ± 0.78
	8	73.21 ± 0.78	74.21 ± 0.69	74.33 ± 0.77
	9	73.24 ± 0.89	73.84 ± 0.89	73.64 ± 0.49
	10	73.20 ± 0.95	74.41 ± 0.75	73.68 ± 0.66
	11	73.10 ± 0.86	74.53 ± 0.64	74.06 ± 1.07
	12	73.15 ± 0.91	74.10 ± 0.66	74.13 ± 0.66
	13	73.16 ± 0.84	74.03 ± 0.63	74.05 ± 1.00
	14	73.08 ± 0.82	74.26 ± 1.04	73.95 ± 1.01
	15	73.05 ± 0.85	74.60 ± 0.60	74.07 ± 0.35
	16	73.15 ± 0.78	74.37 ± 0.86	74.14 ± 0.48
	17	73.05 ± 0.89	73.96 ± 0.57	74.11 ± 0.74
	18	73.06 ± 0.84	74.54 ± 0.62	74.27 ± 0.74
	19	73.02 ± 0.83	74.13 ± 0.50	74.06 ± 0.75
	20	73.07 ± 0.85	74.11 ± 0.70	74.13 ± 0.50
	21	73.07 ± 0.87	74.28 ± 1.18	74.06 ± 0.69
	22	73.08 ± 0.90	74.77 ± 0.78	74.00 ± 0.63
	23	73.06 ± 0.84	74.22 ± 0.71	74.25 ± 0.85
	24	73.07 ± 0.89	74.30 ± 0.63	73.99 ± 0.67
	25	73.03 ± 0.89	74.35 ± 0.63	74.19 ± 0.60
	26	73.04 ± 0.89	74.44 ± 0.63	74.01 ± 0.63
	27	73.04 ± 0.88	74.76 ± 0.69	74.44 ± 0.33
	28	73.09 ± 0.85	74.32 ± 0.68	74.34 ± 0.53
	29	73.06 ± 0.88	74.53 ± 0.59	74.24 ± 0.69
	30	73.07 ± 0.91	74.67 ± 0.71	74.07 ± 0.48

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>abalone</i>	31	73.02 ± 0.89	74.28 ± 0.60	74.27 ± 0.34
<i>allhypo</i>	1	0.94 ± 0.56	1.17 ± 0.31	0.99 ± 0.46
	2	0.83 ± 0.41	1.38 ± 0.90	1.71 ± 0.73
	3	0.83 ± 0.40	1.90 ± 0.68	1.23 ± 0.61
	4	0.67 ± 0.37	1.71 ± 0.54	1.30 ± 0.50
	5	0.57 ± 0.25	2.12 ± 1.13	1.66 ± 0.56
	6	0.51 ± 0.14	2.06 ± 1.07	1.64 ± 0.34
	7	0.54 ± 0.15	1.56 ± 0.37	2.74 ± 1.39
	8	0.50 ± 0.18	2.14 ± 1.03	2.08 ± 0.66
	9	0.49 ± 0.12	2.42 ± 1.05	2.42 ± 1.17
	10	0.49 ± 0.11	2.57 ± 1.47	2.08 ± 0.58
	11	0.47 ± 0.12	2.52 ± 0.61	2.11 ± 0.39
	12	0.45 ± 0.12	2.93 ± 1.49	2.98 ± 1.17
	13	0.47 ± 0.11	2.65 ± 1.76	2.79 ± 1.23
	14	0.44 ± 0.10	3.03 ± 1.49	2.85 ± 1.09
	15	0.45 ± 0.12	2.87 ± 1.09	2.60 ± 0.71
	16	0.43 ± 0.10	3.09 ± 1.59	2.79 ± 1.33
	17	0.41 ± 0.10	4.34 ± 1.34	3.78 ± 1.70
	18	0.42 ± 0.12	3.87 ± 1.56	3.72 ± 1.33
	19	0.44 ± 0.12	3.52 ± 1.35	4.57 ± 1.43
	20	0.44 ± 0.12	4.21 ± 1.61	4.36 ± 1.78
	21	0.43 ± 0.14	3.78 ± 1.60	4.05 ± 1.33
	22	0.43 ± 0.14	4.26 ± 1.13	5.61 ± 1.01
	23	0.41 ± 0.12	4.40 ± 1.43	5.25 ± 1.06
	24	0.40 ± 0.11	5.04 ± 1.49	5.86 ± 0.98
	25	0.42 ± 0.12	5.21 ± 1.56	5.02 ± 1.69
	26	0.40 ± 0.11	5.53 ± 1.11	5.19 ± 1.14
	27	0.40 ± 0.12	5.42 ± 1.58	5.16 ± 1.20
	28	0.40 ± 0.11	5.82 ± 1.16	5.81 ± 0.84
	29	0.41 ± 0.14	5.15 ± 1.20	6.33 ± 0.65
	30	0.39 ± 0.13	6.11 ± 0.85	6.05 ± 0.44
	31	0.40 ± 0.13	6.29 ± 0.67	5.95 ± 1.17
<i>arcene</i>	1	16.33 ± 4.24	19.22 ± 4.07	16.50 ± 2.53
	2	15.33 ± 3.51	23.16 ± 5.18	21.01 ± 3.89
	3	13.49 ± 3.11	21.66 ± 4.43	16.83 ± 3.59
	4	11.44 ± 1.80	21.50 ± 3.16	20.28 ± 2.27
	5	11.33 ± 1.85	22.66 ± 4.83	20.44 ± 2.84
	6	9.67 ± 1.80	25.78 ± 4.56	21.78 ± 3.97
	7	10.00 ± 1.92	22.94 ± 4.19	24.50 ± 2.85
	8	9.39 ± 1.90	27.27 ± 4.13	26.72 ± 3.19
	9	9.44 ± 1.83	29.95 ± 3.19	26.66 ± 4.25
	10	9.28 ± 2.54	27.72 ± 3.65	27.33 ± 4.27
	11	8.94 ± 1.74	30.95 ± 3.19	26.00 ± 3.49
	12	9.00 ± 1.79	29.56 ± 3.21	29.50 ± 2.37

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)	
<i>arcene</i>	13	9.28 ± 1.55	32.22 ± 4.17	31.56 ± 3.00	
	14	8.72 ± 1.83	32.83 ± 3.07	28.67 ± 2.25	
	15	8.72 ± 1.59	29.72 ± 3.32	30.61 ± 4.47	
	16	8.72 ± 1.86	33.33 ± 4.79	29.89 ± 5.28	
	17	8.56 ± 1.31	31.66 ± 1.69	31.00 ± 5.47	
	18	8.28 ± 1.44	32.67 ± 3.80	31.17 ± 3.75	
	19	8.39 ± 1.55	31.17 ± 2.29	31.00 ± 2.58	
	20	8.33 ± 1.95	32.33 ± 3.42	30.39 ± 2.80	
	21	8.28 ± 1.92	32.84 ± 4.76	30.78 ± 2.94	
	22	8.44 ± 1.99	32.50 ± 3.43	30.11 ± 3.64	
	23	8.00 ± 2.13	32.44 ± 3.45	30.55 ± 2.73	
	24	8.05 ± 1.56	35.83 ± 5.15	32.50 ± 3.66	
	25	7.94 ± 1.59	33.22 ± 3.98	31.28 ± 2.35	
	26	8.11 ± 1.75	34.22 ± 3.50	31.89 ± 3.42	
	27	8.22 ± 1.80	33.89 ± 3.87	32.66 ± 3.61	
	28	8.00 ± 1.79	34.06 ± 4.33	33.78 ± 3.27	
	29	7.55 ± 1.94	33.50 ± 2.88	31.83 ± 4.13	
	30	7.83 ± 1.84	33.00 ± 3.93	30.22 ± 3.56	
	31	7.50 ± 1.98	32.83 ± 2.16	32.39 ± 1.82	
	<i>geochemical</i>	1	1.21 ± 1.33	1.34 ± 1.27	0.00 ± 0.00
		2	0.74 ± 0.97	2.29 ± 1.82	1.82 ± 1.27
3		0.67 ± 1.27	2.83 ± 2.17	1.75 ± 0.96	
4		0.47 ± 0.55	2.29 ± 1.36	1.75 ± 1.11	
5		0.60 ± 0.74	4.10 ± 4.12	3.03 ± 3.40	
6		0.27 ± 0.35	3.23 ± 2.14	3.52 ± 4.78	
7		0.47 ± 0.64	2.83 ± 1.00	3.77 ± 2.83	
8		0.47 ± 0.55	5.86 ± 3.16	6.19 ± 4.32	
9		0.47 ± 0.55	9.70 ± 6.50	7.54 ± 3.29	
10		0.47 ± 0.55	10.29 ± 5.29	8.94 ± 8.82	
11		0.60 ± 0.67	14.19 ± 10.09	7.82 ± 7.55	
12		0.47 ± 0.55	17.78 ± 8.28	12.81 ± 5.70	
13		0.47 ± 0.55	20.81 ± 7.31	11.85 ± 5.79	
14		0.60 ± 0.67	22.63 ± 5.84	11.86 ± 7.32	
15		0.54 ± 0.62	21.68 ± 5.61	17.85 ± 4.96	
16		0.54 ± 0.62	23.84 ± 4.05	19.88 ± 4.58	
17		0.54 ± 0.62	24.03 ± 8.68	19.00 ± 4.84	
18		0.54 ± 0.62	23.23 ± 5.60	23.57 ± 5.59	
19		0.54 ± 0.62	27.33 ± 6.29	24.57 ± 6.52	
20		0.47 ± 0.55	25.92 ± 5.26	21.82 ± 4.96	
21		0.54 ± 0.62	26.94 ± 3.84	24.51 ± 5.77	
22		0.54 ± 0.62	26.80 ± 4.10	24.11 ± 6.57	
23		0.47 ± 0.55	26.86 ± 6.47	26.59 ± 5.27	
24		0.54 ± 0.62	28.48 ± 4.42	25.06 ± 4.40	
25		0.47 ± 0.55	30.71 ± 4.57	26.41 ± 3.68	

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>geochemical</i>	26	$0.47 \pm 0.55$	$28.35 \pm 3.46$	$25.58 \pm 3.38$
	27	$0.47 \pm 0.55$	$28.48 \pm 2.99$	$24.99 \pm 2.97$
	28	$0.47 \pm 0.55$	$29.02 \pm 3.43$	$23.44 \pm 4.07$
	29	$0.40 \pm 0.47$	$32.25 \pm 4.21$	$25.26 \pm 3.58$
	30	$0.47 \pm 0.55$	$30.84 \pm 4.10$	$27.40 \pm 2.65$
	31	$0.47 \pm 0.55$	$30.97 \pm 4.87$	$27.81 \pm 5.38$
<i>kr-vs-kp</i>	1	$2.75 \pm 1.62$	$1.67 \pm 0.86$	$2.03 \pm 0.35$
	2	$2.70 \pm 1.66$	$2.36 \pm 1.48$	$2.34 \pm 1.16$
	3	$1.94 \pm 0.53$	$1.58 \pm 0.56$	$1.68 \pm 0.64$
	4	$1.64 \pm 0.59$	$1.34 \pm 0.65$	$1.81 \pm 1.15$
	5	$1.72 \pm 0.52$	$1.47 \pm 0.46$	$1.80 \pm 0.75$
	6	$1.89 \pm 0.38$	$1.55 \pm 0.41$	$1.52 \pm 0.41$
	7	$1.87 \pm 0.34$	$1.62 \pm 0.50$	$1.44 \pm 0.35$
	8	$1.83 \pm 0.44$	$1.72 \pm 0.40$	$1.73 \pm 0.43$
	9	$1.83 \pm 0.42$	$1.75 \pm 0.27$	$1.79 \pm 0.58$
	10	$1.69 \pm 0.43$	$1.76 \pm 0.38$	$1.71 \pm 0.41$
	11	$1.68 \pm 0.48$	$2.19 \pm 0.53$	$1.93 \pm 0.44$
	12	$1.67 \pm 0.33$	$2.29 \pm 0.70$	$2.13 \pm 0.35$
	13	$1.69 \pm 0.43$	$2.20 \pm 0.50$	$2.24 \pm 0.51$
	14	$1.60 \pm 0.57$	$2.44 \pm 0.42$	$2.46 \pm 0.53$
	15	$1.61 \pm 0.52$	$2.41 \pm 0.34$	$2.37 \pm 0.24$
	16	$1.52 \pm 0.45$	$2.62 \pm 0.46$	$2.54 \pm 0.56$
	17	$1.44 \pm 0.56$	$2.58 \pm 0.59$	$2.80 \pm 0.60$
	18	$1.56 \pm 0.48$	$2.71 \pm 0.39$	$3.29 \pm 0.36$
	19	$1.61 \pm 0.52$	$2.83 \pm 0.46$	$2.80 \pm 0.36$
	20	$1.61 \pm 0.53$	$2.93 \pm 0.23$	$2.96 \pm 0.40$
	21	$1.62 \pm 0.53$	$3.01 \pm 0.74$	$3.26 \pm 0.41$
	22	$1.61 \pm 0.53$	$3.38 \pm 0.53$	$3.22 \pm 0.42$
	23	$1.61 \pm 0.53$	$3.21 \pm 0.36$	$3.43 \pm 0.32$
	24	$1.69 \pm 0.38$	$3.44 \pm 0.48$	$3.78 \pm 0.33$
	25	$1.68 \pm 0.38$	$3.53 \pm 0.52$	$3.60 \pm 0.59$
	26	$1.68 \pm 0.38$	$3.65 \pm 0.60$	$3.51 \pm 0.50$
	27	$1.61 \pm 0.49$	$3.54 \pm 0.49$	$3.80 \pm 0.50$
	28	$1.73 \pm 0.34$	$3.70 \pm 0.43$	$3.69 \pm 0.62$
	29	$1.57 \pm 0.46$	$3.88 \pm 0.59$	$3.99 \pm 0.54$
	30	$1.73 \pm 0.34$	$3.64 \pm 0.38$	$3.62 \pm 0.54$
	31	$1.69 \pm 0.30$	$4.03 \pm 0.52$	$3.73 \pm 0.45$
<i>optdigits</i>	1	$32.59 \pm 4.67$	$31.46 \pm 2.82$	$30.61 \pm 2.27$
	2	$30.50 \pm 4.64$	$27.91 \pm 1.38$	$29.20 \pm 1.99$
	3	$22.77 \pm 1.55$	$26.51 \pm 3.02$	$25.57 \pm 2.03$
	4	$19.58 \pm 1.24$	$23.06 \pm 3.09$	$23.06 \pm 1.94$
	5	$17.32 \pm 1.37$	$24.16 \pm 2.56$	$21.50 \pm 2.04$
	6	$16.04 \pm 1.33$	$21.95 \pm 2.83$	$20.79 \pm 2.11$
	7	$15.34 \pm 1.31$	$19.78 \pm 2.65$	$21.26 \pm 3.07$

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos			
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)	
<i>optdigits</i>	8	14.99 ± 1.43	20.19 ± 1.58	19.60 ± 2.64	
	9	14.33 ± 1.66	19.39 ± 3.46	19.76 ± 3.71	
	10	13.73 ± 1.53	18.03 ± 2.49	17.58 ± 1.63	
	11	13.37 ± 1.52	17.24 ± 2.14	17.78 ± 2.75	
	12	13.00 ± 1.35	15.79 ± 2.21	17.25 ± 1.88	
	13	12.73 ± 1.27	15.92 ± 1.47	15.75 ± 1.36	
	14	12.56 ± 1.30	15.39 ± 1.92	15.02 ± 2.21	
	15	12.40 ± 1.16	14.87 ± 1.63	14.52 ± 1.70	
	16	12.28 ± 1.23	14.39 ± 1.56	15.57 ± 1.84	
	17	12.36 ± 1.22	14.46 ± 0.97	14.12 ± 0.53	
	18	12.48 ± 1.32	14.24 ± 1.43	14.77 ± 1.91	
	19	12.27 ± 1.23	13.93 ± 1.20	13.17 ± 1.54	
	20	12.18 ± 1.16	13.82 ± 0.80	13.52 ± 1.15	
	21	12.01 ± 1.17	13.38 ± 1.31	13.87 ± 1.14	
	22	11.85 ± 1.16	13.46 ± 1.26	13.49 ± 1.78	
	23	11.87 ± 1.09	13.99 ± 1.78	12.07 ± 1.46	
	24	11.87 ± 1.08	13.34 ± 1.43	12.66 ± 1.45	
	25	11.68 ± 1.27	12.76 ± 1.22	12.13 ± 1.15	
	26	11.58 ± 1.24	13.51 ± 1.76	12.96 ± 1.87	
	27	11.41 ± 1.07	12.18 ± 1.30	11.57 ± 0.82	
	28	11.33 ± 1.11	12.05 ± 1.26	12.10 ± 1.40	
	29	11.27 ± 1.06	12.58 ± 1.21	12.09 ± 1.11	
	30	11.24 ± 1.17	12.22 ± 1.49	12.12 ± 1.28	
	31	11.16 ± 1.13	12.35 ± 1.67	12.42 ± 1.19	
	<i>segmentation</i>	1	9.22 ± 1.75	8.01 ± 1.76	9.26 ± 1.98
		2	8.07 ± 1.37	9.03 ± 2.18	8.41 ± 0.77
		3	7.26 ± 1.55	7.61 ± 1.30	8.56 ± 1.90
		4	6.46 ± 1.24	7.28 ± 1.11	8.14 ± 1.46
		5	6.18 ± 0.82	7.62 ± 1.27	8.35 ± 1.53
		6	5.78 ± 0.71	8.39 ± 1.58	7.58 ± 1.64
		7	5.60 ± 0.60	7.70 ± 0.95	7.74 ± 1.06
8		5.36 ± 0.68	7.37 ± 0.97	7.73 ± 0.76	
9		5.42 ± 0.68	7.98 ± 0.92	7.33 ± 0.80	
10		5.36 ± 0.79	7.92 ± 1.02	7.37 ± 0.83	
11		5.35 ± 0.88	7.38 ± 0.69	7.95 ± 0.73	
12		5.31 ± 0.82	8.52 ± 0.73	7.81 ± 1.12	
13		5.22 ± 0.73	7.94 ± 0.98	8.58 ± 0.86	
14		5.06 ± 0.80	8.13 ± 0.83	8.34 ± 0.91	
15		5.18 ± 0.68	8.56 ± 1.18	8.28 ± 1.07	
16		5.09 ± 0.67	8.40 ± 0.97	8.51 ± 0.92	
17		5.11 ± 0.59	8.71 ± 1.14	8.80 ± 0.96	
18		5.08 ± 0.65	8.32 ± 0.95	9.37 ± 1.24	
19		5.07 ± 0.74	8.20 ± 0.62	8.37 ± 1.32	
20		5.14 ± 0.69	8.71 ± 0.83	8.44 ± 1.03	

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>segmentation</i>	21	5.12 ± 0.61	8.76 ± 1.17	8.89 ± 0.71
	22	5.11 ± 0.54	8.56 ± 0.96	8.61 ± 0.69
	23	5.01 ± 0.54	8.63 ± 0.88	9.42 ± 1.10
	24	4.98 ± 0.57	8.81 ± 0.42	9.01 ± 0.83
	25	4.87 ± 0.50	9.37 ± 0.91	8.78 ± 0.78
	26	4.92 ± 0.43	9.11 ± 0.82	8.96 ± 0.70
	27	4.79 ± 0.36	8.72 ± 0.51	8.86 ± 1.15
	28	4.77 ± 0.38	9.07 ± 0.94	9.07 ± 0.94
	29	4.79 ± 0.38	8.66 ± 0.96	8.82 ± 0.68
	30	4.75 ± 0.46	9.39 ± 1.32	9.02 ± 0.63
	31	4.72 ± 0.40	9.07 ± 0.82	9.39 ± 0.91
<i>splice</i>	1	0.13 ± 0.05	0.14 ± 0.09	0.00 ± 0.00
	2	0.17 ± 0.11	0.17 ± 0.07	0.15 ± 0.10
	3	0.06 ± 0.09	0.17 ± 0.07	0.08 ± 0.08
	4	0.04 ± 0.03	0.16 ± 0.10	0.18 ± 0.16
	5	0.04 ± 0.03	0.24 ± 0.16	0.14 ± 0.09
	6	0.04 ± 0.03	0.24 ± 0.18	0.13 ± 0.10
	7	0.03 ± 0.03	0.15 ± 0.06	0.19 ± 0.14
	8	0.02 ± 0.02	0.17 ± 0.12	0.16 ± 0.11
	9	0.03 ± 0.03	0.26 ± 0.22	0.23 ± 0.16
	10	0.02 ± 0.02	0.20 ± 0.15	0.19 ± 0.07
	11	0.01 ± 0.02	0.21 ± 0.15	0.20 ± 0.12
	12	0.02 ± 0.02	0.18 ± 0.14	0.11 ± 0.07
	13	0.01 ± 0.02	0.17 ± 0.10	0.13 ± 0.12
	14	0.01 ± 0.02	0.21 ± 0.12	0.21 ± 0.10
	15	0.01 ± 0.02	0.20 ± 0.17	0.24 ± 0.17
	16	0.01 ± 0.02	0.18 ± 0.12	0.15 ± 0.08
	17	0.01 ± 0.01	0.24 ± 0.21	0.11 ± 0.05
	18	0.01 ± 0.01	0.14 ± 0.11	0.15 ± 0.07
	19	0.01 ± 0.01	0.19 ± 0.11	0.15 ± 0.10
	20	0.01 ± 0.01	0.13 ± 0.10	0.20 ± 0.16
	21	0.01 ± 0.01	0.16 ± 0.05	0.16 ± 0.14
	22	0.01 ± 0.02	0.16 ± 0.12	0.20 ± 0.16
	23	0.01 ± 0.02	0.21 ± 0.16	0.23 ± 0.12
	24	0.01 ± 0.01	0.17 ± 0.07	0.16 ± 0.11
	25	0.01 ± 0.01	0.29 ± 0.15	0.13 ± 0.08
	26	0.01 ± 0.01	0.18 ± 0.11	0.12 ± 0.08
	27	0.01 ± 0.01	0.12 ± 0.08	0.14 ± 0.08
	28	0.01 ± 0.01	0.17 ± 0.12	0.17 ± 0.10
	29	0.01 ± 0.02	0.14 ± 0.10	0.18 ± 0.13
	30	0.01 ± 0.01	0.17 ± 0.14	0.15 ± 0.14
	31	0.01 ± 0.02	0.17 ± 0.06	0.13 ± 0.13
<i>waveform</i>	1	25.81 ± 1.33	25.54 ± 1.76	25.83 ± 2.07
	2	24.95 ± 1.93	26.72 ± 2.13	25.96 ± 1.29

COM PARTICIONAMENTO — CONJ. DE TREINAMENTO (CONT.)

Base de dados	Tam. do comitê	Algoritmos		
		PGMC original (%)	Particionamento <i>bootstrap</i> (%)	Particionamento <i>aleatório</i> (%)
<i>waveform</i>	3	21.32 ± 0.60	21.88 ± 1.05	22.16 ± 1.11
	4	20.34 ± 0.90	21.71 ± 0.98	21.36 ± 0.81
	5	19.01 ± 0.66	19.43 ± 0.77	19.86 ± 0.67
	6	18.78 ± 0.79	19.60 ± 1.05	19.41 ± 0.95
	7	18.05 ± 0.64	19.15 ± 1.31	18.59 ± 0.43
	8	17.99 ± 0.76	18.50 ± 0.58	18.49 ± 0.55
	9	17.50 ± 0.83	18.24 ± 0.80	17.85 ± 0.66
	10	17.35 ± 0.38	17.97 ± 0.70	17.78 ± 0.79
	11	16.92 ± 0.41	17.47 ± 0.45	17.59 ± 0.74
	12	16.89 ± 0.38	17.49 ± 0.96	17.46 ± 0.55
	13	16.66 ± 0.34	17.63 ± 0.66	17.20 ± 0.50
	14	16.69 ± 0.44	17.14 ± 0.57	16.92 ± 0.69
	15	16.46 ± 0.40	16.65 ± 0.71	16.94 ± 0.71
	16	16.51 ± 0.36	16.96 ± 0.45	17.13 ± 0.69
	17	16.25 ± 0.36	16.65 ± 0.39	16.76 ± 0.35
	18	16.22 ± 0.31	16.64 ± 0.72	16.36 ± 0.92
	19	16.11 ± 0.31	16.71 ± 0.68	16.52 ± 0.46
	20	16.23 ± 0.42	16.52 ± 0.54	16.42 ± 0.36
	21	16.03 ± 0.34	16.46 ± 0.46	16.16 ± 0.43
	22	16.10 ± 0.29	16.59 ± 0.57	16.24 ± 0.64
	23	15.97 ± 0.28	16.26 ± 0.61	16.50 ± 0.63
	24	16.08 ± 0.42	16.32 ± 0.63	16.26 ± 0.40
	25	16.01 ± 0.40	16.15 ± 0.57	16.48 ± 0.65
	26	16.01 ± 0.40	15.87 ± 0.66	15.95 ± 0.53
	27	15.99 ± 0.46	16.01 ± 0.42	15.95 ± 0.55
	28	15.95 ± 0.50	15.89 ± 0.70	15.73 ± 0.53
	29	15.86 ± 0.44	16.05 ± 0.53	15.76 ± 0.62
	30	15.93 ± 0.42	16.27 ± 0.49	15.96 ± 0.40
	31	15.76 ± 0.42	16.05 ± 0.31	15.64 ± 0.63