Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Descrição da Arquitetura e do Projeto do Sistema Computacional GAERE para Realizar Evolução Genética de Classificadores Simbólicos

Flávia Cristina Bernardini
Maria Carolina Monard

Nº 275

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos, Maio de 2006

Descrição da Arquitetura e do Projeto do Sistema Computacional GAERE para Realizar Evolução Genética de Classificadores Simbólicos *

Flávia Cristina Bernardini

Maria Carolina Monard

Departamento de Ciências de Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo — Campus de São Carlos
Caixa Postal 668 - 13560-970 - São Carlos, SP
e-mail: {fbernard, mcmonard}@icmc.usp.br

Resumo

Nos dias atuais, há uma grande quantidade de dados armazenados em bases de dados reais, que continua a crescer rapidamente. Assim, existe a necessidade de extrair conhecimento da informação contida nesses dados. Esse problema é típico da área de Mineração de Dados, na qual é frequentemente utilizado Aprendizado de Máquina para a extração de conhecimento. Entretanto, os algoritmos de aprendizado geralmente realizam um tipo de busca local no espaço de soluções (hipóteses) para encontrar uma solução para o problema, encontrando assim soluções ótimas locais. Além disso, esses algoritmos avaliam a hipótese incompleta a cada iteração até chegar na hipótese final. Para tentar resolver esses problemas, pode-se utilizar Algoritmos Evolutivos (AEs), os quais geralmente realizam uma busca mais global no espaço de hipóteses e, tipicamente, avaliam hipóteses (soluções) completas a cada iteração para chegar à hipótese (solução) final. Também, em problemas que envolvem grandes bases de dados, pode-se retirar amostras dessas bases, induzir classificadores dessas amostras e evoluir esses classificadores utilizando AEs. O objetivo deste trabalho é propor e descrever o projeto de um sistema computacional, denominado GAERE, que implementa Algoritmos Genéticos, um sub-paradigma de AEs, para evoluir hipóteses (classificadores) simbólicas em uma hipótese final, única, preferencialmente melhor que as hipóteses iniciais. O foco deste trabalho está em evoluir classificadores simbólicos, que podem ser expressos na forma de regras de conhecimento, para que o classificador final possa explicar ao usuário a classificação atribuída a um novo exemplo; essa capacidade de explicação é de fundamental importância em muitos domínios de conhecimento, tais como medicina.

Palavras-Chave: Aprendizado de Máquina, Algoritmos Genéticos, Computação Evolutiva, Classificadores Simbólicos.

Maio 2006

*Trabalho realizado com auxílio da FAPESP, Proc. Nº 02/06914-5.

Conteúdo

1 Introdução **1**

2 Aprendizado de Máquina **2**

2.1 Aprendizado de Máquina Supervisionado 2

2.2 Classificadores Simbólicos 3

2.3 Avaliação de Classificadores 3

2.4 Avaliação de Regras 7

2.5 Regras de Associação 11

3 Algoritmos Evolutivos **11**

3.1 Algoritmos Genéticos 13

3.1.1 Algoritmo Genético Clássico 14

3.1.2 Componentes de um AG Clássico 15

3.1.3 Um Exemplo 17

3.2 Algoritmos Genéticos Aplicados na Evolução de Classificadores Simbólicos 18

4 Proposta de um Algoritmo Genético para Evoluir Classificadores Simbólicos **20**

4.1 Codificação dos Indivíduos 21

4.2 Operadores Genéticos 21

4.3 Funções de Avaliação 22

4.4 Critério de Parada 24

4.5 Pós-processamento do Indivíduo Resultante 24

4.6 Algoritmo Genético Proposto 24

5 O Ambiente Discover **25**

5.1 O Ambiente Computacional DLE 26

5.2 A Biblioteca de Classes DOL 27

5.3 O Ambiente Computacional Sniffer 28

5.4 O Sistema Computacional NNRules 31

5.5 O Sistema Computacional ELE 32

6 Descrição do Sistema Computacional para Evoluir Classificadores Simbólicos — GAERE **34**

6.1 Avaliação Experimental do AG proposto 34
6.2 O Módulo GATree 40
6.3 O Módulo RuleStruct 41
6.4 O Módulo RuleSetGA 41
6.5 O Módulo GA 41

7 Conclusão **42**

A Análise Qualitativa de Regras **44**
A.1 Uma Visão Unificada de Medidas de Avaliação de Regras 44
A.2 Outros Critérios para Seleção da Melhor Regra 44

B Extensão do Ambiente Sniffer **48**

C Configurações do Algoritmo Genético **49**

Lista de Figuras

1 Técnica de validação simples 6
2 Exemplo de *k-fold cross-validation*, com $k = 3$ 7
3 Regras ordenadas entre classes — conjunto de subconjuntos de regras 10
4 Uma geração de um algoritmo genético 15
5 Base de regras na sintaxe padrão $\mathcal{PBM}$. 21
6 Exemplo de codificação de um indivíduo 21
7 Exemplo de experimento organizado em diretórios para o ambiente Sniffer . . . 29
8 Estimativa de erro ε^* de um classificador $\mathbf{h}^*$ evoluído com o AG proposto 37
9 Configuração da árvore de diretórios para induzir os classificadores simbólicos cujas regras compõem a população inicial do(s) AG(s) a ser(em) executado(s) . . 38
10 Configuração da árvore de diretórios para executar o algoritmo genético proposto 38
11 Árvore de diretórios instanciada para induzir os classificadores simbólicos cujas regras compõem a população inicial do(s) AG(s) a ser(em) executado(s) 40
12 Instanciação da árvore de diretórios para executar os AGs representados pelos identificadores GA.mracc.hqprec, GA.mracc.hqpc e GA.srlap.hqprec 40
13 Diagrama UML que envolve os módulos RuleStruct, RuleSetGA e GA . . . 43
14 Diagrama UML da extensão realizada no ambiente Sniffer 48
15 Arquivo de configurações do AG 50

Lista de Tabelas

1 Conjunto de exemplos no formato atributo-valor 2

2 Matriz de confusão 4

3 Matrix de confusão para problemas de classificação com somente duas classes . . 5

4 Matriz de contingência de uma regra 8

5 Identificadores especiais para diretórios reconhecidos atualmente pelo SNIFFER para algoritmos de aprendizado simbólico 30

6 Identificadores que indicam qual função de avaliação utilizar na execução do algoritmo genético 36

7 Medidas genéricas de avaliação de regras 46

8 Medidas relativas de avaliação de regras 47

9 Medidas relativas de avaliação de regras com peso 47

10 Parâmetros do AG proposto 49

[1]Este documento foi elaborado com o LaTeX, sua bibliografia é mantida com o auxílio da ferramenta BibView (Prati et al., 1999) e gerada automaticamente pelo BibTeX

1 Introdução

Nos dias atuais, há uma grande quantidade de dados armazenados em bases de dados reais, que continua a crescer rapidamente. Assim, existe a necessidade de minerar esses dados para extração de conhecimento da informação contida nesses dados. Esse problema é típico da área de Mineração de Dados[1] (MD) (Cabena et al., 1998; Mannila, 1996; Mitchell, 1999; Rezende et al., 2003; Weiss and Indurkhya, 1998), na qual são frequentemente utilizados algoritmos de Aprendizado de Máquina (AM) (Mitchell, 1997b) para a extração de padrões/modelos desses dados.

Os algoritmos de aprendizado supervisionado geralmente realizam um tipo de busca local no espaço de soluções (hipóteses) para encontrar uma solução para o problema, ou seja, uma hipótese que represente o modelo. Em alguns casos, tais algoritmos podem encontrar soluções ótimas locais, "distantes" da solução ótima global. Além disso, esses algoritmos tipicamente constroem a hipótese e a avaliam de maneira incremental, ou seja, avaliam a hipótese incompleta a cada iteração até chegar na hipótese final (Freitas, 2002a). Para tentar resolver esses problemas, pode-se utilizar Algoritmos Evolutivos (AEs), como o algoritmo proposto neste trabalho. AEs geralmente realizam uma busca mais global no espaço de hipóteses, e tipicamente avaliam hipóteses (soluções) completas a cada iteração para chegar à hipótese (solução) final.

Como mencionado, MD lida com grandes bases de dados. A qualidade das hipóteses induzidas pelos atuais sistemas de aprendizado depende, principalmente, da quantidade e da qualidade dos atributos e exemplos (ou casos) disponíveis a serem utilizados na etapa de treinamento. Freqüentemente, em experimentos em que são utilizadas grandes bases de dados, nas quais muitos atributos irrelevantes estão presentes, são induzidas hipóteses de baixa precisão. Por outro lado, muitos dos sistemas de aprendizado de máquina conhecidos não estão preparados para trabalhar com uma quantidade muito grande de exemplos. Para que os conceitos sejam aprendidos a partir de grandes bases de dados utilizando sistemas de AM, de maneira tal que esses problemas sejam reduzidos, pode-se utilizar varias abordagens. Uma delas está relacionada com extração de amostras de exemplos da base de dados original, induzir hipóteses dessas amostras e combinar essas hipóteses de alguma maneira. AEs é um paradigma que pode ser utilizado para evoluir essas hipóteses em uma hipótese (final) única.

Em trabalhos anteriores (Bernardini and Monard, 2004, 2005a; Bernardini et al., 2005; Bernardini and Monard, 2005b), bem como no presente trabalho, estamos interessados, entre outros, com a explicação da classificação de novos exemplos. Para extrair modelos que expressam o conhecimento induzido em uma linguagem que seja compreensível para o especialista do domínio, geralmente são utilizados algoritmos de aprendizado simbólico. Modelos (hipóteses) induzidos por algoritmos de aprendizado simbólicos podem, geralmente, ser expressos na forma de regras de conhecimento. Tais algoritmos são utilizados neste trabalho.

Na maioria dos projetos desenvolvidos pelos pesquisadores do Laboratório de Inteligência Computacional — LABIC[2] — relacionados com aquisição automática de conhecimento e avaliação de conhecimento, inclusive no projeto no qual está inserido este trabalho, diversas tarefas como transformação de dados e formatos, execução de algoritmos de aprendizado, medições, entre outras, devem ser executadas diversas vezes. Para tentar resolver tais problemas, foi inicialmente proposto por Baranauskas and Batista (2000) um projeto denominado Projeto Discover, que tem como principal objetivo integrar e padronizar os diversos projetos e ferra-

[1] *Data Mining*
[2] http://labic.icmc.usp.br

mentas desenvolvidos pelos pesquisadores do LABIC, relacionados com aquisição automática de conhecimento e avaliação de conhecimento.

Neste trabalho, é proposto e descrito o projeto de um módulo computacional integrado ao DISCOVER que implementa Algoritmos Genéticos (AGs), um sub-paradigma de AEs, cujo objetivo é evoluir hipóteses simbólicas, as quais podem ser provenientes de hipóteses induzidas utilizando diversas amostras da base de dados, ou podem ser provenientes de outros processos, em uma hipótese final, única, preferencialmente melhor que as hipóteses iniciais. Tal sistema computacional foi denominado GAERE — *Genetic Algorithms for Evolving Rule Sets Environment.*

Este trabalho está organizado da seguinte maneira: Na Seção 2 são descritos conceitos de aprendizado de máquina supervisionado e a notação utilizada neste trabalho. Na Seção 3, são descritos conceitos de algoritmos evolutivos, bem como métodos para aplicar algoritmos genéticos na evolução de hipóteses simbólicas. Na Seção 4, é proposto um AG para evoluir classificadores simbólicos. Na Seção 5, é descrito o ambiente DISCOVER, no qual está inserido o ambiente computacional GAERE. Na Seção 6, é descrito o sistema proposto neste trabalho para evolução de classificadores simbólicos. Na Seção 7, são descritos os resultados obtidos nos experimentos realizados com o sistema proposto. Finalmente, na Seção 8, é apresentada a conclusão do trabalho realizado.

2 Aprendizado de Máquina

Nesta seção são brevemente descritos alguns conceitos de aprendizado de máquina supervisionado, com o objetivo de facilitar a leitura da proposta e do projeto do sistema descrito neste trabalho.

2.1 Aprendizado de Máquina Supervisionado

No problema padrão de aprendizado supervisionado, a entrada do algoritmo consiste de um conjunto de exemplos S, com N exemplos $T_i, i = 1, ..., N$, escolhidos de um domínio X com uma distribuição D fixa, desconhecida e arbitrária, da forma $\{(\mathbf{x}_1,y_1),...,(\mathbf{x}_N,y_N)\}$ para alguma função desconhecida $y = f(\mathbf{x})$. Os $\mathbf{x}_i$ são tipicamente vetores da forma (x_{i1}, x_{i2}, ..., x_{iM}) com valores discretos ou numéricos. x_{ij} refere-se ao valor do atributo j, denominado $\mathbf{X}_j$, do exemplo T_i, como mostra a Tabela 1. Os valores y_i referem-se ao valor do atributo Y, freqüentemente denominado classe.

	X_1	X_2	$\dots$	X_M	Y
T_1	x_{11}	x_{12}	$\dots$	x_{1M}	y_1
T_2	x_{21}	x_{22}	$\dots$	x_{2M}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
T_N	x_{N1}	x_{N2}	$\dots$	x_{NM}	y_N

Tabela 1: Conjunto de exemplos no formato atributo-valor

Os valores de y são tipicamente pertencentes a um conjunto discreto de classes $C_v, v = 1, ..., N_{Cl}$, da forma $\{C_1, ..., C_{N_{Cl}}\}$, quando se trata de *classificação*, ou ao conjunto de números reais em caso de *regressão*. O enfoque deste trabalho é problemas de classificação. Para facilitar a leitura deste documento, quando for dito que um exemplo pertence à uma determinada classe

C_v, isso significa que o exemplo possui C_v como valor de y ou, ainda, o valor C_v foi associado a y quando o problema é associar uma classe a um exemplo.

Dado um conjunto S de exemplos a um algoritmo de aprendizado, um *classificador* **h** será induzido. O classificador consiste da *hipótese* feita sobre a verdadeira (mas desconhecida) função f. Dados novos exemplos **x**, o classificador, ou hipótese, **h** prediz o valor correspondente y.

2.2 Classificadores Simbólicos

Algoritmos de aprendizado simbólicos induzem o classificador **h** de tal forma que o conceito descrito por **h** é facilmente interpretável por seres humanos (Mitchell, 1997b; Monard and Baranauskas, 2003). Neste trabalho, são considerados simbólicos os classificadores cuja linguagem de descrição pode ser transformada em um conjunto de N_R regras *if-then*, *i.e.* $\mathbf{h} = \{R_1, R_2, ..., R_{N_R}\}$, do tipo

$$Corpo \rightarrow Cabeça \text{ ou } Body \rightarrow Head.$$

Um *complexo* é uma disjunção de conjunções de testes de atributos da forma X_i *op Valor*, onde X_i é o nome de um atributo, *op* é um operador pertencente ao conjunto $\{=, \neq, <, \leq, >, \geq\}$ e *Valor* é um valor válido do atributo X_i. Uma *regra* R assume a forma **if** B **then** H ou simbolicamente $B \rightarrow H$, onde H é a cabeça, ou *conclusão* da regra, e B é o corpo, ou *condição* da regra. H e B são ambos complexos sem nenhum atributo em comum. Em uma regra de classificação, H (a cabeça da regra) assume a forma **class** $= C_i$, onde $C_i \in \{C_1, ..., C_{N_{Cl}}\}$.

Classificadores compostos por regras podem incluir uma regra especial — a regra *default* —, a qual possui *Corpo* vazio e a *Cabeça* é dada pela classe predita pela maioria dos exemplos no conjunto de treinamento.

O conjunto de regras que constituem um classificador simbólico $\mathbf{h} = \{R_1, ..., R_{N_R}\}$, pode ser avaliado tanto considerando a precisão de **h** como um todo, ou seja, como um classificador tipo caixa preta — Seção 2.3 — , ou avaliando separadamente cada uma das regras que constituem **h** — Seção 2.4.

2.3 Avaliação de Classificadores

Dado um conjunto de exemplos S, esse conjunto de exemplos deve ser utilizado em diferentes fases do aprendizado. Para cada fase, é utilizado um subconjunto desse conjunto de exemplos, identificado como:

Conjunto de Treinamento: Esse conjunto é a principal entrada dos algoritmos de aprendizado. É a partir dele que são construídos os modelos e, portanto, ele deve ser um subconjunto representativo do conjunto original, ou seja, que tenha uma distribuição o mais semelhante possível do conjunto original, para que seja realizada inferência sobre S. Na literatura, esse conjunto também é conhecido como *seen cases*, pois se refere aos exemplos que foram "vistos"pelo algoritmo de aprendizado durante a construção do modelo.

Conjunto de Teste: Esse conjunto é utilizado para avaliar o modelo construído. O conjunto de teste, também conhecido como *unseen cases*, não deve ser apresentado ao algoritmo

de aprendizado durante a construção do modelo. Idealmente, esse conjunto não deve ter exemplos em comum com o conjunto de treinamento.

Conjunto de Validação: Em alguns casos, pode ser necessário utilizar exemplos para realizar ajustes no modelo construído pelo algoritmo de aprendizado. Esses exemplos não são utilizados para construir o modelo, mas são utilizados para seu "ajuste". Dessa maneira, esses exemplos são indiretamente "vistos"durante o processo de aprendizado, o que obriga que os exemplos de validação sejam distintos dos exemplos de teste.

Para avaliar um classificador $\mathbf{h}$, é necessário coletar informações das decisões tomadas pelo classificador no conjunto de teste $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}, y_i \in \{C_1, ..., C_{N_{Cl}}\}$. Para isso, constrói-se uma matriz bi-dimensional, cujas dimensões são denominadas *classe verdadeira* e *classe predita*. A essa matriz dá-se o nome de matriz de confusão, mostrada na Tabela 2 para N_{Cl} classes. Cada elemento $M(C_i, C_j)$ da matriz, definido pela Equação 1, indica o número de exemplos que pertencem à classe C_i e foram preditos como pertencentes à classe C_j. O número de predições corretas para cada classe são os apresentados na diagonal principal da matriz de confusão, ou seja, os valores associados a $M(C_i, C_i)$. Todos os outros elementos da matriz $M(C_i, C_j)$, para $i \neq j$, são referentes ao número de erros cometidos em cada classe.

Classe Verdadeira	Predita C_1	Predita C_2	$\cdots$	Predita $C_{N_{Cl}}$
C_1	$M(C_1, C_1)$	$M(C_1, C_2)$	$\cdots$	$M(C_1, C_{N_{Cl}})$
C_2	$M(C_2, C_1)$	$M(C_2, C_2)$	$\cdots$	$M(C_2, C_{N_{Cl}})$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$C_{N_{Cl}}$	$M(C_{N_{Cl}}, C_1)$	$M(C_{N_{Cl}}, C_2)$	$\cdots$	$M(C_{N_{Cl}}, C_{N_{Cl}})$

Tabela 2: Matriz de confusão

$$M(C_i, C_j) = \sum_{\forall(\mathbf{x},y)\in S|y=C_i} ||h(\mathbf{x}) = C_j|| \tag{1}$$

Para simplificar, considere um problema de classificação com duas classes, geralmente rotulados como sendo "+" e "-". Na Tabela 3 é ilustrada a matriz de confusão para problemas com somente dois valores no atributo classe, onde T_P é o número de exemplos positivos classificados corretamente, F_N é o número de exemplos positivos classificados erroneamente, F_P é o número de exemplos negativos classificados erroneamente, e T_N é o número de exemplos negativos classificados corretamente, de um total de $N = (T_P + F_N + F_P + T_N)$ exemplos. A acurácia e a taxa de erro, definidas pelas Equações 2 e 3 respectivamente, são duas das medidas mais utilizadas para avaliar a performance de uma hipótese. Outras medidas que também podem ser utilizadas para avaliar um classificador são sensibilidade ou *recall* (Equação 4), precisão (Equação 5) e F_1 (Equação 6). Para podermos visualizar o que F_1 significa em termos dos valores T_P, F_P e F_N, a Equação 6 foi desenvolvida na Equação 7. Deve ser observado que, para serem utilizadas tais medidas em problemas com mais de dois valores para o atributo classe, o processo que se utiliza geralmente é tornar cada classe como sendo a classe positiva e a união de todas as outras é considerada a classe negativa. Entretanto, para facilitar a utilização das medidas de *recall* e precisão, foram utilizadas neste trabalho as Equações 8 e 9, respectivamente[3].

[3]A definição das medidas *Recall* e *Prec* para classes com mais de dois valores foi baseada na definição utilizada no KDD-Cup de 2005 — http://www.acm.org/sigs/sigkdd/kdd2005/kddcup.html.

Classe Verdadeira	Predita +	Predita -
+	Verdadeiros Positivos T_P	Falsos Negativos F_N
-	Falsos Positivos F_P	Verdadeiros Negativos T_N

Tabela 3: Matrix de confusão para problemas de classificação com somente duas classes

$$Acc(\mathbf{h}) = \frac{T_P + T_N}{N} \tag{2}$$

$$Err(\mathbf{h}) = 1 - Acc(\mathbf{h}). \tag{3}$$

$$Recall(\mathbf{h}) = \frac{T_P}{T_P + F_N}. \tag{4}$$

$$Prec(\mathbf{h}) = \frac{T_P}{T_P + F_P}. \tag{5}$$

$$F_1(\mathbf{h}) = \frac{2 \times Prec(\mathbf{h}) \times Recall(\mathbf{h})}{Prec(\mathbf{h}) + Recall(\mathbf{h})}. \tag{6}$$

$$\begin{aligned} F1(\mathbf{h}) = & \frac{2 \times Prec(\mathbf{h}) \times Recall(\mathbf{h})}{Prec(\mathbf{h}) + Recall(\mathbf{h})} \\ = & \frac{2 \times \frac{T_P}{T_P+F_P} \times \frac{T_P}{T_P+F_N}}{\frac{T_P}{T_P+F_P} + \frac{T_P}{T_P+F_N}} \\ = & \frac{2T_P^2}{T_P(T_P+F_N+T_P+F_P)} = \frac{2T_P}{2T_P+F_N+F_P} \end{aligned} \tag{7}$$

$$Recall(\mathbf{h}) = \frac{\sum_{v=1}^{N_{Cl}} T_{P_{C_v}}}{\sum_{v=1}^{N_{Cl}} T_{P_{C_v}} + F_{N_{C_v}}}. \tag{8}$$

$$Prec(\mathbf{h}) = \frac{\sum_{v=1}^{N_{Cl}} T_{P_{C_v}}}{\sum_{v=1}^{N_{Cl}} T_{P_{C_v}} + F_{P_{C_v}}}. \tag{9}$$

Como mencionado, para estimar o erro, deve ser utilizado o conjunto de teste com exemplos não vistos pelo algoritmo na indução do classificador. A fim de estimar o erro verdadeiro, existem diversas técnicas para construir o conjunto de treinamento e teste, dentre elas pode-se citar:

Validação Simples: Nesta técnica, retiram-se de 5 a 33% dos exemplos da base de dados para teste e o restante é utilizado para a construção do classificador. Após construído o classificador, esse conjunto de teste é usado para estimar o erro verdadeiro do classificador. Na Figura 1 é ilustrada a técnica de validação simples.

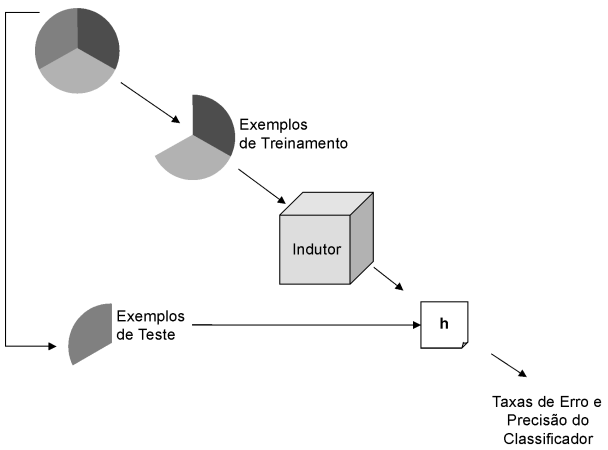

Figura 1: Técnica de validação simples

***K-Fold Cross-Validation*:** Se existe uma quantidade limitada de dados (exemplos) para construir o classificador, fica difícil utilizar a técnica de validação simples, já que retirar uma porcentagem dos dados pode significar ter poucos dados para treinamento. Na técnica de *k-fold cross-validation*, o conjunto S de dados é dividido aleatoriamente em k partições $S_1, ..., S_k$ disjuntas, sendo todas as partições de conjuntos de dados de aproximadamente o mesmo tamanho. Após, são executadas k iterações de indução e teste de um classificador. Na primeira iteração, é induzido o classificador $\mathbf{h}_1$ com a união dos conjuntos de dados $S_2, ..., S_k$, sendo depois testado com o conjunto S_1, obtendo assim a taxa de erro $Err(\mathbf{h}_1)$[4] . Na segunda iteração, é induzido o classificador $\mathbf{h}_2$ com os conjuntos $S_1, S_3, ..., S_k$, sendo depois testado com o conjunto S_2, obtendo assim a taxa de erro $Err(\mathbf{h}_2)$, e assim sucessivamente. Tendo em vista que agora existem k estimadores de erro independentes $Err(\mathbf{h}_1), ..., Err(\mathbf{h}_k)$, com eles pode-se estimar a média e o erro padrão do erro, definidos respectivamente pelas Equações 10 e 11, do modelo final, o qual é construído utilizando todos os exemplos disponíveis. Na Figura 2 é ilustrada a técnica de *k-fold cross-validation*, com $k = 3$.

$$m_err(\mathbf{h}) = \frac{1}{k}\sum_{k=1}^{k} Err(\mathbf{h}_k) \tag{10}$$

$$se_err(\mathbf{h}) = \frac{1}{\sqrt{k-1}}\sqrt{\frac{1}{k}\sum_{k=1}^{k}(Err(\mathbf{h}_k) - m_err(\mathbf{h}_k))^2} \tag{11}$$

***K-Fold Cross-Validation* Estratificado:** Esta técnica é semelhante à técnica *k-fold cross-validation*. A diferença está na maneira em que é feita a divisão do conjunto de dados original em k partições. No método de *k-fold cross-validation* estratificado, as partições são feitas de modo que seja respeitada a distribuição dos exemplos nas classes. Ou seja, se no conjunto de dados original existem 22% dos exemplos na classe C_1 e 78% dos exemplos na classe C_2, em cada partição construída com a técnica de *k-fold cross-validation* estratificado

[4]também denominado neste trabalho por $\varepsilon(\mathbf{h}_1)$

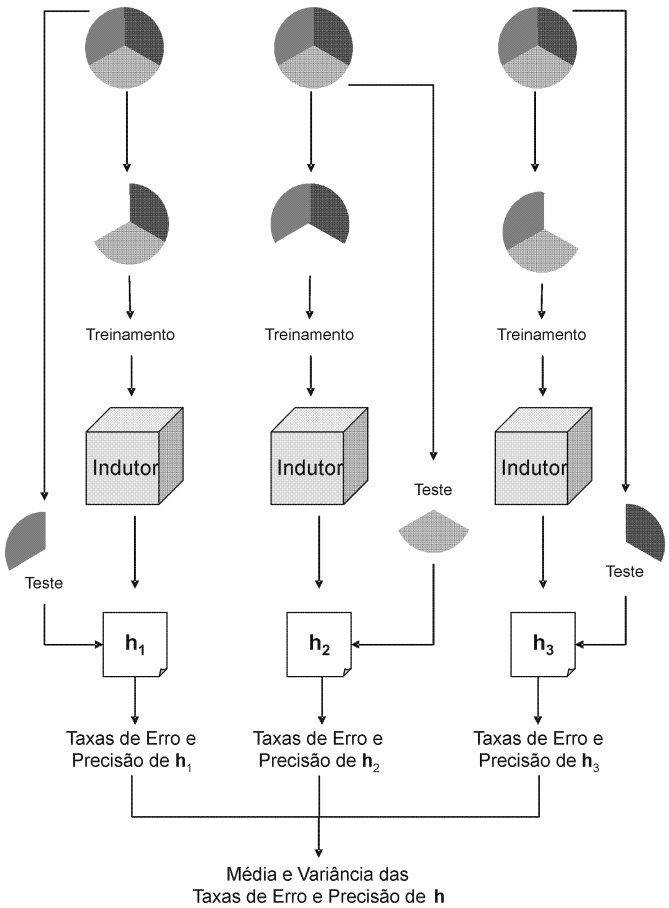

Figura 2: Exemplo de *k-fold cross-validation*, com $k = 3$

existem aproximadamente 22% dos exemplos pertencentes à classe C_1 e 78% dos exemplos pertencentes à classe C_2.

2.4 Avaliação de Regras

Dada uma regra $R = B \rightarrow H$ e um conjunto de exemplos $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, a cobertura de uma regra é definida como segue: considerando uma regra $R = B \rightarrow H$, os exemplos que satisfazem o corpo da regra, ou seja, que satisfazem B, compõem o *conjunto de cobertura* de R, denominado conjunto B; em outras palavras, esses exemplos são *cobertos* por R. Exemplos que satisfazem B e H são *corretamente cobertos* por R, e tais exemplos pertencem ao conjunto $B \cap H$. Exemplos que satisfazem B mas não H são *incorretamente cobertos* pela regra, e pertencem ao conjunto $B \cap \overline{H}$. Por outro lado, exemplos que não satisfazem B *não são cobertos* pela regra, e pertencem ao conjunto $\overline{B}$. Dados uma regra e um conjunto de dados, uma maneira de avaliar a performance dessa regra é calculando a *matriz de contingência* dessa regra (Lavrac et al., 1999), como mostrada na Tabela 4. Denotando a cardinalidade de um conjunto A como a, *i.e.* $a = |A|$, então b e h na Tabela 4 denotam respectivamente o número de exemplos pertencentes aos conjuntos B e H, *i.e.* $b = |B|$ e $h = |H|$. Similarmente, $\overline{b} = |\overline{B}|$; $\overline{h} = |\overline{H}|$; $bh = |B \cap H|$; $\overline{b}h = |\overline{B} \cap H|$; $b\overline{h} = |B \cap \overline{H}|$; e $\overline{bh} = |\overline{B} \cap \overline{H}|$.

	H	$\overline{H}$	
B	hb	$\overline{h}b$	b
$\overline{B}$	$h\overline{b}$	$\overline{hb}$	$\overline{b}$
	h	$\overline{h}$	N

onde:

hb: Número de exemplos para os quais H é verdadeira e B é verdadeiro
$\overline{h}b$: Número de exemplos para os quais H é falsa e B é verdadeiro
$h\overline{b}$: Número de exemplos para os quais H é verdadeira e B é falso
$\overline{hb}$: Número de exemplos para os quais H é falsa e B é falso
b: Número de exemplos para os quais B é verdadeiro
$\overline{b}$: Número de exemplos para os quais B é falso
h: Número de exemplos para os quais H é verdadeira
$\overline{h}$: Número de exemplos para os quais H é falsa
N: Número total de exemplos

Tabela 4: Matriz de contingência de uma regra

Quando se consideram regras pertencente a um classificador **h**, para calcular tal matriz, deve ser considerado o procedimento utilizado pelo algoritmo de AM para induzir as regras que constituem o classificador simbólico **h**. Os procedimentos de indução de regras utilizados pelos algoritmos de AM induzem basicamente 3(três) tipos de conjuntos de regras:

1. Regras não-ordenadas (*Unordered*);
2. Regras ordenadas (*Ordered*); e
3. Regras ordenadas entre as classes (*Interclass*)

descritos a seguir.

Regras não-ordenadas

Para um conjunto de regras não-ordenadas, essas informações são calculadas verificando a cobertura de cada regra do classificador simbólico $\mathbf{h} = \{R_1, ..., R_{N_R}\}$ no conjunto de exemplos S. No Algoritmo 1 é mostrado como é realizado o cálculo dessas informações para cada regra de um conjunto de regras não-ordenadas.

No Algoritmo 1, L é uma lista de quatro elementos, os quais armazenam os valores de hb, $\overline{h}b$, $\overline{hb}$ e $h\overline{b}$, respectivamente, que são os elementos da matriz de contingência da regra. A função $cobertura(R_u, (\mathbf{x}_i, y_i))$ retorna um número inteiro no intervalo $1, ..., 4$, o qual é usado como índice para incrementar o elemento correspondente da lista L. O índice 1 corresponde a hb, 2 a $\overline{h}b$, 3 a $\overline{hb}$ e 4 a $h\overline{b}$.

Regras ordenadas

No caso de regras ordenadas, existe um "**else**" implícito entre cada regra que constitui o classificador. Assim, a cobertura do exemplo deve ser analisada a partir da primeira regra, até que uma delas cubra esse exemplo (B é verdadeiro para o exemplo). Esse exemplo é coberto apenas por essa regra. Para as regras seguintes àquela que cobre o exemplo, as informações devem ser atualizadas incrementando-se os valores de $h\overline{b}$ e $\overline{hb}$ conforme

Algoritmo 1 Cálculo de informações para avaliação de regras não-ordenadas

Require: $\mathbf{h} = \{R_1, ..., R_{N_R}\}$: Conjunto de regras não-ordenadas
$S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$: Conjunto de exemplos
$L := [0, 0, 0, 0]$;
for all $R_u \in \mathbf{h}$ **do**
  for all $(\mathbf{x}_i, y_i) \in S$ **do**
    $j := cobertura(R_u, (\mathbf{x}_i, y_i))$;
    $L[j] := L[j] + 1$;
  end for
end for
return L

a classe C_v do exemplo. No Algoritmo 2 é mostrado como é realizado o cálculo dessas informações para um conjunto de regras ordenadas. A variável $first$ está associada a cada exemplo $(\mathbf{x}_i, y_i)$ e recebe $false$ se o exemplo não foi coberto por uma regra ou $true$ se já foi coberto por uma regra

Algoritmo 2 Cálculo de informações para avaliação de regras ordenadas

Require: $\mathbf{h} = \{R_1, ..., R_{N_R}\}$: Conjunto de regras ordenadas
$S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$: Conjunto de exemplos
$L := [0, 0, 0, 0]$;
$first := false$;
for all $R_u \in \mathbf{h}$ **do**
  for all $(\mathbf{x}_i, y_i) \in S$ **do**
    $j := cobertura(R_u, (\mathbf{x}_i, y_i))$;
    if $first = true$ and ($j = 1$ or $j = 2$) **then**
      $j := 4 - j + 1$; {o exemplo já foi coberto por uma regra. Assim, o elemento j deverá ser igual a 3 ou 4 }
    end if
    if $first = false$ and ($j = 1$ or $j = 2$) **then**
      $first := true$;
    end if
    Incrementar em 1 o elemento j da lista L;
  end for
end for
return L

Regras ordenadas entre as classes

As regras ordenadas entre as classes são separadas em N_{Cl} blocos para cada uma das N_{Cl} diferentes classes, existindo um "**else**" implícito entre cada um desses blocos. Para regras em um mesmo bloco C_v, as informações são calculadas como descrito anteriormente para regras não-ordenadas. Ao ser encontrada uma regra que cubra o exemplo em um bloco C_v, as informações extraídas das regras subseqüentes devem ser atualizadas como descrito anteriormente para regras ordenadas, nos $C_{N_{Cl}-v}$ blocos seguintes.

Por exemplo, a Figura 3 mostra que, para cada classe, existe um conjunto de subconjuntos de regras $\mathbf{h}' = \{\mathbf{h}'_1, ..., \mathbf{h}'_{N_{Cl}}\}$[5]. É importante observar que, além do **else** implícito entre

[5]Observar que $\mathbf{h}' = \mathbf{h}$ - *default*. Como mencionado, a regra *default* é uma regra especial cujo corpo é vazio e

$$\mathbf{h}'_1 \quad \begin{cases} R_1 : Corpo \text{ then CLASS } = \alpha \\ R_2 : Corpo \text{ then CLASS } = \alpha \end{cases}$$

else

$$\mathbf{h}'_2 \quad \begin{cases} R_3 : Corpo \text{ then CLASS } = \beta \\ R_4 : Corpo \text{ then CLASS } = \beta \\ R_5 : Corpo \text{ then CLASS } = \beta \end{cases}$$

else

$$\vdots$$

$$\mathbf{h}'_{N_{Cl}} \quad \begin{cases} R_{u-2} : Corpo \text{ then CLASS } = \gamma \\ R_{u-1} : Corpo \text{ then CLASS } = \gamma \end{cases}$$

com $\alpha, \beta, \gamma \in C = C_1, ..., C_{N_{Cl}}$.

Figura 3: Regras ordenadas entre classes — conjunto de subconjuntos de regras

cada um desses subconjuntos de regras, esses subconjuntos são formados conforme a ordem em que o algoritmo de AM induz as regras correspondentes às classes que compõem a hipótese $\mathbf{h}$. Cada elemento de $\mathbf{h}'$, ou seja, $\mathbf{h}'_1, ..., \mathbf{h}'_{N_{Cl}}$, é uma hipótese constituída por um subconjunto de regras que predizem a mesma classe.

Assim, o exemplo deve ser aplicado a todas as regras de $\mathbf{h}'_1$ e as informações são calculadas como descrito anteriormente para regras não-ordenadas. Se nenhuma regra de $\mathbf{h}'_1$ cobrir o exemplo, ele deve ser aplicado a $\mathbf{h}'_2$ e assim por diante. Ao ser encontrada uma regra em um $\mathbf{h}'_v$ que cobre o exemplo, as informações extraídas das regras subseqüentes são atualizadas como descrito anteriormente para regras ordenadas, nos $\mathbf{h}'_{N_{Cl}-v}$ subconjuntos seguintes.

Após determinar as informações para cada uma das regras R_u[6] de um classificador simbólico $\mathbf{h}$, os valores encontrados podem ser expressos sob a forma de freqüências relativas, as quais são utilizadas neste trabalho como uma estimativa de probabilidade. Por exemplo, a probabilidade $P(HB)$ pode ser determinada da seguinte forma:

$$P(HB) = f_{bh} = \frac{hb}{N}$$

onde N é o número total de exemplos no conjunto de exemplos S.

De forma semelhante podem ser determinados os valores das probabilidades $P(\overline{H}B), P(\overline{HB})$ e $P(H\overline{B})$. Conhecidas essas probabilidades, os valores de $P(B)$, $P(\overline{B})$, $P(H)$ e $P(\overline{H})$ podem ser determinados da seguinte forma:

$$P(B) = P(\overline{H}B) + P(HB) \quad ou \quad P(\overline{B}) = P(H\overline{B}) + P(\overline{HB})$$

Tais probabilidades são utilizadas para calcular as medidas de avaliação de regras. Várias medidas de regras têm sido propostas na literatura (Lavrac et al., 1999; Freitas, 1998b,a; Hil-

a cabeça é a classe de maior freqüência no conjunto de exemplos. Assim, no caso de não existir uma regra que cubra algum exemplo, uma hipótese $\mathbf{h}$ irá classificar esse exemplo devido à existência da regra *default*.

[6]Se R_u é a regra *default*, então as informações não são calculadas para essa regra.

derman and Hamilton, 1999). Neste trabalho, serão utilizadas tanto medidas de avaliação de regras unificadas em um *framework* proposto por Lavrac et al. (1999) quanto outras medidas encontradas na literatura que não estão presentes nesse *framework*. No Anexo A, encontra-se uma descrição de medidas de avaliação de regras.

2.5 Regras de Associação

Na terminologia de regras de associação, dada uma base de dados S no formato atributo-valor, cada exemplo é denominado transação e cada atributo seguido de um valor (válido) é denominado item. Uma regra de associação é composta por duas partes: antecedente (*left-hand side*) e conseqüente (*right-hand side*) na forma

$$LHS \Rightarrow RHS$$

onde ambos antecedente e conseqüente são compostos por um conjunto de itens — *itemset* — que freqüentemente ocorrem juntos em S e $LHS \cap RHS = \emptyset$. A tarefa de descoberta de regras de associação é determinística e bem definida. Foi desenvolvida inicialmente a partir da análise dos itens presentes em compras de supermercados, com o objetivo de identificar relações do tipo "*um cliente que compra os produtos* $pr_1, pr_2, ..., pr_i$ *também irá comprar os produtos* $pr_k, pr_q, ..., pr_j$ *com probabilidade* $c\%$" (Agrawal and Srikant, 1994). Em termos gerais, uma regra de associação $LHS \Rightarrow RHS$ determina o quanto a presença de um conjunto de itens It_1 nos registros de uma base de dados implica na presença de algum outro conjunto distinto de itens It_2 nos mesmos registros. A regra $LHS \Rightarrow RHS$ ocorre no conjunto de transações S com confiança $conf$ se em $conf\%$ das transações de S em que ocorre LHS ocorre também RHS. A regra $LHS \Rightarrow RHS$ tem suporte sup se em $sup\%$ das transações em S ocorre $LHS \cup RHS$. Os algoritmos utilizados para descobrir essas regras procuram em todo o conjunto de regras descobertas por aquelas regras que possuem um critério de qualidade (confiança $conf$ e suporte sup) maior que o limite inferior especificado pelo usuário.

Ainda que é utilizado o modo de aprendizado não-supervisionado para obter regras de associação, como cada conjunto de itens pode ser interpretado como um conjunto de condições atributo-valor, regras de classificação podem ser derivadas de um conjunto de regras de associação filtrando desse conjunto todas as regras cujo conseqüente possui somente um item referente a um específico atributo alvo (atributo classe). Esse tipo de regras de classificação serão também utilizadas neste trabalho.

Na próxima seção, é feita uma descrição do paradigma de Algoritmos Evolutivos, foco deste trabalho.

3 Algoritmos Evolutivos

O paradigma de algoritmos evolutivos consiste de algoritmos de busca estocástica baseados na abstração dos processos de evolução Darwiniana. Ainda que diversos tipos de algoritmos evolutivos (como discutido mais adiante nesta seção) têm sido propostos, quase todos eles têm alguns elementos e características básicas em comum (Freitas, 2002a), tais como:

- AEs tipicamente trabalham com uma população de indivíduos (soluções candidatas) a

cada iteração, ao invés de uma única solução candidata por vez;

- Usam um método de seleção baseado na aptidão do indivíduo, ou seja, uma medida de qualidade da solução candidata representada por um indivíduo. Assim, quanto melhor a aptidão do indivíduo, maior a probabilidade do indivíduo ser selecionado e maior a chance de algumas partes de seu "material genético"(partes da solução candidata) ser passada para as gerações futuras de indivíduos;

- Geram novos indivíduos através de um mecanismo de herança dos indivíduos existentes. Descendentes dos indivíduos são gerados aplicando-se operadores estocásticos (probabilísticos) nos indivíduos existentes na geração corrente. Dois operadores freqüentemente utilizados são *crossover* (recombinação) e *mutação*. Em essência, o operador de *crossover* mistura o material genético de dois ou mais indivíduos, enquanto que o operador de mutação muda o valor de uma pequena parte do material genético de um indivíduo para um valor aleatório, simulando a auto-replicação errônea dos indivíduos.

Segundo (Freitas, 2002a), existem quatro sub-paradigmas de AEs:

1. **Estratégias Evolutivas**: tipicamente utilizam um vetor de valores reais para representar os indivíduos. As primeiras estratégias evolutivas deram ênfase ao operador de mutação como o principal operador de exploração no espaço de busca, mas atualmente tanto mutação quanto *crossover* são utilizados. Um indivíduo geralmente representa não somente variáveis com valores reais do problema a ser resolvido mas também parâmetros que controlam a distribuição de mutação, caracterizando uma auto-adaptação dos parâmetros de mutação. O operador de mutação geralmente modifica os indivíduos segundo uma distribuição normal multi-variada, na qual pequenas mutações têm preferência em relação a grandes mutações.

2. **Programação Evolutiva**: foi originalmente desenvolvida para evoluir máquinas de estado finito. Diferentemente das estratégias evolutivas, programação evolutiva geralmente não utiliza *crossover*; similarmente às estratégias evolutivas, também utiliza mutação com probabilidade segundo uma distribuição normal e auto-adaptação dos parâmetros de mutação.

3. **Algoritmos Genéticos (AGs)**: dão ênfase ao operador de *crossover* como principal operador de exploração no espaço de busca e considera a mutação como um operador menor, tipicamente aplicado com probabilidade muito baixa. No início, os AGs clássicos utilizavam cadeias de bits para representar os indivíduos mas, atualmente, representações mais elaboradas são também utilizadas.

4. **Programação Genética**: bastante semelhante aos AGs, entretanto, os indivíduos são geralmente algum tipo de programa de computador, ou seja, consistem não somente de estruturas de dados mas também de funções (ou operações) aplicáveis a essas estruturas. A idéia de ter os indivíduos explicitamente representando tanto dados quanto funções e operadores (ao invés de somente parâmetros de funções) parece ser relativamente pouco usado nos outros sub-paradigmas de AEs.

Neste trabalho, propomos o uso de AGs para evoluir classificadores simbólicos. Assim, a seguir são descritos os AGs, dando ênfase ao funcionamento do AG clássico. Na seqüência, é feita uma descrição das características de AGs utilizados para evoluir classificadores simbólicos.

3.1 Algoritmos Genéticos

AGs são métodos de otimização e busca inspirados nos mecanismos de evolução de populações de seres vivos. Foram introduzidos por John Holland (Holland, 1975) e popularizados por um dos seus alunos, David Goldberg (Goldberg, 1989). Esses algoritmos seguem o princípio da seleção natural e sobrevivência do mais apto, enunciado em 1859 pelo naturalista e fisiologista inglês Charles Darwin em seu livro **A Origem das Espécies**. De acordo com Charles Darwin,

> *Quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes.*

A otimização é a busca da melhor solução para um dado problema. Consiste em tentar várias soluções e utilizar a informação obtida nesse processo de forma a encontrar soluções cada vez melhores. As técnicas de busca e otimização geralmente apresentam:

1. Um espaço de busca, onde estão todas as possíveis soluções do problema;
2. Uma função de aptidão (ou função objetivo), que é utilizada para avaliar as possíveis soluções, associando a cada uma delas uma nota.

Em termos matemáticos, a otimização consiste em achar a solução ótima da função objetivo, ou seja, a solução que corresponda ao ponto de máximo ou mínimo da função objetivo. Entretanto, em muitas funções, existem várias soluções ótimas locais (pontos ótimos locais), ou seja, soluções que otimizam o valor da função objetivo. Tais pontos são denominados *ótimos locais* porque a função atinge, nesses pontos, valores ótimos em relação à vizinhança desses pontos. Entretanto, a melhor solução para esse problema está no ponto em que a função atinge seu "melhor" valor ótimo, denominado *ótimo global*. Em muitas funções, uma grande quantidade de técnicas de otimização (por exemplo, os métodos de gradiente) não são capazes de localizar o ponto de ótimo global de uma função com múltiplos pontos ótimos. Nesses casos, AGs geralmente são capazes de encontrá-los.

A terminologia descrita a seguir foi baseada em (Mitchell, 1997a). Em AGs, necessita-se de uma estrutura de dados que codifica uma solução para um problema que se deseja resolver, ou seja, uma representação para um ponto no espaço de busca. Tal estrutura é denominada *genoma* e tal ponto no espaço é denominado *cromossomo*. A união de um cromossomo com sua aptidão é denominada *indivíduo*. Um parâmetro codificado no cromossomo é denominado *gene*, e aos valores que um gene pode assumir dá-se o nome de *alelo*. O *genótipo* representa a informação contida no cromossomo (ou genoma). Quando se decodifica o cromossomo, essa decodificação recebe o nome de *fenótipo*. Por exemplo, se um cromossomo codifica parâmetros de um conjunto de regras, o fenótipo é o conjunto de regras; se um cromossomo codifica os parâmetros de uma rede neural, o fenótipo é a rede neural.

Existem problemas nos quais há uma interação entre genes do cromossomo, ou seja, um valor de gene influencia o valor de outro gene. A essa interação, dá-se o nome de *epistasia*. Deve ser observado que problemas com alta epistasia são difíceis de se resolver utilizando AGs.

A seguir, é descrito o algoritmo genético clássico, bem como é feita uma descrição mais detalhada de alguns componentes de um AG clássico. É também mostrado um exemplo de funcionamento de um AG clássico e são feitas algumas considerações em relação ao uso de AGs.

3.1.1 Algoritmo Genético Clássico

Dados:

- Um espaço de busca;
- Uma função de aptidão;
- Um conjunto de operadores genéticos a serem aplicados aos cromossomos para criar uma nova população. Esse conjunto geralmente é composto pelos operadores de *seleção*, *crossover* e *mutação*. O operador de seleção seleciona os cromossomos mais aptos para produzir uma população de genitores, o operador de *crossover* produz um ou mais cromossomos filhos a partir da recombinação dos genes de seus pais (população de descendentes), e o de *mutação* substitui o valor de um ou mais locus de um cromossomo por outro valor. Tais operadores são melhor descritos na Seção 3.1.2;

o algoritmo genético clássico segue os passos descritos no Algoritmo 3.

Algoritmo 3 Algoritmo genético clássico.

Require: $\mathcal{D}$: Espaço de busca;
$f(\mathbf{i})$: Função de avaliação utilizada para avaliar cada indivíduo $\mathbf{i}$;
cross, *mut* e *select*: Operadores de *crossover*, mutação e seleção.

1: $current_population = random_individual(N)$;
2: **while** critério de parada não satisfeito **do**
3: **for all** $\mathbf{i} \in current_population$ **do**
4: Calcular $f(\mathbf{i})$;
5: **end for**
$new_population = \emptyset$;
6: **for** $i = 1$ to N **do**
7: $(\mathbf{i}_1, \mathbf{i}_2) = select(current_population)$;
8: $\mathbf{i}'_i = cross(\mathbf{i}_1, \mathbf{i}_2)$;
9: $\mathbf{i}_i = mut(\mathbf{i}'_i)$;
10: $new_population = new_population + \{\mathbf{i}_i\}$;
11: **end for**
12: $current_population = new_population$;
13: **end while**
14: **return** $best_individual(current_population)$;

No Algoritmo 3, a função *random_individual* retorna N indivíduos gerados aleatoriamente, pertencentes ao espaço de busca $\mathcal{D}$, e a função *best_individual* retorna o melhor indivíduo da última população gerada pelo AG segundo a função de avaliação $f(\mathbf{i})$.

Podem existir diversos critérios de parada, tais como não ocorrer alterações da população anterior para a população atual ou alcançar um número pré-fixado de gerações. *Geração* é o nome dado a cada iteração do algoritmo descrito. Geralmente, o número de gerações de um AG típico é de 50 a 500. Ainda, pode-se fixar um número maior de gerações. O conjunto completo de iterações é denominado *execução*[7]. Ao final de uma execução, há um ou mais cromossomos com alta aptidão na população. Duas execuções diferentes utilizando sementes aleatórias distintas

[7] *run.*

contém detalhes distintos devido à aleatoriedade amplamente utilizada. Na Figura 4 é ilustrada uma geração de um AG, com início em população não avaliada.

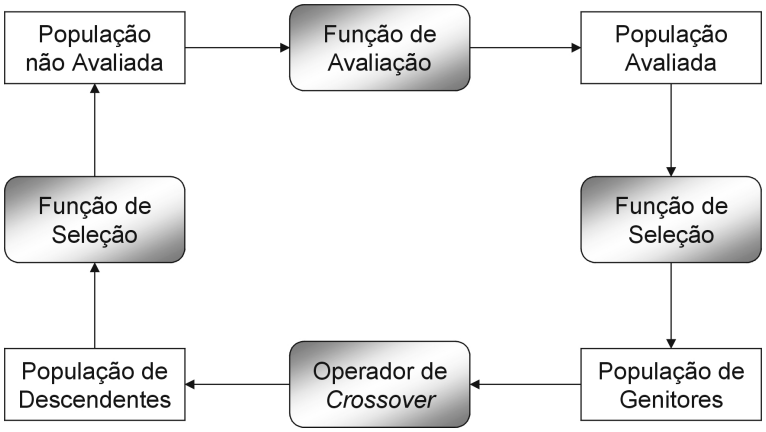

Figura 4: Uma geração de um algoritmo genético (Milaré, 2003)

3.1.2 Componentes de um AG Clássico

Nesta seção são descritos em maiores detalhes a função de aptidão, o método de seleção de indivíduos, os operadores genéticos e os valores dos parâmetros utilizados pelos AGs. Esta descrição está baseada no trabalho de Milaré (2003).

Função de Aptidão: A função de aptidão[8] determina a aptidão de cada indivíduo, ou seja, quão boa a solução por ele representada é para a resolução de um problema. Ela é responsável por determinar se o indivíduo continuará fazendo parte da população ou não. A escolha da função de aptidão, também denominada função de avaliação, está diretamente relacionada ao domínio do problema.

Método de Seleção: Uma vez que os AGs baseiam-se no princípio da seleção natural, eles devem ser capazes de identificar os indivíduos mais aptos, para que eles permaneçam na população durante o processo de evolução (procedimento iterativo), e os mais fracos, para que eles sejam excluídos do processo. Existem vários métodos que podem ser utilizados para selecionar indivíduos para formar uma nova população (Michalewicz, 1999; Freitas, 2002a; Carvalho et al., 2003). Um critério de seleção bastante comum em AG é a seleção proporcional, que consiste em selecionar os indivíduos com probabilidade proporcional à sua aptidão. Conceitualmente, a seleção proporcional é equivalente a dividir uma roleta em n partes, sendo n o número de indivíduos da população. Cada uma das partes da roleta é proporcional à aptidão do indivíduo associado àquela parte. A roleta é girada n vezes, e a cada uma delas o indivíduo indicado pelo ponteiro é selecionado e inserido na nova população.

Com o objetivo de preservar e usar na próxima geração as melhores soluções encontradas na geração atual, pode-se utilizar uma estratégia denominada *elitismo*. Essa estratégia consiste em conservar os m melhores indivíduos da população atual, copiando-os para a

[8] *Fitness*.

próxima geração sem nenhuma alteração. O restante dos indivíduos, $n - m$, são normalmente gerados por meio do método de seleção.

Operadores Genéticos: Os operadores genéticos quando aplicados à população geram novos indivíduos e constituem os principais mecanismos utilizados pelos AGs para explorar regiões desconhecidas do espaço de busca. Deve ser observado que *crossover* e mutação são os operadores genéticos mais utilizados e fazem parte do AG clássico.

- ***Crossover***: Esse operador genético permite a troca de informação entre soluções diferentes. Ele combina características de dois indivíduos pais para formar dois indivíduos filhos por meio da troca de segmentos correspondentes dos indivíduos pais. Por exemplo, considere os seguintes indivíduos pais representados por uma cadeia de 0's e 1's de comprimento 8:

$$\begin{array}{ll} pai_1 & \mathit{0\ 0\ 1\ 0\ 0\ 0\ 0\ 0} \\ pai_2 & 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \end{array}$$

  aplicando o operador de *crossover* depois da terceira posição:

$$\begin{array}{ll|l} pai_1 & \mathit{0\ 0\ 1} & \mathit{0\ 0\ 0\ 0\ 0} \\ pai_2 & 1\ 0\ 0 & 1\ 1\ 1\ 1\ 1 \end{array}$$

  são gerados os seguintes indivíduos filhos:

$$\begin{array}{ll} filho_1 & \mathit{0\ 0\ 1}\ 1\ 1\ 1\ 1\ 1 \\ filho_2 & 1\ 0\ 0\ \mathit{0\ 0\ 0\ 0\ 0} \end{array}$$

- **Mutação**: Esse operador genético introduz variabilidade dentro da população e consiste em arbitrariamente alterar o valor de um ou mais genes de um indivíduo. Por exemplo, considere o seguinte indivíduo representado por uma cadeia de 0's e 1's de comprimento 8:

$$1\ 0\ 0\ 0\ 0\ 0\ 0\ 1$$

  aplicando o operador de mutação na quarta posição, o seguinte indivíduo é gerado:

$$1\ 0\ 0\ \mathit{1}\ 0\ 0\ 0\ 1$$

Valores de Parâmetros: Um AG utiliza vários parâmetros, entre eles: o tamanho da população, o número de genes dos indivíduos, a probabilidade de aplicação dos operadores genéticos, o número máximo de gerações, e outros. Os valores para esses parâmetros são definidos geralmente por tentativa e erro.

3.1.3 Um Exemplo

Para ilustrar o funcionamento de um AG, considere um exemplo bem simples apresentado em (Mitchell, 1997a), no qual os indivíduos são representados por um vetor binário de comprimento 8 e a função de aptidão é igual ao número de 1's do indivíduo. Os outros parâmetros são: tamanho da população $n = 4$, probabilidade de *crossover* $p_c = 0.5$ e probabilidade de mutação $p_m = 0.1$. Todos os valores dos parâmetros do AG, assim como a função de aptidão, foram escolhidos para simplificar o exemplo.

Inicialmente, os indivíduos da população são criados inicializando aleatoriamente todos os 8 elementos do vetor que representam cada gene do indivíduo. Considere que os seguintes quatro indivíduos formam a população inicial:

$$\begin{aligned} A =& (0\ 0\ 0\ 0\ 0\ 1\ 1\ 0) \\ B =& (1\ 1\ 1\ 0\ 1\ 1\ 1\ 0) \\ C =& (0\ 0\ 1\ 0\ 0\ 0\ 0\ 0) \\ D =& (0\ 0\ 1\ 1\ 0\ 1\ 0\ 0) \end{aligned}$$

O próximo passo consiste em calcular a função de aptidão para cada indivíduo. Lembrando que a função de aptidão, para esse exemplo, é o número de 1's presentes em cada indivíduo, o valor da função de aptidão para os quatro indivíduos é:

$$\begin{aligned} A =& (0\ 0\ 0\ 0\ 0\ 1\ 1\ 0) = 2 \\ B =& (1\ 1\ 1\ 0\ 1\ 1\ 1\ 0) = 6 \\ C =& (0\ 0\ 1\ 0\ 0\ 0\ 0\ 0) = 1 \\ D =& (0\ 0\ 1\ 1\ 0\ 1\ 0\ 0) = 3 \end{aligned}$$

Deve-se agora selecionar alguns indivíduos para formar uma nova população. Considere que, utilizando a seleção proporcional, foram selecionados para formar a nova população os seguintes indivíduos (os identificadores entre parênteses referem-se aos identificadores dos indivíduos selecionados da população anterior):

$$\begin{aligned} A' =& (1\ 1\ 1\ 0\ 1\ 1\ 1\ 0)(B) \\ B' =& (0\ 0\ 1\ 1\ 0\ 1\ 0\ 0)(D) \\ C' =& (1\ 1\ 1\ 0\ 1\ 1\ 1\ 0)(B) \\ D' =& (0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)(C) \end{aligned}$$

Note que o indivíduo A não foi selecionado, enquanto que o indivíduo B foi selecionado duas vezes (B era o indivíduo com maior probabilidade de ser selecionado, já que possuía maior valor da função de aptidão). Agora pode-se aplicar o operador genético *crossover* na nova população. Com probabilidade de *crossover* $p_c = 0.5$, espera-se que, na média, 50% dos indivíduos, ou seja, dois indivíduos, sofram *crossover* em cada geração. Considere que o operador de *crossover*

seja aplicado nos indivíduos A' e B' depois da primeira posição, formando os indivíduos $A'' =$ 10110100 e $B'' =$ 01101110. A população atual agora é formada pelos seguintes indivíduos:

$$\begin{aligned} A'' =&(1\ 0\ 1\ 1\ 0\ 1\ 0\ 0) \\ B'' =&(0\ 1\ 1\ 0\ 1\ 1\ 1\ 0) \\ C' =&(1\ 1\ 1\ 0\ 1\ 1\ 1\ 0) \\ D' =&(0\ 0\ 1\ 0\ 0\ 0\ 0\ 0) \end{aligned}$$

O próximo operador genético a ser aplicado é o operador de mutação, com probabilidade $p_m = 0.1$. Espera-se, assim, que 10%, na média, dos elementos (genes) de cada vetor (indivíduo) sofram mutacão. Neste caso, há $4 \times 8 = 32$ genes na população, portanto, são esperadas 3.2 mutações, em média, em cada geração. Considere que o indivíduo A'' seja mutado no sexto gene, o indivíduo C' no segundo gene e o indivíduo D' no primeiro gene. A população agora é constituída pelos seguintes indivíduos:

$$\begin{aligned} A'' =&(1\ 0\ 1\ 1\ 0\ \mathit{0}\ 0\ 0) \\ B'' =&(0\ 1\ 1\ 0\ 1\ 1\ 1\ 0) \\ C' =&(1\ \mathit{0}\ 1\ 0\ 1\ 1\ 1\ 0) \\ D' =&(\mathit{1}\ 0\ 1\ 0\ 0\ 0\ 0\ 0) \end{aligned}$$

Com isso, uma geração é completada. Note que o melhor indivíduo foi perdido durante essa geração, mas a aptidão média da população aumentou de $\frac{12}{4}$ (12 valores 1's presentes nos quatro primeiros indivíduos — A, B, C e D) para $\frac{15}{4}$ (15 valores 1's presentes nos quatro últimos indivíduos — A'', B'', C' e D').

O algoritmo genético clássico descrito é a base para a maioria das aplicações de AGs. Entretanto, em muitas casos, existe a necessidade de codificar os indivíduos de outra forma diferente da codificação com cadeias de bits. Além disso, em muitas aplicações é necessário realizar modificações nos operadores genéticos de mutação e *crossover*, além de muitas vezes ser necessário acrescentar novos operadores genéticos. Na próxima seção, são descritos alguns métodos de AGs que podem ser utilizados para evoluir classificadores simbólicos, objeto de estudo deste trabalho.

3.2 Algoritmos Genéticos Aplicados na Evolução de Classificadores Simbólicos

AGs é um dentre uma classe de paradigmas para simular a evolução utilizando os conceitos de evolução de Darwin para, de forma iterativa, gerar soluções apropriadas (indivíduos) incrementalmente em um ambiente estático ou que muda dinamicamente. Como mencionado, o AG clássico utiliza cadeias de bits de tamanho fixo para codificar os indivíduos e somente dois operadores para gerar aleatoriedade na população corrente de cada geração: o operador de *crossover* e o de mutação. Devido à dificuldade de se aplicar o algoritmo genético clássico, descrito anteriormente, a todo tipo de problema, várias implementações distintas foram propostas para problemas particulares. Aliás, uma representação natural de uma solução potencial para um dado problema somada à família de operadores "genéticos" aplicáveis poderia ser mais útil na

aproximação de soluções de muitos problemas. Assim, vários pesquisadores têm explorado o uso de outras representações para codificar indivíduos, tais como listas ordenadas, listas embutidas, listas de elementos variáveis, cadeias de tamanhos variáveis (incluindo cadeias cujos elementos são regras *if-then-else*), estruturas mais complexas do que cadeias de bits, bem como o uso de operadores genéticos modificados para atender as necessidades de aplicações particulares (Michalewicz, 1999).

O principal objetivo de aprendizado de máquina envolve desenvolver e implementar algoritmos capazes de encontrar novo conhecimento ou melhorar conhecimento previamente construído, utilizando como informação de entrada observações (casos ou exemplos) do mundo. Nos últimos anos tem incrementado o interesse de aplicar técnicas de computação evolutiva em aprendizado de máquina devido à idéia atrativa de cromossomos, representando conhecimento, serem tratados como dados a serem manipulados por operadores genéticos, e, ao mesmo tempo, serem códigos executáveis para serem utilizados na realização de alguma tarefa.

No caso de aprendizado simbólico, *i.e.* aprendizado (indução) de classificadores simbólicos, as seguintes duas abordagens têm sido propostas pela comunidade de AGs[9] (Freitas, 2002a,b):

Pittsburgh – cada indivíduo codifica um conjunto de regras de conhecimento. Por ter sido inicialmente proposto por De Jong e Smith na Universidade de Pittsburgh, essa abordagem também recebeu o nome da universidade de origem, mas também é referenciada na literatura por "abordagem Pitt".

Michigan – um indivíduo codifica uma única regra de conhecimento. Por ter sido inicialmente proposta por Holland e Reitman na década de 70 na Universidade de Michigan, essa abordagem recebeu o nome da universidade de origem.

A escolha entre uma dessas duas abordagens depende do tipo de regras de conhecimento que se quer descobrir. Supondo que o interesse seja classificação — Seção 2.1 na página 2 — , geralmente há um maior interesse na qualidade do conjunto de regras como um todo do que na qualidade das regras individualmente, ou seja, a interação entre as regras é importante. Neste caso, a abordagem Pittsburgh parece mais natural (Freitas, 2002b).

Por outro lado, a abordagem Michigan parece mais natural quando se quer encontrar um pequeno conjunto de regras com alto nível de predição (ou com algum outro valor desejado utilizando alguma medida de avaliação de regras — Seção 2.4 na página 7), e cada regra é avaliada independentemente de outras regras. Usos dessa abordagem são encontrados em (Noda et al., 1999; Weiss and Hirsh, 1998).

Também, deve ser levada em conta a complexidade de codificação e de cálculos a serem realizados pelo AG. Na abordagem Michigan os indivíduos, os quais representam somente uma regra, são simples e relativamente pequenos. Esses fatores tendem a reduzir a complexidade do cômputo da função objetivo e a simplificar o projeto dos operadores genéticos. Todavia, essa vantagem tem um custo. Em primeiro lugar, desde que a função objetivo avalia a qualidade de cada regra individualmente, não é fácil computar a qualidade de um conjunto de regras, ou seja, considerar interações entre as regras no cômputo da função objetivo. Outro problema é impedir que o AG convirja a um indivíduo único quando se quer descobrir um conjunto de regras (indivíduos), e não uma única regra. Existem duas maneiras de resolver esse problema:

[9] Em (Michalewicz, 1999), dá-se o nome de computação evolutiva quando se altera a maneira de codificar os indivíduos. Já em (Freitas, 2002a), não existe essa distinção. Neste trabalho, adotamos as definições propostas em (Freitas, 2002a).

1. Utilizar um método de *niching*. Os métodos de *niching* fazem com que a população inicial convirja para um subgrupo (subpopulação) e não para um único indivíduo. Assim, esses métodos proporcionam uma extensão aos AGs, permitindo-os trabalhar em domínios que requerem múltiplas soluções, bem como manutenção nessas soluções (Mahfoud, 1995). Problemas de classificação e aprendizado de máquina estão incluídos nessa gama de problemas.

2. Executar o AG várias vezes, descobrindo em cada iteração uma regra diferente, o que é computacionalmente caro.

Exemplos de AG para o problema de classificação em aprendizado que utilizam a técnica Michigan são COGIN (Greene and Smith, 1993) e REGAL (Giordana and Neri, 1995). Em contraste a esse cenário, utilizando a abordagem Pittsburgh, essa técnica leva em conta a interação entre as regras no cômputo da função objetivo. Ainda, os indivíduos são sintaticamente longos, muitas vezes levando à necessidade de modificações nas operações genéticas padrões (*crossover* e mutação) para lidar com esses indivíduos relativamente complexos. Exemplos que utilizam a abordagem Pittsburgh nos quais são observadas tais características são GABIL (De Jong et al., 1993), GIL (Janikow, 1993) e HDPDCS (Pei et al., 1997).

O objetivo deste trabalho consiste em: dada uma população inicial composta por classificadores simbólicos (constituídos por conjuntos de regras), construir um classificador simbólico que seja no mínimo tão preciso quanto ou, preferencialmente, mais preciso que o menos preciso dentre todos os classificadores iniciais, utilizando AGs.

Analisando as duas abordagens competitivas para tratarem do problema de utilização de computação evolutiva em aprendizado de máquina, pode-se observar que a abordagem Pittsburgh é a mais indicada para essa tarefa do que a abordagem Michigan, tendo em vista que o objetivo final é construir um conjunto de regras que constituem o classificador final.

Nos trabalhos relacionados com evolução de classificadores simbólicos citados anteriormente, somente encontramos trabalhos que mapeiam os indivíduos em cadeias de bits, o que torna os indivíduos sintaticamente longos, conforme mencionado. Neste trabalho, a sintaxe padrão $\mathcal{PBM}$, proposta no ambiente DISCOVER (descrito na Seção 5 na página 25), permite-nos codificar os indivíduos de maneira automática, *i.e*, sem que seja necessário alguma conversão para cadeias de bits, a única transformação necessária é da linguagem de descrição do modelo do algoritmo que construiu as regras para a sintaxe padrão $\mathcal{PBM}$, fazendo com que os indivíduos sejam sintaticamente menores. Como os indivíduos são sintaticamente menores, a implementação dos operadores genéticos do AG se torna mais fácil, como mostrado na Seção 6 na página 34. Na próxima seção, é apresentada a nossa proposta de um AG para evoluir classificadores simbólicos em um único classificador.

4 Proposta de um Algoritmo Genético para Evoluir Classificadores Simbólicos

O AG proposto neste trabalho segue os passos do AG clássico descrito na Seção 3.1 na página 13. O que varia nesta proposta são os componentes do AG. Utilizando a abordagem Pittsburgh, na qual os indivíduos são codificados como sendo um conjunto de regras, propomos, além de utilizar as operações de crossover e mutação de maneira usual, usar diferentes funções de avaliação. A seguir, são discutidos os componentes do AG proposto neste trabalho.

4.1 Codificação dos Indivíduos

No algoritmo genético proposto neste trabalho, cada gene de um indivíduo (classificador) é uma regra, e cada indivíduo é, portanto, uma seqüência de regras (na verdade, é um conjunto de regras — Seção 2 — que, para vias de implementação, é transformado em uma seqüência de regras). Para facilitar a geração de regras para compor os cromossomos (classificadores), dado um conjunto de exemplos de treinamento, são induzidos classificadores utilizando algoritmos de aprendizado simbólicos — tais como $\mathcal{CN}2$ (Clark and Niblett, 1989), $\mathcal{C}4.5$ e $\mathcal{C}4.5rules$ (Quinlan, 1988) — e um algoritmo de geração de regras de associação — $\mathcal{A}$priori (Borgelt and Kruse, 2002)[10]. Todas as regras induzidas (no caso de indução de classificadores simbólicos) e descobertas (no caso de geração de regras de associação) são transformadas para a sintaxe padrão de regras $\mathcal{PBM}$ (Prati et al., 2002) e armazenadas em uma base de regras, como ilustrado na Figura 5. Cada indivíduo é então codificado como sendo um vetor de identificadores de regras, ilustrado na Figura 6, sendo que cada regra participa somente uma vez em cada indivíduo.

01	**if** ... **then** ...
02	**if** ... **then** ...
	⋮
W	**if** ... **then** ...

Figura 5: Base de regras na sintaxe padrão $\mathcal{PBM}$.

01	10	15	09	07	15

Figura 6: Exemplo de codificação de um indivíduo

Na população inicial, todos os indivíduos possuem o mesmo tamanho. Entretanto, com a aplicação dos operadores genéticos, os indivíduos podem variar de tamanho, conforme descrito mais adiante.

4.2 Operadores Genéticos

São dois os operadores genéticos utilizados no AG proposto neste trabalho:

***Crossover*:** O operador de *crossover* utilizado é o *crossover* assimétrico. Em um *crossover* assimétrico são escolhidas duas posições, uma para cada indivíduo pai. Os indivíduos pais são divididos em dois segmentos, conforme as posições escolhidas. Por fim, são criados os indivíduos filhos por meio da composição dos segmentos dos indivíduos pai. Por exemplo, considere os seguintes indivíduos pai compostos por seqüências de números inteiros, os quais correspondem a índices das regras que constituem esses classificadores, armazenadas na base de regras:

$$\begin{array}{ll} pai_1 & \mathit{07\ 02\ 11\ 00\ 04\ 08\ 03\ 01} \\ pai_2 & 05\ 23\ 10\ 06\ 12\ 09\ 20\ 15 \end{array}$$

[10]Neste trabalho, são utilizadas somente as regras geradas nas quais o lado direito da regra de associação somente possui um valor que é o valor do atributo classe.

O operador de *crossover* assimétrico escolhe duas posições. Por exemplo, suponha que para o indivíduo pai_1 a posição depois do terceiro gene é escolhida, e para o indivíduo pai_2 é escolhida a posição depois do quinto gene.

$$\begin{array}{ll} pai_1 & \mathit{07\ 02\ 11} \text{ — } \mathit{00\ 04\ 08\ 03\ 01} \\ pai_2 & 05\ 23\ 10\ 06\ 12 \text{ — } 09\ 20\ 15 \end{array}$$

Então, como resultado, são gerados os seguintes indivíduos filhos:

$$\begin{array}{ll} filho_1 & 07\ 02\ 11\ 09\ 20\ 15 \\ filho_2 & 05\ 23\ 10\ 06\ 12\ 00\ 04\ 08\ 03\ 01 \end{array}$$

Por meio da seleção realizada a cada geração, o AG pode tender a aumentar ou diminuir o número de regras em cada indivíduo. Dessa forma, um número apropriado de regras para os indivíduos pode ser encontrado para o domínio analisado.

Mutação: A operação de mutação é utilizada de maneira usual: para aplicar a operação de mutação, seleciona-se um gene de um indivíduo (aleatoriamente) e troca-se a regra selecionada por outra selecionada da base de regras disponível. O número de genes selecionados é dado pela probabilidade de mutação p_m.

Além desses operadores, outro operador possível de se utilizar é um operador de generalização/especialização de classificadores. Um indivíduo pode ser generalizado adicionando-se uma regra a ele. Essa operação generaliza um conjunto de regras (classificador) porque o conjunto de regras é uma disjunção de regras. Assim, quando uma regra é inserida no classificador, um disjunto está sendo adicionado ao conjunto, o que em geral tende a aumentar o número de exemplos cobertos pelo classificador. Ao contrário, um conjunto de regras pode ser especializado removendo uma regra do conjunto de regras (Freitas, 2002a). Neste trabalho, esse operador é utilizado após a aplicação das operações de *crossover* e mutação. Para esse operador, é definido pelo usuário um número mínimo de exemplos que uma regra precisa cobrir (min_{cov_R}), um número mínimo de exemplos que o classificador (indivíduo) precisa cobrir ($min_{cov_\mathbf{h}}$) e uma probabilidade de aplicação desse operador (p_e). Assim, dos indivíduos selecionados são excluídas as regras que cobrem menos exemplos que min_{cov_R}, bem como é adicionada uma regra da base de regras caso a cobertura do indivíduo seja menor que $min_{cov_\mathbf{h}}$.

Pretendemos testar o AG proposto variando a função de avaliação e os parâmetros inerentes de um AG (tamanho da população, probabilidade de *crossover* e mutação, número máximo de gerações, entre outros). Como trabalho futuro, pensamos em adicionar ao AG esse operador de generalização/especialização de classificadores.

4.3 Funções de Avaliação

Para avaliar um conjunto de regras, deve-se avaliar o comportamento do conjunto de regras (classificador) sobre um conjunto de exemplos de teste. Assim, pode-se avaliar o comportamento

do conjunto de regras tanto como uma "caixa preta" quanto fazer uma avaliação do conjunto de regras juntamente com o comportamento das regras individuais. Para avaliar os classificadores como sendo uma caixa preta, pode-se utilizar as medidas de acurácia, precisão e F_1, definidas na Seção 2.3 na página 3. Para avaliar as regras individualmente, pode-se utilizar as medidas de avaliação de regras descritas no Apêndice A. Neste trabalho, é proposto o uso das 5 (cinco) medidas seguintes — Equações 12 a 16 — denominadas HQ (*Hypothesis Quality*) para avaliar o desempenho do classificador.

$$HQ_{Acc}(\mathbf{h}) = Acc(\mathbf{h}) \tag{12}$$

$$HQ_{Prec}(\mathbf{h}) = Prec(\mathbf{h}) \tag{13}$$

$$HQ_{F1}(\mathbf{h}) = F1(\mathbf{h}) \tag{14}$$

$$HQ_{A\times C}(\mathbf{h}) = Acc(\mathbf{h}) \times mean(Cov(R_i)), \forall R_i \in \mathbf{h} \tag{15}$$

$$HQ_{P\times C}(\mathbf{h}) = Prec(\mathbf{h}) \times mean(Cov(R_i)), \forall R_i \in \mathbf{h} \tag{16}$$

Deve ser observado que, por se tratar de classificadores simbólicos, podem ser utilizadas diversas maneiras para se descobrir a classificação de um novo exemplo, *i.e.*, dado um exemplo $\mathbf{x}$ e um classificador $\mathbf{h} = \{R_1, ..., R_{N_R}\}$, pode-se utilizar diversos métodos para classificar $\mathbf{x}$. Existem três maneiras para verificar como um conjunto de regras cobre um novo exemplo, como descrito na Seção 2.4 na página 7: ordenada, não-ordenada e ordenada entre classes. Uma vez que não existe uma ordem implícita nas regras que compõem os indivíduos, os métodos para classificar $\mathbf{x}$ propostos neste trabalho consideram os conjuntos de regras como sendo conjuntos de regras não ordenados.

Além disso, neste trabalho são propostos 8 (oito) métodos para classificar $\mathbf{x}$ dado um indivíduo (classificador) denominados SR (*Single Rule classification method*) e SR (*Multiple Rule classification method*). Os quatro primeiros classificam $\mathbf{x}$ considerando a "melhor" regra do classificador, segundo uma medida de qualidade de regras, enquanto que os outros quatro métodos utilizam todo o conjunto de regras para realizar a classificação.

1. SR_{Acc}: É utilizada a melhor regra segundo a medida de avaliação de regra precisão (Acc) — Equação 18 na página 46 — para classificar $\mathbf{x}$;
2. SR_{Lacc}: É utilizada a melhor regra segundo a medida de avaliação de regra precisão de Laplace ($Lacc$) — Equação 36 na página 45 — para classificar $\mathbf{x}$.
3. SR_{Cov}: É utilizada a melhor regra segundo a medida de avaliação de regra cobertura (Cov) — Equação 23 na página 46 — para classificar $\mathbf{x}$.
4. SR_{Ipe}: É utilizado um índice de precisão menos erro denominado Ipe para classificar $\mathbf{x}$, definido pela Equação 17.

$$Ipe = f_{bh} - f_{b\bar{h}} \tag{17}$$

5. MR_{Acc}: São utilizadas todas as regras que cobrem $\mathbf{x}$ para encontrar sua classificação. Para determinar a classe de $\mathbf{x}$, para cada classe $C_v \in \{C_1, ..., C_{N_{Cl}}\}$ é somado o valor da medida de precisão (Acc) — Equação 18 na página 46 — para as regras que classificam o exemplo na classe em questão. A classe com maior valor total é a classificação de $\mathbf{x}$.

6. MR_{Lacc}: São utilizadas todas as regras que cobrem $\mathbf{x}$ para encontrar sua classificação. Para determinar a classe de $\mathbf{x}$, para cada classe $C_v \in \{C_1, ..., C_{N_{Cl}}\}$ é somado o valor da medida de precisão de laplace ($Lacc$) — Equação 36 na página 45 — para as regras que classificam o exemplo na classe em questão. A classe com maior valor total é a classificação de $\mathbf{x}$.

7. MR_{Cov}: São utilizadas todas as regras que cobrem $\mathbf{x}$ para encontrar sua classificação. Para determinar a classe de $\mathbf{x}$, para cada classe $C_v \in \{C_1, ..., C_{N_{Cl}}\}$ é somado o valor da medida de cobertura (Cov) — Equação 23 na página 46 — para as regras que classificam o exemplo na classe em questão. A classe com maior valor total é a classificação de $\mathbf{x}$.

8. MR_{Ipe}: São utilizadas todas as regras que cobrem $\mathbf{x}$ para encontrar sua classificação. Para determinar a classe de $\mathbf{x}$, para cada classe $C_v \in \{C_1, ..., C_{N_{Cl}}\}$ é somado o valor do índice de precisão menos cobertura (Ipe) — Equação 17 — para as regras que classificam o exemplo na classe em questão. A classe com maior valor total é a classificação de $\mathbf{x}$.

4.4 Critério de Parada

Neste trabalho, são utilizados dois critérios de parada:

1. Por um número máximo de gerações que o AG executa. Como saída, o AG oferece o melhor indivíduo dentre todas as gerações executadas;

2. Utilizando um método de convergência. Dado um número de gerações N_{ger}, se a função de avaliação do melhor indivíduo não melhorar nas últimas N_{ger}, algoritmo genético pára de executar.

4.5 Pós-processamento do Indivíduo Resultante

Como saída, o AG oferece um classificador simbólico. Analogamente aos algoritmos de indução de classificadores, no AG aqui proposto é realizado um pós-processamento desse classificador resultante da execução do AG.

Antes de iniciar a execução do AG, é retirado do conjunto de treinamento S'_{tr} oferecido como entrada ao AG, uma amostra com 10% dos exemplos dessa base, denominada S_{val}. Os exemplos restantes em S'_{tr}, que compõem S_{tr}, são utilizados em cada geração do AG, e o conjunto de exemplos $S'_{tr} = S_{tr} \cup S_{val}$ é utilizado no pós-processamento. Nessa fase de pós-processamento, são retiradas do classificador resultante todas as regras que não cobrem nenhum exemplo em S'_{tr}.

4.6 Algoritmo Genético Proposto

Como mencionado anteriormente, o AG proposto neste trabalho segue os passos do AG clássico descrito na Seção 3.1 na página 13. O que varia nesta proposta são os componentes

do AG, ou seja, a maneira que os indivíduos (classificadores) classificam novos exemplos e a função de avaliação para avaliar os cromossomos. Neste trabalho, o AG é testado variando seus componentes, ou seja, utilizando as diferentes funções de avaliação propostas anteriormente e variando os parâmetros do AG.

A seguir, é descrito o ambiente DISCOVER, no qual está inserido o sistema que implementa o AG proposto neste trabalho.

5 O Ambiente DISCOVER

Na maioria dos projetos desenvolvidos pelos pesquisadores do Laboratório de Inteligência Computacional — LABIC — relacionados com aquisição automática de conhecimento e avaliação de conhecimento, inclusive no projeto no qual está inserido este trabalho, diversas tarefas como transformação de dados e formatos, execução de algoritmos de aprendizado, medições, entre outras, devem ser executadas diversas vezes. Muitas dessas funcionalidades podem ser realizadas de maneira semi-automática por sistemas comerciais. Porém, além da aquisição desses sistemas demandar um alto custo, muitas vezes proibitivo para as universidades públicas, esses sistemas fazem uso de algoritmos e ferramentas proprietários, o que dificulta sua utilização por pesquisadores que pretendem analisar e desenvolver novos algoritmos e ferramentas (Batista, 2003).

Assim, muitas vezes, alguns algoritmos e ferramentas são re-implementados por pesquisadores do próprio LABIC, ou por outros pesquisadores, que disponibilizam ferramentas de domínio público, tais como $\mathcal{MLC}$++ (Kohavi et al., 1994), WEKA (Witten and Frank, 2005) e YALE (Fischer et al., 2002). Entretanto, eventuais problemas decorrentes da utilização de algoritmos re-implementados, tais como um funcionamento diferente do proposto originalmente, fazem com que os algoritmos implementados pelos próprios idealizadores sejam preferidos para a realização de experimentos para avaliar diferentes algoritmos. Adicionalmente, muitas ferramentas são implementadas para automatizar parcial ou integralmente algumas tarefas. Desse modo, não é incomum a ocorrência de sobreposições de implementações pela falta de comunicação e documentação do que já foi implementado pelos pesquisadores do laboratório. Ainda, também não é incomum a perda de implementações quando seus autores se desligam do laboratório ao final de seu projeto de pós-graduação.

Para tentar resolver tais problemas, foi inicialmente proposto por Baranauskas and Batista (2000) um projeto denominado Projeto DISCOVER, que tem como principal objetivo integrar e padronizar os diversos projetos e ferramentas desenvolvidos pelos pesquisadores do LABIC relacionados com aquisição automática de conhecimento e avaliação de conhecimento. No início, o projeto DISCOVER consistiria apenas de um repositório de *scripts* para facilitar a configuração e execução de experimentos, mas surgiu a proposta de criação de um ambiente integrado no qual os *scripts* seriam substituídos por bibliotecas de classes desenvolvidas em PERL (PERL, 1999). Deve ser ressaltado que todo o processo de implementação do projeto foi estudado segundo os fundamentos da Engenharia de Software (Rozante, 2003). Além disso, quanto a arquitetura do ambiente DISCOVER, foi proposto em Prati (2003) um *framework* para a integração dos componentes do ambiente DISCOVER utilizando *software patterns*, no qual os componentes são integrados por meio de uma linguagem denominada xDML, baseada em XML. A implementação da interface gráfica para essa integração será realizada futuramente.

O ambiente DISCOVER oferece vantagens em relação a outros sistemas com objetivos seme-

lhantes, pois permite a visão unificada que os formatos baseados em padrões proporcionam ao pesquisador (desenvolvedor) de novos componentes. Os padrões de representação foram sendo definidos por área, sendo que em (Prati et al., 2001a) é proposta uma sintaxe padrão para representação de conhecimento de diversos indutores simbólicos denominada $\mathcal{PBM}$ (Prati et al., 2002, 2001b). Para a representação de dados foi proposta uma sintaxe padrão (Batista, 2001), denominada DSX — Discover *Dataset Syntax*, a qual permite a utilização da biblioteca de classes DOL (Batista and Monard, 2006), para converter os arquivos de dados para a sintaxe utilizada por diversos sistemas de aprendizado simbólico, tais como $\mathcal{C}4.5$, $\mathcal{C}4.5rules$ e $\mathcal{CN}2$ entre outros. Mais especificamente, o ambiente Discover Learning Environment — DLE é composto pela biblioteca de classes Discover Object Library — DOL e pelo ambiente para gerenciamento de experimentos Sniffer, que foram desenvolvidos para diferentes tarefas de pré-processamento em mineração de dados (Batista, 2003; Batista and Monard, 2003, 2006). Novas funcionalidades estão sendo especificadas, principalmente, para regras de regressão (Dosualdo, 2003; Pugliesi, 2004) e regras de associação (Melanda, 2004). Além disso, outros trabalhos foram ou estão sendo desenvolvidos na área de aprendizado de máquina relacional e proposicional (Lee, 2005; Ferro, 2006, 2004; Milaré, 2003; Gomes, 2002; Bernardini, 2002), aprendizado de máquina semi-supervisionado (Sanches, 2003; Matsubara, 2004), mineração de dados (Paula, 2003; Pila, 2003; Chiara, 2003; Baranauskas, 2001) e mineração de textos (Martins, 2003; Matsubara, 2004; Matsubara et al., 2003).

Para desenvolver o sistema para executar e avaliar o AG proposto neste trabalho, foram utilizados:

1. O ambiente computacional DLE, que possui funcionalidades que facilitam a implementação de sistemas a serem integrados no Discover;
2. O ambiente computacional ELE, que possui facilidades para avaliação de métodos de AM que utilizam vários classificadores como dados de entrada;
3. O ambiente computacional NNRules, sobre o qual foram utilizadas técnicas de reuso de software para construir o ambiente computacional GAERE.

Assim, antes de ser detalhado o desenvolvimento do sistema para executar o AG proposto, são descritos a seguir os objetivos e as funcionalidades dos ambientes computacionais Discover Learning Environment - DLE (Batista and Monard, 2003, 2006), NNRules (Milaré, 2003) e ELE (Bernardini and Monard, 2004).

5.1 O Ambiente Computacional DLE

O ambiente computacional DLE tem como principal objetivo prover um *framework* para que novas técnicas de pré-processamento de dados possam ser facilmente e rapidamente implementadas e avaliadas experimentalmente. O ambiente computacional DLE é composto por dois módulos distintos: a biblioteca de classes Discover Object Library - DOL, cujo objetivo principal é ser uma biblioteca de métodos de pré-processamento de dados, e o ambiente para gerenciamento de experimentos Sniffer, cujo objetivo principal é automatizar avaliações e comparações experimentais de algoritmos de aprendizado. Para utilizar a biblioteca de classes DOL, foi proposta uma sintaxe padrão para representação de dados, denominada DSX – *Discover Dataset Syntax* (Batista, 2001). A seguir, são descritas as principais funcionalidades da biblioteca de classes DOL e do ambiente computacional Sniffer. As descrições aqui apresentadas estão baseadas

em (Batista and Monard, 2003). Maiores detalhes do ambiente computacional DLE, tais como detalhes de arquitetura, projeto e implementação do ambiente, os quais fogem do escopo deste trabalho, podem ser encontrados em (Batista and Monard, 2003; Batista, 2003).

5.2 A Biblioteca de Classes DOL

Entre as principais funcionalidades da biblioteca DOL, pode-se citar:

Manipulação de atributos e dados A biblioteca DOL provê métodos que fornecem informações do conjunto de dados[11] que está sendo pré-processado. Utilizando a biblioteca, os dados podem ser manipulados, ou seja, lidos e modificados. Entre as principais tarefas de manipulação de atributos e dados pode-se citar: alteração de nomes de atributos, conversão de tipos de atributos, modificação de posição de atributos na tabela de dados, modificação do atributo classe, remoção temporária ou efetiva de atributos e exemplos, adição de exemplos ao conjunto de dados, dentre outras.

Integração com diversos sistemas de AM A biblioteca DOL pode ser integrada com os principais sistemas de aprendizado acadêmicos disponibilizados pelos autores. Para que essa funcionalidade fosse possível, foram criados conversores de dados que convertem os dados carregados da sintaxe da biblioteca DOL para as sintaxes requeridas pelos seguintes sistemas de aprendizado simbólico: $\mathcal{C}4.5$ (Quinlan, 1988), $\mathcal{C}4.5rules$ (Quinlan, 1987), $\mathcal{ID}3$ (Quinlan, 1986), $\mathcal{N}ew\mathcal{ID}$ (Boswell, 1990), $\mathcal{CN}2$ (Clark and Boswell, 1991), dentre outros.

Filtro de Exemplos e Atributos A biblioteca DOL oferece um sistema de filtros que permite ocultar temporariamente parte dos exemplos ou dos atributos de um conjunto de dados. Os filtros podem ser criados a partir de expressões lógicas envolvendo atributos ou utilizando uma lista que indica explicitamente quais exemplos devem ser filtrados. A biblioteca permite que vários filtros sejam definidos. Além disso, é possível definir filtros em função de outros filtros previamente definidos. O sistema de filtragem de exemplos é, por exemplo, a base para a criação de um conjunto de classes PERL que criam amostras aleatórias e estratificadas de um conjunto de exemplos.

Estatísticas Descritivas e Correlações Estatísticas descritivas podem ser utilizadas tanto pelo usuário quanto por métodos de pré-processamento de dados. Ainda, informações sobre correlações entre atributos, como correlação linear e covariância, são a base para a construção de diversos métodos de pré-processamento de dados.

Métodos de *Resampling* Como mencionado, os métodos de *resampling* são métodos estatísticos capazes de estimar apropriadamente a taxa de erro verdadeira, mesmo em um conjunto reduzido de dados. Os métodos de *resampling* utilizam conjuntos de teste para estimar a taxa de erro verdadeira, os quais contém exemplos que não foram utilizados no treinamento do sistema de aprendizado. Diferentes métodos de *resampling* particionam de diferentes formas o conjunto de dados original em conjuntos de treinamento e teste. Atualmente, a biblioteca de classes DOL provê dois métodos de *resampling*: o *k-fold cross-validation* e o *k-fold cross validation estratificado*, descritos na Seção 2.3 na página 3.

[11] Atualmente são considerados dados no formato atributo-valor — Tabela 1 na página 2.

5.3 O Ambiente Computacional SNIFFER

Conforme dito anteriormente, o ambiente SNIFFER (Batista and Monard, 2003) é um ambiente computacional para gerenciamento de avaliações e comparações experimentais de algoritmos de aprendizado. Tal ambiente está integrado com diversos sistemas de aprendizado, citados anteriormente. Freqüentemente, para realizar avaliações e comparações de algoritmos de aprendizado, é necessário executar um experimento diversas vezes. Primeiro, porque a repetição dos experimentos utilizando os métodos de *resampling* fornece ao usuário uma estimativa da variância das médias dos resultados; segundo, porque na maioria das vezes é necessário ajustar alguns parâmetros para cada conjunto de dados; e finalmente, porque as análises experimentais freqüentemente devem ser executadas em diversos conjuntos de dados, utilizando diferentes algoritmos de aprendizado, para uma melhor compreensão dos resultados. Dessa maneira, é bastante comum que o usuário seja obrigado a executar um determinado experimento centenas de vezes.

Diante de tantas execuções, torna-se muito difícil gerenciar os experimentos manualmente, sendo necessário automatizar essa tarefa. O ambiente computacional SNIFFER foi desenvolvido com esse fim. Tal ambiente possui as seguintes principais funcionalidades:

Gerenciamento de sintaxes dos sistemas de aprendizado Em experimentos que envolvem execuções de diferentes sistemas de aprendizado, é comum que os sistemas utilizem sintaxes distintas para os arquivos de declaração de dados e de atributos. O ambiente SNIFFER converte, quando necessário, o conjunto de dados na sintaxe do DISCOVER para a sintaxe do sistema de aprendizado a ser executado.

Aplicação de métodos de *resampling* para estimar a taxa de erro verdadeira Os conjuntos de dados devem ser divididos em conjuntos de treinamento e teste para que a taxa de erro verdadeira seja estimada no conjunto de teste. O ambiente SNIFFER realiza essa tarefa, desde que solicitado pelo usuário. O usuário pode também escolher qual método de *resampling* deve ser aplicado ao conjunto de dados, bem como definir explicitamente os conjuntos de treinamento e teste.

Recuperação das taxas de erro Os principais sistemas de aprendizado permitem que sejam fornecidos conjuntos de treinamento e teste para a criação do modelo e a estimativa da taxa de erro verdadeira, respectivamente. Normalmente, os sistemas de aprendizado gravam a matriz de confusão em um arquivo texto com sintaxe proprietária. O ambiente SNIFFER recupera essa matriz de confusão e a armazena para posterior análise dos resultados.

Cálculo de medidas de desempenho Recuperadas as matrizes de confusão para cada iteração do método de *resampling*, é necessário calcular estatísticas que fornecem um indicativo de desempenho do sistema de aprendizado no conjunto de dados analisado. O ambiente SNIFFER calcula medidas de desempenho geral levando em consideração todos os valores do atributo classe.

Comparação de medidas de desempenho Uma vez que as medidas de desempenho foram calculadas é necessário compará-las para verificar se existem diferenças significativas entre elas, ou seja, se um método superou outro com uma diferença significativa de 95% ou 99% de confiabilidade. O ambiente SNIFFER utiliza o teste t para realizar essa tarefa.

Um outro objetivo importante do SNIFFER é automatizar, sempre que possível, a publicação dos resultados. Assim, o ambiente fornece ao usuário uma segurança maior de que os resultados

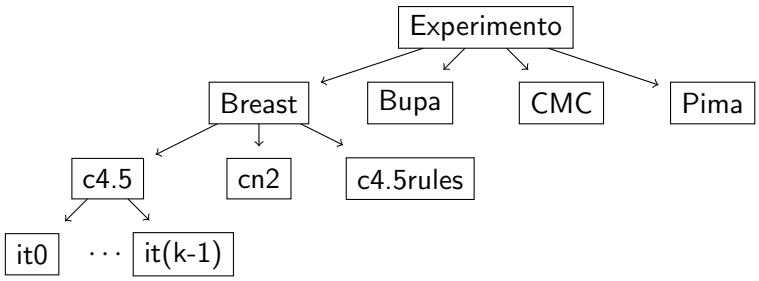

Figura 7: Exemplo de experimento organizado em diretórios para o ambiente SNIFFER (Batista and Monard, 2003)

publicados são fiéis aos resultados obtidos nos experimentos. Como freqüentemente os resultados dos experimentos envolvem uma grande quantidade de valores numéricos, é muito comum a introdução de erros durante o processo manual de confecção de tabelas e gráficos. O ambiente SNIFFER fornece ao usuário relatórios resumidos e detalhados dos resultados obtidos, além de tabular os resultados em um formato que pode ser utilizado para gerar gráficos no Gnuplot[12] e tabelas no processador de textos LaTeX[13].

Para organizar um experimento, o ambiente SNIFFER utiliza a estrutura do sistema de arquivos do sistema operacional no qual está sendo executado. Os conjuntos de dados a serem analisados devem ser dispostos em diretórios, sendo que alguns identificadores de diretórios possuem significado especial para o ambiente SNIFFER. Para tornar mais simples a descrição do funcionamento do ambiente SNIFFER, o mesmo é ilustrado utilizando o exemplo a seguir.

Considere que desejamos comparar o desempenho de três sistemas de aprendizado: $\mathcal{C}4.5$, $\mathcal{C}4.5rules$ e $\mathcal{CN}2$ sobre quatro conjuntos de dados da UCI (Blake et al., 1998): **Breast**, **Bupa**, **CMC** e **Pima**. Inicialmente, o usuário deve organizar os conjuntos de dados em diretórios. A forma que os diretórios devem ser organizados depende de como o usuário deseja que os testes de hipótese sejam feitos. Por hora, vamos pensar que os conjuntos de dados foram organizados pelo usuário da forma em que são mostrados na Figura 7. Existe um diretório raiz para o experimento clamado Experimento. Dentro desse diretório, o qual engloba todo os experimento, foram criados quatro diretórios: Breast, Bupa, CMC e Pima, ou seja, um para cada conjunto de dados que será analisado. Dentro de cada diretório dos conjuntos de dados, foram criados outros três diretórios: c4.5, cn2 e c4.5rules (um para cada sistema de aprendizado a ser utilizado). Na Figura 7 são mostrados somente os diretórios contidos no diretório Breast, por motivo de simplificação os demais foram omitidos. Por fim, opcionalmente, o usuário pode criar diretórios enumerados que devem seguir o padrão it0, it1, ... , it(k-1), ou seja, um para cada iteração do método de *resampling* por ele considerado, sendo k o número total de iterações do método de *resampling* utilizado. Nesse caso, o usuário necessita colocar embaixo de cada um desses diretórios o conjunto de teste respectivo. Caso o usuário especificar um dos métodos de *resampling* definidos atualmente no SNIFFER, *i.e k-fold cross-validation* ou *k-fold cross-validation* estratificado, esses diretórios `it0`,`it1`,...,`it(k-1)` são automaticamente criados pelo SNIFFER bem como os conjuuntos de treinamento e teste correspondentes dentro de cada um desses diretórios.

Ao ambiente SNIFFER deve ser fornecido um *ponto de entrada*, ou seja, um diretório que

[12] http://www.gnuplot.info
[13] http://www.latex-project.org

seja raiz para todo o experimento. A partir desse diretório, o ambiente vasculha a árvore de diretório contida no ponto de entrada a procura de diretórios com identificadores especiais. Os identificadores especiais identificam um sistema de aprendizado específico. Atualmente, os identificadores especiais para algoritmos de aprendizado simbólico que o ambiente reconhece estão listados na Tabela 5. Entretanto, o SNIFFER está preparado para acomodar outros sistemas de aprendizado.

Identificador	Sistema de Aprendizado
`c4.5`	$\mathcal{C}4.5$ (Quinlan, 1988)
`c4.5rules`	$\mathcal{C}4.5rules$ (Quinlan, 1987)
`id3`	$\mathcal{ID}3$ (Quinlan, 1986)
`cn2`	$\mathcal{CN}2$ (Clark and Boswell, 1991)
`newid`	$\mathcal{N}ew\mathcal{ID}$ (Boswell, 1990)

Tabela 5: Identificadores especiais para diretórios reconhecidos atualmente pelo SNIFFER para algoritmos de aprendizado simbólico

Quando um diretório com identificador especial é encontrado, o ambiente SNIFFER realiza uma das seguintes tarefas:

1. Caso for solicitado pelo usuário, o ambiente executa um método de *resampling* dividindo os dados em diversos conjuntos de treinamento e teste. Nesse caso, os arquivos com extensão `.data` e `.names` correspondentes a todo o conjunto de dados precisam estar presentes no diretório com o identificador especial. Como resultado, são criados k diretórios com os identificadores `it0`, ... , `it(k-1)`, um para cada iteração do método de *resampling*, sendo k o número total de iterações do método. Dentro de cada um desses diretórios o ambiente grava três arquivos com as extensões `.data`, `.test` e `.names`, os quais contém, respectivamente, o conjunto de dados de treinamento e teste dessa iteração, e o arquivo de declaração de atributos desses dados;

2. Se o usuário não desejar que o ambiente aplique um método de *resampling*, o usuário pode criar os diretórios com os conjuntos de treinamento e teste, que ele deve fornecer, os quais devem ser nomeados `it0`, ... ,`it(k-1)`. Isso é útil em algumas situações quando o usuário deseja avaliar dois ou mais sistemas de aprendizado nos mesmos conjuntos de treinamento e teste. Nesse caso, o usuário solicita que o ambiente não execute um método de *resampling*, e o ambiente utiliza os arquivos de dados com as extensões `.data`, `.test` e `.names` que foram fornecidos e armazenados pelo usuário dentro de cada diretório com o identificador `it` correspondente.

A partir daí, o ambiente passa a executar sobre esses dados o sistema de aprendizado identificado pelo nome do diretório atual — Tabela 5. Para isso, o ambiente SNIFFER acessa cada diretório `it` e realiza as seguintes tarefas:

1. Caso for solicitado pelo usuário, converte os arquivos de dados, ou seja, converte os exemplos presentes nos arquivos `.data`, `.names` e `.test`, que estão na sintaxe padrão DSX, para a sintaxe do sistema de aprendizado a ser executado;

2. Executa o sistema de aprendizado selecionado sobre o conjunto de treinamento e estima o erro de classificação sobre o conjunto de teste correspondente;

3. Por fim, extrai a matriz de confusão dos arquivos de saída do sistema de aprendizado, a qual contém o número de classificações incorretas feitas pelo classificador induzido sobre o conjunto de teste. As matrizes de confusão são armazenadas pelo ambiente para a realização de cálculos de desempenho.

O ambiente SNIFFER percorre toda a árvore de diretórios contida no ponto de entrada fornecido pelo usuário. Ao final, o ambiente pode tanto gerar relatórios descrevendo os resultados obtidos (por exemplo, média e desvio padrão dos erros cometidos pelo indutor nas k iterações de *resampling*, bem como a comparação entre os resultados obtidos para cada um dos conjuntos de dados utilizados) quanto ser acessado através de sua API para que o usuário recupere determinados dados ou estatísticas de desempenho.

5.4 O Sistema Computacional NNRULES

O principal objetivo do desenvolvimento do ambiente NNRULES (Milaré, 2003) foi construir um ambiente que permite avaliar experimentalmente o uso de sistemas de aprendizado simbólico e AGs para a extração de conhecimento simbólico de Redes Neurais Artificiais (RNAs). O projeto do ambiente NNRULES foi criado tendo-se em vista o desenvolvimento de um ambiente flexível que pudesse ser facilmente modificado para avaliar e testar novas idéias. Por esse motivo, decidiu-se dividir o ambiente em módulos, os quais podem ser compostos da forma mais adequada para a realização de um determinado experimento.

Os módulos do NNRULES, os quais são implementados por meio de classes, podem ser vistos como bibliotecas. Foi implementado, também, um conjunto de scripts, os quais utilizam os métodos implementados pelos módulos para automatizar uma determinada tarefa. Os scripts utilizados nos experimentos realizam tarefas que vão desde o pré-processamento dos dados até a execução dos métodos de extração de conhecimento de RNAs.

O ambiente NNRULES faz parte do projeto DISCOVER. Assim como os outros módulos do DISCOVER, o ambiente NNRULES foi também implementado em Perl (PERL, 1999) e orientado a objeto. O ambiente NNRULES está dividido em três módulos principais: RULEBASE, RULESET e GA, descritos a seguir.

O Módulo RuleBase: O módulo RULEBASE armazena e gerencia uma base de regras. As regras armazenadas em uma mesma instância do módulo RULEBASE devem pertencer a um domínio comum, isto é, devem fazer referência a um conjunto comum de atributos. As regras são armazenadas em uma instância do módulo RULEBASE na sintaxe padrão de regras $\mathcal{PBM}$ (Prati et al., 2001b). Também, para as regras armazenadas, o módulo calcula a matriz de contingência das regras — Tabela 4 na página 8 — para posterior avaliação das regras armazenadas. Para converter as regras da sintaxe do algoritmo de indução para a sintaxe $\mathcal{PBM}$ bem como para calcular as matrizes de contingência, o módulo RULEBASE utiliza uma classe chamada RuleStream.

O módulo RULEBASE provê diversas funcionalidades para a manipulação de regras. Entre as principais estão prover informações sobre a matriz de contingência das regras e sobre o sistema de aprendizado que induziu essas regras; informar quais são as regras *default* e remover essas regras da base; prover informações sobre a complexidade sintática das regras; verificar se uma regra cobre um determinado exemplo, entre outras.

O Módulo RuleSet: O módulo RuleSet permite implementar classificadores com subconjuntos de regras armazenadas em uma instância do módulo RuleBase. Uma instância do módulo RuleSet não armazena as regras diretamente, mas somente uma lista com os índices das regras que pertencem ao classificador. Como mencionado anteriormente, essas regras estão armazenadas em uma instância do módulo RuleBase.

Pode-se criar diversas instâncias do módulo RuleSet para uma mesma base de regras armazenada em uma instância do módulo RuleBase. Cada classificador pode ser composto por diferentes regras, bem como por diferentes quantidades de regras. Uma das principais características do módulo RuleSet é que ele permite classificar um exemplo segundo um conjunto de regras nele armazenado, utilizando as abordagens descritas na Seção 2.4 na página 7 — ordenada, não-ordenada e ordenada entre as classes.

O Módulo GA: O módulo GA implementa um Algoritmo Genético para realizar uma busca por um conjunto de regras que possua grande fidelidade com uma RNA. O ambiente NNRules utiliza um AG na busca por um conjunto de regras que classifique um conjunto de exemplos da mesma forma que uma RNA treinada classifica esses exemplos. Dessa forma, o conjunto de regras pode ser utilizado para explicar essa RNA. Sendo assim, o AG tem como principal objetivo encontrar um conjunto de regras que maximize a taxa de fidelidade entre o conjunto de regras e a RNA, tal que a explicação correspondente à classificação de novos exemplos pela RNA é realizada utilizando esse conjunto de regras.

No AG implementado no ambiente NNRules, cada indivíduo é um conjunto de regras gerenciado por uma instância do módulo RuleSet. As operações de *crossover* e mutação foram implementadas no módulo AG por meio dos métodos de adição e remoção de regras provida pela interface do módulo RuleSet.

Os classificadores simbólicos induzidos pelos sistemas de aprendizado são convertidos para regras na sintaxe $\mathcal{PBM}$ e carregados em uma instância do módulo RuleBase. Uma vez que as regras estão armazenadas na base de regras, o AG inicia a busca por um conjunto de regras com alta fidelidade com a RNA que gerou os novos valores do atributo classe.

Nos experimentos descritos em (Milaré, 2003), a taxa de fidelidade é utilizada como função de aptidão. A taxa de fidelidade é o percentual de exemplos de um conjunto de dados classificados de forma idêntica pelo conjunto de regras, isto é, um indivíduo, e a RNA que se deseja explicar.

5.5 O Sistema Computacional ELE

Assim como o NNRules e outros projetos atrelados ao ambiente Discover, o sistema computacional ELE foi criado tendo-se em vista o desenvolvimento de um ambiente flexível, que pudesse ser facilmente modificado para avaliar e testar novas idéias relacionadas à construção de *ensembles*[14] simbólicos e combinação de classificadores simbólicos (Bernardini and Monard, 2004). Decidiu-se, então, dividir o sistema ELE em 2 (dois) módulos principais, os quais podem ser compostos da forma mais adequada para a realização de um determinado experimento: o módulo EnsembleTree e o módulo EnsembleMethods, descritos a seguir.

O Módulo EnsembleTree: Para construir *ensembles* simbólicos como os propostos em (Bernardini and Monard, 2004), é necessário construir os classificadores que compõem o *en-*

[14]Um *ensemble* de classificadores é um conjunto de classificadores cujas decisões individuais são combinadas de alguma forma para classificar um novo caso.

semble, estimar o erro desses classificadores componentes, tendo em vista que alguns dos métodos de combinação propostos utilizam como parâmetro a taxa de erro dos classificadores, e realizar a combinação das decisões dos classificadores para classificar novos exemplos dados ao *ensemble*. Como um *ensemble* de classificador (simbólico ou não) pode ser visto como um classificador, os métodos de construção de *ensembles* propostos no citado trabalho necessitam ser avaliados empiricamente.

Dessa maneira, o módulo EnsembleTree foi desenvolvido para construir uma árvore de diretórios que permite:

1. construir (induzir) os classificadores que compõem o *ensemble* bem como estimar suas respectivas taxas de erro;
2. compor o *ensemble*, formado pelas hipóteses (classificadores) induzidas e suas taxas de erro, bem como um identificador que indique o método de classificação dos exemplos[15] e o método de combinação das decisões de classificação;
3. estimar a taxa de erro do *ensemble* construído.

Para tais tarefas, é construída uma árvore de diretórios pelo EnsembleTree para que sejam utilizadas as facilidades oferecidas pelo Sniffer. Uma descrição mais detalhada da árvore de diretórios construída por EnsembleTree pode ser encontrada em (Bernardini and Monard, 2004).

O Módulo EnsembleMethods: Este módulo consiste de uma biblioteca orientada a objeto que tem como objetivo dar suporte ao teste de diferentes métodos de construção de *ensembles*, compostos por métodos de classificação de exemplos e de combinação de classificadores. As principais funcionalidades da biblioteca EnsembleMethods são:

Métodos de classificação de exemplos: Além de classificar exemplos utilizando o classificador como um todo, pode-se considerar em classificar um novo exemplo utilizando a melhor regra dada uma determinada heurística (medida de regra) para selecionar essa regra. A biblioteca EnsembleMethods provê diversos métodos de classificação de exemplos utilizando ambas abordagens, os quais estão descritos em (Bernardini and Monard, 2004).

Métodos de combinação de classificadores: Pode-se utilizar diversos métodos de combinação de classificadores. A biblioteca EnsembleMethods provê os métodos de combinação de classificadores descritos em (Bernardini and Monard, 2004).

Como mencionado, para implementar o AG proposto na Seção 4 na página 20, foram utilizadas técnicas de reuso de software do sistema NNRules, proposto em (Milaré, 2003), com o objetivo de construir o sistema computacional GAERE, bem como foram utilizadas facilidades oferecidas pelo sistema computacional ELE (Bernardini and Monard, 2004). A seguir, é descrito o ambiente computacional GAERE proposto neste trabalho.

[15] Como os classificadores que compõem o *ensemble* são simbólicos, além de se utilizar cada classificador como caixa preta, pode ser utilizada a melhor regra de cada classificador segundo algum critério de avaliação de regra para classificar um novo caso.

6 Descrição do Sistema Computacional para Evoluir Classificadores Simbólicos — GAERE

Quando são considerados classificadores induzidos por algoritmos de aprendizado de máquina, um dos aspectos mais importantes é a avaliação experimental desses classificadores, pois não existem instrumentos formais para decidir qual algoritmo de aprendizado é ótimo, *i.e* induzirá o melhor classificador, para um dado problema (Dietterich, 1989; Kibler and Langley, 1988; Schaffer, 1994). Assim, é necessário realizar a avaliação experimental do AG proposto neste trabalho, além de fornecer ao usuário uma maneira de evoluir classificadores simbólicos utilizando o referido AG.

Para avaliar o AG, neste trabalho é proposta uma metodologia análoga à metodologia de avaliação do ambiente ELE, descrito sucintamente na Seção 5.5. Tal metodologia foi implementada em um módulo, denominado GATree. Tanto a referida metodologia quanto o módulo GATree são descritos a seguir. Além do módulo GATree, foram construídos 3 (três) outros módulos para compor o sistema GAERE: o módulo GA, no qual está implementado o AG proposto neste trabalho, o módulo RuleStruct, responsável por carregar classificadores simbólicos no formato padrão $\mathcal{PBM}$ em uma única base de regras, e o módulo RuleSetGA, responsável por construir os indivíduos do AG e manipulá-los, bem como classificar novos exemplos dado um indivíduo (classificador) segundo os métodos de classificação propostos na Seção 4 na página 20. A seguir, são descritos cada um desses módulos.

6.1 Avaliação Experimental do AG proposto

Para evoluir classificadores simbólicos utilizando o AG proposto neste trabalho, inicialmente é necessário construir a população inicial composta de classificadores simbólicos. Para construir a população inicial, podem ser utilizadas tanto regras criadas pelo usuário do AG quanto podem ser utilizadas regras induzidas por algoritmos de aprendizado simbólico. Para facilitar a construção do sistema GAERE e facilitar a execução do AG, foram utilizados os ambientes Sniffer e DLE, descritos na Seção 5, foi estendido o ambiente Sniffer, extensão esta descrita no Apêndice B. Para facilitar a construção da população inicial, foi decidido utilizar regras construídas pelos algoritmos de AM simbólicos $\mathcal{CN}2$ (Clark and Niblett, 1989), $\mathcal{C}4.5$ e $\mathcal{C}4.5rules$ (Quinlan, 1988) e pelo algoritmo de geração de regras de associação $\mathcal{A}$priori (Borgelt and Kruse, 2002), conforme mencionado anteriormente. Dado um conjunto de exemplos S, são induzidos classificadores utilizando os indutores indicados pelo usuário dentre os previamente mencionados (podem ser utilizados todos eles, inclusive). Para utilizar o AG, é necessário utilizar um conjunto de treinamento para avaliar os classificadores construídos a cada geração, um conjunto de validação para realizar pós-processamento no classificador resultante e um conjunto de teste utilizado para testar o classificador final.

Para avaliar o AG proposto, pode-se utilizar diversas técnicas. Neste trabalho, a metodologia proposta para avaliar o AG é baseada na técnica de *k-fold cross-validation* estratificado, descrita na Seção 2 na página 2. Na Figura 8 é ilustrada essa metodologia. Vamos explicar essa figura observando-a da esquerda para a direita. Inicialmente, o conjunto de dados S é dividido em k partições $S'_0, ..., S'_{(k-1)}$. Na **primeira iteração** do *k-fold cross-validation*, $S'_{tr_0} = S'_1 \cup ... \cup S'_{(k-1)}$. S'_{tr_0} é então utilizado para construir regras utilizando os algoritmos $\mathcal{C}4.5$ e/ou $\mathcal{C}4.5rules$ e/ou $\mathcal{CN}2$ e/ou $\mathcal{A}$priori. As regras induzidas/geradas formam a base de regras BR_0. Do mesmo conjunto S'_{tr_0} é extraída uma amostra com 10% do número de exemplos de S'_{tr_0}. Tal amostra

forma o conjunto S_{val_0} e a diferença entre os conjuntos S'_{tr_0} e S_{val_0} forma o conjunto S_{tr_0}; *i.e.*, $S_{tr_0} = S'_{tr_0} - S_{val_0}$. O AG é então executado, sendo que a base de regras BR_0 é utilizada para construir a população inicial e fornecer regras para os operadores de *crossover* e mutação; o conjunto de dados S_{tr_0} é utilizado para avaliar os indivíduos (classificadores) a cada geração do AG, já que a tarefa de avaliar um classificador depende de um conjunto de dados; o conjunto de dados $S'_{tr_0} = S_{tr_0} \cup S_{val_0}$ é utilizado para realizar pós-processamento no classificador resultante das iterações do algoritmo genético, *i.e.*, são retiradas regras que não cobrem nenhum dos exemplos pertencentes a S'_{tr_0}; e, finalmente, a partição restante S'_0 é utilizada como conjunto de teste S_{te_0} para avaliar o classificador gerado após a tarefa de pós-processamento — dessa avaliação, é extraída a taxa de erro ε^*_1. Na **segunda iteração** do *k-fold cross-validation*, $S'_{tr_1} = S'_0 \cup S'_2 \cup ... \cup S'_{(k-1)}$. Analogamente, S'_{tr_1} é então utilizado para construir regras que formam a base de regras BR_1 e do mesmo conjunto S'_{tr_1} é extraída uma amostra com 10% dos exemplos, formando o conjunto S_{val_1} e $S_{tr_1} = S'_{tr_1} - S_{val_1}$. Da mesma maneira, o AG é então executado, sendo usados BR_1, S_{tr_1}, S_{val_1} e S_{te_1} e sendo encontrada a taxa de erro ε^*_1. O processo se repete até que sejam realizadas k iterações, tendo como resultado k taxas de erro . De $\varepsilon^*_0, ..., \varepsilon^*_{(k-1)}$ extrai-se a estimativa da taxa de erro do AG ε^* segundo os parâmetros utilizados.

Para executar essa metodologia para avaliação do AG proposto, no sistema GAERE foi feita uma extensão do ambiente SNIFFER (Apêndice B) de maneira a reconhecer identificadores do AG proposto. Para utilizar o ambiente SNIFFER, é necessário construir uma árvore de diretórios para que o SNIFFER possa ser chamado. Antes de descrever a construção e a estrutura de tal árvore de diretórios, é necessário definir que, quando queremos indicar um caminho na árvore de diretórios de um diretório <dir_pai> para um diretório <dir_filho>, sendo que <dir_filho> está dentro do diretório <dir_pai>, representamos como sendo <dir_pai>→<dir_filho>. Na Figura 9 é ilustrada a árvore de diretórios utilizada para construir as regras que formam as bases de regras nas k iterações do *k-fold cross-validation*. Dado um conjunto de exemplos S no formato padrão DSX, cujo arquivo está localizado no diretório Experimento, inicialmente são construídos k diretórios it0,...,it(k-1). Dentro de cada diretório it0,...,it(k-1) são colocados os arquivos de dados representando o conjunto de treinamento, o conjunto de validação e o conjunto de teste da respectiva iteração. Para construir esses conjuntos, o conjunto de dados inicial S é dividido em k partições $S_0, ..., S_{(k-1)}$. Em it_0 o conjunto S'_0 é colocado como sendo o conjunto de teste S_{te_0}; do conjunto $S_{tr_0} = S'_1 \cup ... \cup S'_{(k-1)}$ é extraída uma amostra com 10% dos exemplos, sendo essa amostra colocada em it_0 como conjunto de validação S_{val_0} e o restante dos exemplos em S'_{tr_0} é colocado em it_0 como o conjunto de treinamento S_{tr_0}; em it_1 o conjunto S'_1 é colocado como sendo o conjunto de teste S_{te_1}, do conjunto $S'_{tr_1} = S'_0 \cup S'_2 \cup ... \cup S_{(k-1)}$ é extraída uma amostra com 10% dos exemplos, sendo essa amostra colocada em it_1 como conjunto de validação S_{val_1} e o restante dos exemplos em S_{U_1} é colocado em it_1 como o conjunto de treinamento S_{tr_1}; e assim sucessivamente. Ainda, dentro de cada diretório it0,...,it(k-1) é construído um diretório denominado ClassifiersInduction. Dentro de cada diretório ClassifiersInduction é construído um diretório S_l, com $l = 1, ..., L$, onde L é o número de algoritmos de aprendizado de máquina selecionados pelo usuário. Dentro de cada diretório S_l, é construído um diretório cujo nome é dado pelo identificador do algoritmo de aprendizado[16]. Dentro de cada um desses diretórios, é construído um diretório denominado it0.

Dadas as regras construídas em arquivos no formato $\mathcal{PBM}$ presentes na árvore de diretórios ilustrada na Figura 9, é construída outra árvore de diretórios, ilustrada na Figura 10 para o AG ser executado. Nessa figura, <GAId> pode assumir os identificadores listados na Tabela 6 na página seguinte. Deve ser observado que os identificadores são diferentes conforme altera o

[16]Os identificadores reconhecidos pelo SNIFFER após realizada a extensão podem ser encontrados no Apêndice B

método de classificação de cada exemplo e a função de avaliação. Nos diretórios it_0,...,it_(k-1) da Figura 10, devem estar presentes os arquivos i correspondentes, $i = 0, ..., (k-1)$ contendo as regras construídas, pertencentes ao conjunto BR_i, e os conjuntos de treinamento S_{tr_i}, validação S_{val_i} e teste S_{te_i}. Se na raíz da árvore ilustrada na Figura 9, indicada pelo nó Experimento, for encontrado um arquivo nomeado GA.conf, este arquivo é também copiado para cada um dos diretórios it_0,...,it_(k-1). Esse arquivo contém valores de parâmetros para execução do AG. Caso não seja encontrado este arquivo, o AG é executado com seus parâmetros *default*. No Apêndice C, são encontrados maiores detalhes desse arquivo de configuração do AG bem como as configurações *default* mencionadas.

Identificador (FitnessFunction)	Método Utilizado para Classificar Novos Exemplos	Função Utilizada para Avaliar o Desempenho do Classificador
GA.sracc.hqacc	SR_{Acc}	HQ_{Acc}
GA.srlap.hqacc	SR_{Lacc}	
GA.srcov.hqacc	SR_{Cov}	
GA.srpe.hqacc	SR_{Ipe}	
GA.mracc.hqacc	MR_{Acc}	
GA.mrlap.hqacc	MR_{Lacc}	
GA.mrcov.hqacc	MR_{Cov}	
GA.mrpe.hqacc	MR_{Ipe}	
GA.sracc.hqprec	SR_{Acc}	HQ_{Prec}
GA.srlap.hqprec	SR_{Lacc}	
GA.srcov.hqprec	SR_{Cov}	
GA.srpe.hqprec	SR_{Ipe}	
GA.mracc.hqprec	MR_{Acc}	
GA.mrlap.hqprec	MR_{Lacc}	
GA.mrcov.hqprec	MR_{Cov}	
GA.mrpe.hqprec	MR_{Ipe}	
GA.sracc.hqf1	SR_{Acc}	HQ_{F1}
GA.srlap.hqf1	SR_{Lacc}	
GA.srcov.hqf1	SR_{Cov}	
GA.srpe.hqf1	SR_{Ipe}	
GA.mracc.hqf1	MR_{Acc}	
GA.mrlap.hqf1	MR_{Lacc}	
GA.mrcov.hqf1	MR_{Cov}	
GA.mrpe.hqf1	MR_{Ipe}	
GA.sracc.hqpc	SR_{Acc}	HQ_{PC}
GA.srlap.hqpc	SR_{Lacc}	
GA.srcov.hqpc	SR_{Cov}	
GA.srpe.hqpc	SR_{Ipe}	
GA.mracc.hqpc	MR_{Acc}	
GA.mrlap.hqpc	MR_{Lacc}	
GA.mrcov.hqpc	MR_{Cov}	
GA.mrpe.hqpc	MR_{Ipe}	

Tabela 6: Identificadores que indicam qual função de avaliação utilizar na execução do algoritmo genético

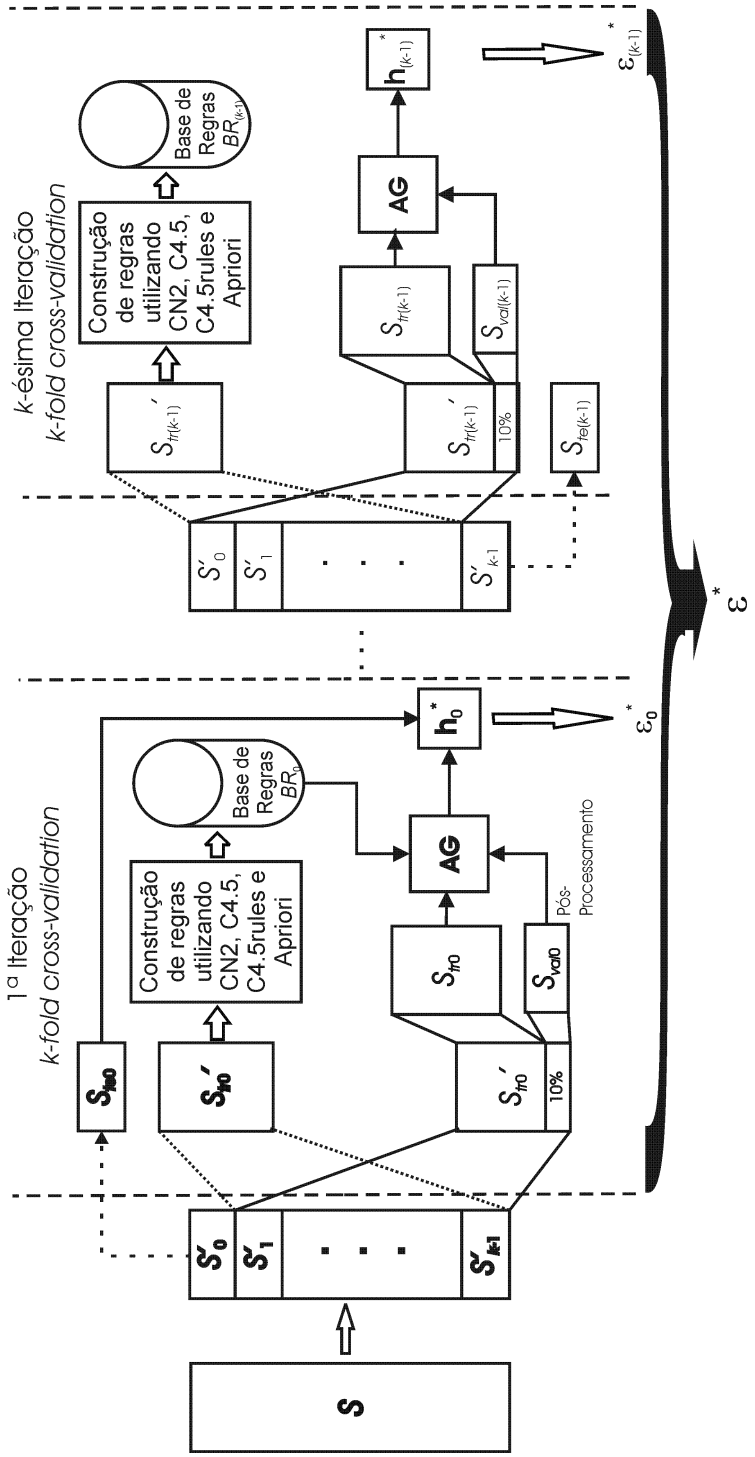

Figura 8: Estimativa de erro ε^* de um classificador $\mathbf{h}^*$ evoluído com o AG proposto

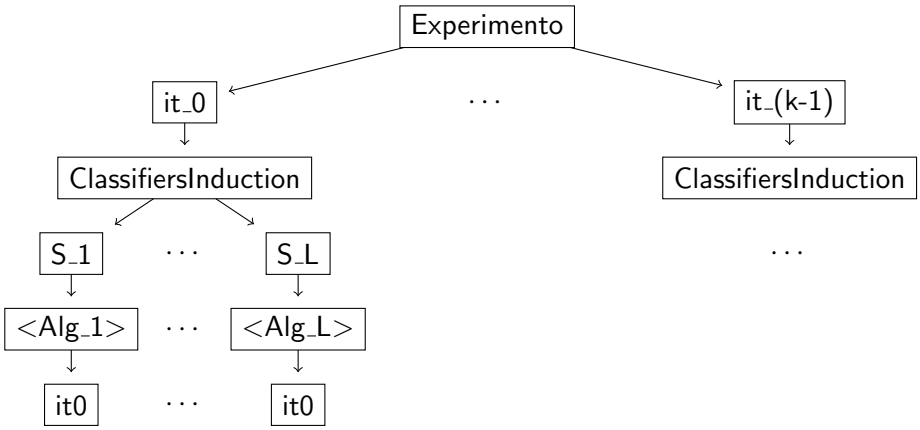

Figura 9: Configuração da árvore de diretórios para induzir os classificadores simbólicos cujas regras compõem a população inicial do(s) AG(s) a ser(em) executado(s)

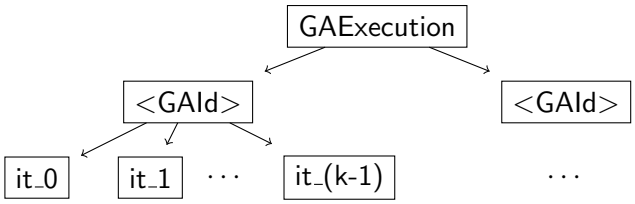

Figura 10: Configuração da árvore de diretórios para executar o algoritmo genético proposto

Exemplo de Construção das Árvores de Diretórios: Para facilitar o entendimento de como são construídas essas árvores de diretórios, vamos ilustrar a construção delas com um exemplo. Vamos supor que queremos avaliar o AG proposto utilizando os identificadores de AG GA.mracc.hqprec, GA.mracc.hqpc e GA.srlap.hqprec bem como utilizando *10-fold cross-validation*. Vamos supor também que representam o conjunto de dados completo os arquivos ds.data e ds.names. Para evoluir uma população inicial de classificadores construída aleatoriamente, queremos que as regras que compõem essa população inicial sejam provenientes dos classificadores induzidos por $\mathcal{CN}2$, $\mathcal{C}4.5$ e $\mathcal{C}4.5rules$ e provenientes do conjunto de regras construído pelo $\mathcal{A}$priori. Assim, inicialmente deve ser construída uma instanciação da árvore geral ilustrada na Figura 9. Tal instanciação é ilustrada na Figura 11. Após execução do SNIFFER nessa árvore, cada diretório folha it0 vai conter uma hipótese (ou um conjunto de regras caso o algoritmo para construção de regras seja o $\mathcal{A}$priori). Assim, a árvore ilustrada na Figura 12 é construída. Nessa figura, o diretório

GAExecution → GA.mracc.hqprec → **it_0**

contém os arquivos com regras presentes nos diretórios

Experimento → **it_0** → ClassifiersInduction → S_1 → c4.5 → it0,
Experimento → **it_0** → ClassifiersInduction → S_2 → c4.5rules → it0,
Experimento → **it_0** → ClassifiersInduction → S_3 → cn2 → it0 e
Experimento → **it_0** → ClassifiersInduction → S_4 → apriori → it0

Tais arquivos são nomeados, respectivamente, ds1.out, ds2.out, ds3.out e ds4.out, e são convertidos para a sintaxe padrão $\mathcal{PBM}$, renomeados para ds1.stdrules, ds2.stdrules, ds3.stdrules e ds4.stdrules. Também, contém os arquivos representando os conjuntos de dados de validação S_{val_0} e teste S_{te_0} presentes no diretório Experimento → **it_0**. Já o diretório

GAExecution → GA.mracc.hqprec → **it_1**

contém os arquivos com regras presentes nos diretórios

Experimento → **it_1** → ClassifiersInduction → S_1 → c4.5 → it0,
Experimento → **it_1** → ClassifiersInduction → S_2 → c4.5rules → it0,
Experimento → **it_1** → ClassifiersInduction → S_3 → cn2 → it0 e
Experimento → **it_1** → ClassifiersInduction → S_4 → apriori → it0

também nomeados, respectivamente, ds1.out, ds2.out, ds3.out e ds4.out, e são convertidos para a sintaxe padrão $\mathcal{PBM}$, renomeados para ds1.stdrules, ds2.stdrules, ds3.stdrules e ds4.stdrules, e contém os arquivos representando os conjuntos de dados de validação S_{val_1} e teste S_{te_1} presentes no diretório Experimento → **it_1**. Esse processo é repetido para as iterações seguintes. Tal estrutura se repete nos diretórios GAExecution → GA.mracc.hqpc e GAExecution → GA.srlap.hqprec. Então, SNIFFERGA (descrito no Apêndice B) é chamado nessa árvore de diretórios, chamando o AG passando qual o método de classificação e qual a função de avaliação a ser utilizada.

Deve ser observado que árvore construída pelo GATREE, ilustrada na Figura 12, pode ser construída "à mão", ou seja, sem utilizar o módulo GATREE. Neste caso, supondo que os arquivos referentes ao conjunto de dados sejam ds.data e ds.names, precisam estar presentes diretórios

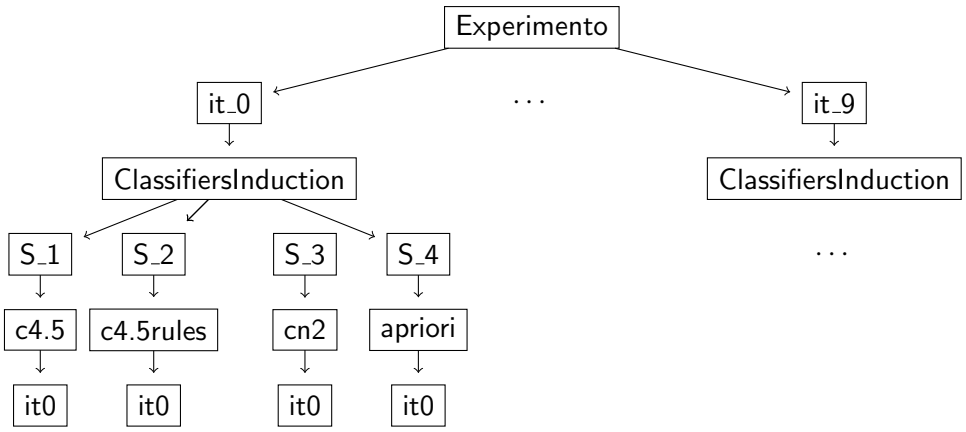

Figura 11: Árvore de diretórios instanciada para induzir os classificadores simbólicos cujas regras compõem a população inicial do(s) AG(s) a ser(em) executado(s)

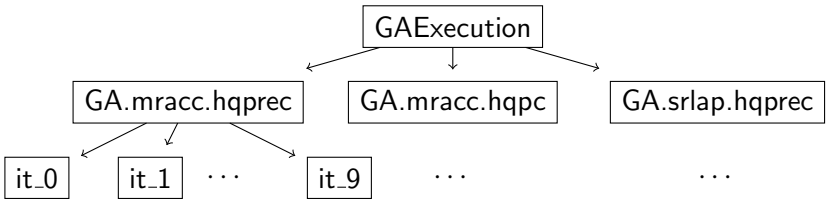

Figura 12: Instanciação da árvore de diretórios para executar os AGs representados pelos identificadores GA.mracc.hqprec, GA.mracc.hqpc e GA.srlap.hqprec

it_i, $i = 0, ..., (k-1)$, os arquivos ds.data, ds.names, ds.validation, ds.test, referentes ao conjunto de dados, e um arquivo ds1.stdrules contendo regras criadas pelo usuário na sintaxe padrão $\mathcal{PBM}$, além de arquivos ds$j$.stdrules, $j = 2, ..., L$, também na sintaxe padrão, provenientes de algoritmos de aprendizado ou de outras fontes, a critério do usuário.

6.2 O Módulo GATree

Lembrando que, para execução do algoritmo genético, em nossa proposta é necessário ter regras iniciais para construir a população inicial de indivíduos, e que utilizaremos algoritmos de aprendizado para construir essas regras iniciais, para facilitar tais tarefas utilizamos as facilidades do ambiente Sniffer e do módulo EnsembleTree para construir o módulo GATree. GATree é uma extensão da classe EnsembleTree, única classe pertencente ao módulo EnsembleTree. A classe EnsembleTree foi modificada de forma a permitir que somente sejam construídos os classificadores, sem que suas taxas de erro fossem estimadas. A tarefa da classe GATree é construir a árvore de diretórios onde se executa o algoritmo genético (Figura 10). Assim, para criar um objeto da classe GATree, é necessário passar como parâmetro um objeto EnsembleTree, onde estão presentes as regras construídas pelos algoritmos $\mathcal{CN}2$, $\mathcal{C}4.5$, $\mathcal{C}4.5rules$ e $\mathcal{A}$priori.

6.3 O Módulo RuleStruct

O módulo RuleStruct armazena e gerencia a base de regras utilizada para compor os indivíduos da população inicial e as regras utilizadas para realizar as operações de mutação e *crossover*. As regras armazenadas em uma mesma instância do módulo RuleStruct devem pertencer a um domínio comum, isto é, devem fazer referência a um conjunto comum de atributos. Em outras palavras, utilizam exemplos do mesmo domínio. As regras são armazenadas em uma instância do módulo RuleStruct na sintaxe padrão de regras $\mathcal{PBM}$ (Prati et al., 2001b). Também, para as regras armazenadas, o módulo calcula a matriz de contingência das regras para avaliação das regras armazenadas.

A principal classe do módulo RuleStruct é também denominada RuleStruct. Essa classe cria uma base de regras que contém todas as regras dos classificadores simbólicos já na sintaxe $\mathcal{PBM}$. Para carregar essas regras na base é utilizada a classe LoadRule. Como mencionado, os L classificadores de entrada podem ser induzidos utilizando L sistemas de aprendizado diferentes sobre um mesmo conjunto de dados; utilizando um ou mais sistemas de aprendizado, com variações dos parâmetros desses sistemas de aprendizado sobre o mesmo conjunto de dados ou, ainda, utilizando um ou mais sistemas de aprendizado sobre diversas amostras de um mesmo conjunto de dados. Tais classificadores são induzidos utilizando o módulo GATree, descrito anteriormente.

O módulo RuleStruct provê diversas funcionalidades para a manipulação de regras. Entre as principais estão prover informações sobre a matriz de contingência das regras e sobre o sistema de aprendizado que induziu as regras; informar quais são as regras *default* e remover essas regras da base de regras; prover informações sobre a complexidade sintática das regras e verificar se uma regra cobre um determinado exemplo, entre outros. As regras armazenadas na base são referenciadas por meio de um índice, isto é, cada regra possui um índice único que a identifica. Os demais módulos do ambiente GAERE que utilizam o módulo RuleStruct armazenam somente os índices das regras armazenadas, e utilizam esses índices para realizar operações sobre as regras.

6.4 O Módulo RuleSetGA

O módulo RuleSetGA é responsável por construir e manipular os indivíduos do AG, inclusive classificar um conjunto de exemplos dado um indivíduo e dado o método de classificação que deve ser utilizado (SR_{Acc}, SR_{Lacc}, SR_{Cov}, SR_{Ipe}, MR_{Acc}, MR_{Lacc}, MR_{Cov} e MR_{Ipe} — Seção 4.3 na página 22). Esse módulo é composto pelas classes RuleSetGA, RuleSetSRAcc, RuleSetSRLap, RuleSetSRCov, RuleSetSRPE, RuleSetMRAcc, RuleSetMRLap, RuleSetMRCov e RuleSetMRPE.

6.5 O Módulo GA

O módulo GA é composto pelas classes GA, HypQualityAcc, HypQualityPrec, HypQualityLap, HypQualityF1, HypQualityAC e HypQualityPC. A classe GA implementa o AG proposto na Seção 4. Inclusive, o AG implementado inicializa os indivíduos iniciais. Como parâmetro *default* do AG, são também carregados como indivíduos iniciais os classificadores induzidos pelos algoritmos de AM simbólicos. Entretanto, o usuário pode pedir para não serem carregados os classificadores como indivíduos, utilizando um parâmetro do AG — o LOADCLASS. Para maiores

informações sobre os parâmetros do AG proposto, veja o Apêndice C.

Para avaliar os indivíduos em cada geração do AG, são utilizadas as classes HypQualityAcc, HypQualityPrec, HypQualityLap, HypQualityF1, HypQualityAC e HypQualityPC. Na Figura 13 é apresentado o projeto dos módulos RuleStruct, RuleSetGA e GA por meio de um diagrama de classes em UML[17] (Booch et al., 1998).

7 Conclusão

Neste trabalho, foi proposto um algoritmo genético para evoluir classificadores simbólicos. As operações que inserem variabilidade nos indivíduos ao decorrer das gerações do AG são semelhantes às operações encontradas no algoritmo genético clássico. Entretanto, a maneira de codificação dos indivíduos proposta é bastante interessante por ser simples e eficaz. Também, as funções de avaliação propostas combinam diversas maneiras de classificação de exemplos por parte dos classificadores indivíduos e diversas maneiras de avaliar esses classificadores. O projeto e a implementação do sistema computacional GAERE, que implementa o AG proposto, foi descrito em detalhes. Deve ser destacado que no sistema GAERE também encontra-se implementada uma maneira de automatizar a tarefa de avaliação do AG utilizando a técnica de *k-fold cross-validation* estratificado. Essa funcionalidade é muito importante para facilitar a execução de experimentos e testes estatísticos para avaliar os resultados experimentais.

[17] *Unified Modeling Language*

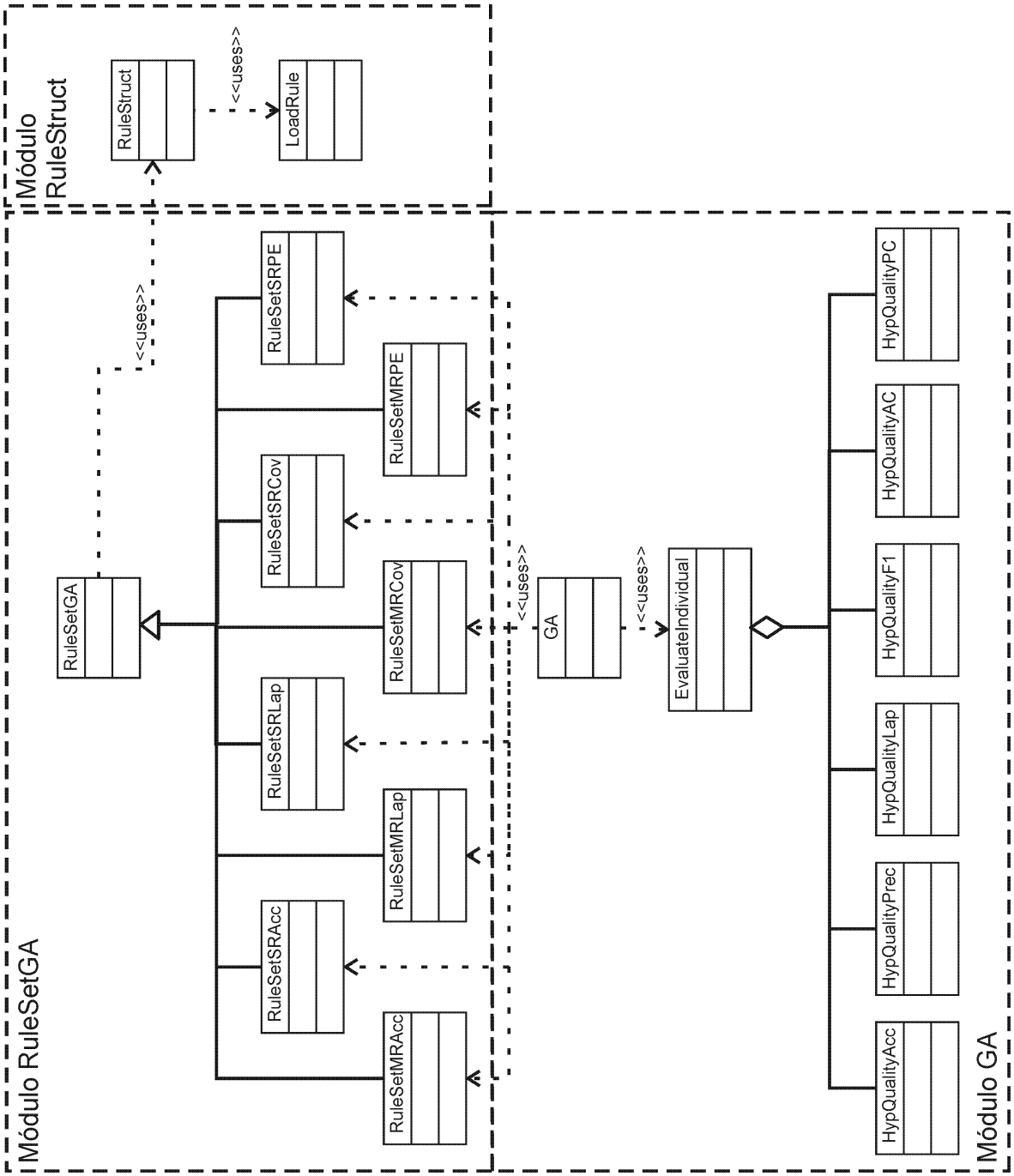

Figura 13: Diagrama UML que envolve os módulos RuleStruct, RuleSetGA e GA

A Análise Qualitativa de Regras

Sob o aspecto qualitativo, regras são avaliadas com o objetivo de saber quais são aquelas melhor sustentadas pelos dados (Lavrac et al., 1999). A análise qualitativa de regras pode ser efetuada por meio de procedimentos que implementam medidas de avaliação de regras. Nesta seção, são descritas as medidas de avaliação de regras unificadas no *framework* proposto por Lavrac et al. (1999), bem como são descritos outros critérios de avaliação de regras. Em (Gomes, 2002) encontra-se uma descrição mais abrangente que inclui outras medidas de avaliação de regras.

A.1 Uma Visão Unificada de Medidas de Avaliação de Regras

Esta seção está baseada no trabalho de Lavrac et al. (1999). O objetivo do *framework* proposto nesse trabalho é proporcionar uma análise unificada de medidas de avaliação de regras selecionadas da literatura. Para calcular essas medidas de avaliação de regras, é necessário construir a matriz de contingência de cada regra a ser avaliada. A maneira de se construir a matriz de contingência de uma regra está descrita na Seção 2.4 na página 7. Nesse *framework*, algumas medidas de avaliação encontradas na literatura foram unificadas, considerando uma regra R na forma $B \rightarrow H$. Essas medidas, definidas nas respectivas tabelas, podem ser divididas nas seguintes categorias:

1. Medidas genéricas de avaliação — Tabela 7 na página 46;
2. Medidas relativas de avaliação — Tabela 8 na página 47;
3. Medidas relativas de avaliação com peso — Tabela 9 na página 47.

Baseada na medida de novidade, Lavrac mostra que as medidas relativas de avaliação com peso são iguais entre si e iguais a medida de novidade da regra, *i.e.*

$$\begin{aligned} WRAcc(B \rightarrow H) &= WRSens(B \rightarrow H) = WRSpec(B \rightarrow H) \\ &= WRNegRel(B \rightarrow H) = Nov(B \rightarrow H) \end{aligned} \tag{35}$$

Geralmente, medidas de avaliação são úteis para determinar quais são as "melhores" regras de um classificador simbólico $\mathbf{h}$. As "melhores" regras podem ser aquelas que apresentam o maior (ou menor) valor para uma determinada medida de avaliação. Por exemplo, pode ser considerado que as "melhores" regras R_u de um classificador simbólico $\mathbf{h}$, são aquelas que possuem maior valor para a medida de precisão (ou menor valor para a medida de erro).

A.2 Outros Critérios para Seleção da Melhor Regra

Existem outros critérios de seleção da melhor regra de uma hipótese $\mathbf{h}$ (Horst and Monard, 2000; Gomes, 2002). Um dos possíveis critérios consiste em escolher a regra com o melhor grau de adequação (*rule fitness*) para o problema em questão. Dada uma regra R, na forma $B \rightarrow H$, o grau de adequação de R pode ser denotado por $rf(B \rightarrow H)$. O critério de seleção rf escolhe a melhor regra como sendo aquela que possui maior valor de rf. Existem três possíveis critérios que permitem estimar o grau de adequação de uma regra:

1. Confiabilidade Positiva;
2. Precisão de Laplace; e
3. Novidade

descritas a seguir.

Confiabilidade Positiva: Utiliza a precisão de regra (Acc), definida pela Equação 18 na página seguinte. Entretanto, a confiabilidade positiva possui uma propriedade indesejada. Por exemplo, dadas duas regras R_u e R_q, supondo que $Acc(R_u) = 1$ e $Acc(R_q) = 0.98$, segundo este critério a regra R_u é melhor que a regra R_q. Entretanto, se a regra R_u cobre somente um exemplo e a regra R_q cobre 200 exemplos (portanto, 4 deles são cobertos erroneamente), a regra R_q é claramente melhor que a regra R_u.

Precisão de Laplace Uma solução para o problema apresentado pela confiabilidade positiva (Acc) consiste em substituí-la pela precisão de Laplace (Clark and Boswell, 1991), definida pela Equação 36, onde, como definido anteriormente, N_{Cl} é o número de classes do conjunto de exemplos, bh é o número de exemplos cobertos corretamente pela regra e $b\overline{h}$ é o número de exemplos cobertos incorretamente pela regra.

$$rf(B \rightarrow H) \stackrel{def}{=} Lacc(B \rightarrow H) = \frac{bh+1}{b+N_{Cl}} = \frac{bh+1}{bh+b\overline{h}+N_{Cl}} \tag{36}$$

Considerando esse critério, quanto maior o valor da precisão de Laplace, melhor é a regra. Ainda considerando o exemplo anterior em um problema com $N_{Cl} = 2$ classes, tem-se que $Lacc(R_u) = 0.66$ e $Lacc(R_q) = 0.98$, indicando que a regra R_q é melhor que a regra R_u. Assim como a confiabilidade positiva, a precisão de Laplace assume valores no intervalo $0 \leq Lacc(B \rightarrow H) \leq 1$.

Novidade: Utiliza a novidade de regra (Nov), definida pela Equação 25 na próxima página. Em (Todorovski et al., 2000), são mostrados alguns experimentos nos quais se utiliza a medida de novidade no algoritmo $\mathcal{CN}2$ para induzir classificadores compostos por regras *ordered* e *unordered*. Nos experimentos nos quais foram construídas hipóteses com regras *unordered*, o uso da medida de novidade reduziu, em média, nove vezes o número de regras não ordenadas induzidas pelo algoritmo $\mathcal{CN}2$, com um aumento de erro em torno de 5%. Essa redução do número de regras, com pequena perda de precisão, constitui um avanço no sentido de uma maior compreensibilidade do conjunto final de regras por seres humanos, o que torna a medida novidade bastante interessante.

Precisão

$$Acc(B \rightarrow H) = P(H|B) = \frac{hb}{b} \tag{18}$$

Erro

$$Err(B \rightarrow H) = P(\overline{H}|B) = \frac{\overline{h}b}{b} \tag{19}$$

Confiança Negativa

$$NegRel(B \rightarrow H) = P(\overline{H}|\overline{B}) = \frac{\overline{hb}}{\overline{b}} \tag{20}$$

Sensibilidade

$$Sens(B \rightarrow H) = P(B|H) = \frac{hb}{h} \tag{21}$$

Especificidade

$$Spec(B \rightarrow H) = P(\overline{B}|\overline{H}) = \frac{\overline{hb}}{\overline{h}} \tag{22}$$

Cobertura

$$Cov(B \rightarrow H) = P(B) = \frac{b}{N} \tag{23}$$

Suporte

$$Sup(B \rightarrow H) = P(HB) = \frac{hb}{N} \tag{24}$$

Novidade

$$Nov(B \rightarrow H) = P(HB) - P(H)P(B) = \frac{1}{N}(hb - \frac{h \cdot b}{N}) \tag{25}$$

Satisfação

$$Sat(B \rightarrow H) = \frac{(P(\overline{H}) - P(\overline{H}|B))}{P(\overline{H})} = \frac{\frac{\overline{h}}{N} - \frac{\overline{h}b}{b}}{\frac{\overline{h}}{N}} = 1 - \left(\frac{\overline{h}b}{b} \cdot \frac{N}{\overline{h}}\right) \tag{26}$$

Tabela 7: Medidas genéricas de avaliação de regras

Precisão Relativa

$$RAcc(B \rightarrow H) = P(H|B) - P(H) = \frac{hb}{b} - \frac{h}{N} \tag{27}$$

Confiança Negativa Relativa

$$RNegRel(B \rightarrow H) = P(\overline{H}|\overline{B}) - P(\overline{H}) = \frac{\overline{hb}}{\overline{b}} - \frac{\overline{h}}{N} \tag{28}$$

Sensibilidade Relativa

$$RSens(B \rightarrow H) = P(B|H) - P(B) = \frac{hb}{h} - \frac{b}{N} \tag{29}$$

Especificidade Relativa

$$RSpec(B \rightarrow H) = P(\overline{B}|\overline{H}) - P(\overline{B})\frac{\overline{hb}}{\overline{h}} - \frac{\overline{b}}{N} \tag{30}$$

Tabela 8: Medidas relativas de avaliação de regras

Precisão Relativa com Peso

$$WRAcc(B \rightarrow H) = P(B)(P(H|B) - P(H)) = \frac{b}{N}\left(\frac{hb}{b} - \frac{h}{N}\right) \tag{31}$$

Confiança Negativa Relativa com Peso

$$WRNegRel(B \rightarrow H) = P(\overline{B})(P(\overline{H}|\overline{B}) - P(\overline{H})) = \frac{\overline{b}}{N}\left(\frac{\overline{hb}}{\overline{b}} - \frac{\overline{h}}{N}\right) \tag{32}$$

Sensibilidade Relativa com Peso

$$WRSens(B \rightarrow H) = P(H)(P(B|H) - P(B)) = \frac{h}{N}\left(\frac{hb}{h} - \frac{b}{N}\right) \tag{33}$$

Especificidade relativa com Peso

$$WRSpec(B \rightarrow H) = P(\overline{H})(P(\overline{B}|\overline{H}) - P(\overline{B})) = \frac{\overline{h}}{N}\left(\frac{\overline{hb}}{\overline{h}} - \frac{\overline{b}}{N}\right) \tag{34}$$

Tabela 9: Medidas relativas de avaliação de regras com peso

B Extensão do Ambiente Sniffer

O ambiente computacional Sniffer foi proposto e desenvolvido com o objetivo de gerenciar avaliações e comparações experimentais de algoritmos de aprendizado (Batista and Monard, 2003). Para estender as funcionalidades do ambiente Sniffer para atender as necessidades dos ambientes ELE (Bernardini and Monard, 2004) e do sistema GAERE, descrito neste trabalho, foi necessário realizar uma extensão do ambiente Sniffer. Na Seção 5.1 na página 26, é realizada uma breve descrição do ambiente computacional Sniffer originalmente proposto e desenvolvido. Neste apêndice, realizamos uma descrição para descrever as extensões realizadas no ambiente Sniffer para atender as necessidades dos sistemas computacionais ELE e GAERE.

Para realizar as extensões citadas, foram projetadas e implementadas as classes SnifferELE, SnifferGA e SnifferClassInd, as quais reimplementam alguns métodos da superclasse Sniffer. O objetivo desta extensão é fazer com que o Sniffer possa reconhecer os identificadores de métodos de construção de *ensembles* (Bernardini and Monard, 2004) — SnifferELE — , os identificadores de algoritmos genéticos — SnifferGA — e realizar somente indução de classificadores, sendo assim o $\mathcal{A}$priori também reconhecido pelo Sniffer — SnifferClassInd. Os identificadores a serem reconhecidos pelo Sniffer são:

SnifferGA: Reconhece todos os identificadores relacionados a execução do AG, listados na Tabela 6 na página 36.

SnifferELE: Reconhece todos os identificadores relacionados com métodos de construção de *ensembles* simbólicos (Bernardini and Monard, 2004).

SnifferClassInd: É responsável por indicar ao Sniffer que somente sejam induzidos classificadores sem estimativa de erro. Além dos algoritmos de aprendizado já reconhecidos pelo Sniffer, SnifferClassInd também reconhece o $\mathcal{A}$priori, representado pelo identificador apriori.

Na Figura 14 é ilustrado o diagrama UML referente ao projeto das classes SnifferELE, SnifferGA e SnifferClassInd, as quais, como mencionado, são extensões da superclasse Sniffer.

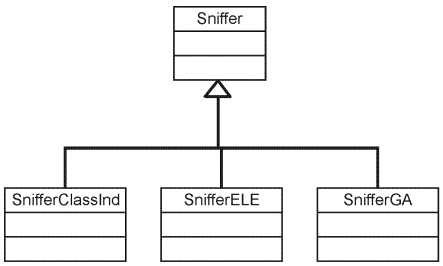

Figura 14: Diagrama UML da extensão realizada no ambiente Sniffer

C Configurações do Algoritmo Genético

O algoritmo genético proposto possui alguns parâmetros que possuem valores *default*. Na Tabela 10 são mostrados esses parâmetros, os valores *default* que esses parâmetros podem assumir e uma descrição de cada um desses parâmetros.

Parâmetro	Valor *Default*	Descrição
STPCRIT	converge	Tipo de critério de parada do AG. Também pode assumir o valor "numgen".
LOADCLASS	1	1: São carregados como indivíduos iniciais os conjuntos de regras que são classificadores induzidos por algoritmos de aprendizado simbólico $\mathcal{CN}2$, $\mathcal{C}4.5$ e $\mathcal{C}4.5rules$. 0: Não são carregados tais classificadores como indivíduos.
PLOTCHART	0	1: O gráfico com a média dos valores obtidos por cada indivíduo na função de avaliação e a o melhor valor dos indivíduos na função de avaliação a cada iteração (geração) do AG são plotados em um gráfico. 0: O gráfico não é plotado.
NREND	5	Usado no critério de parada "converge". O algoritmo pára se a função de avaliação do melhor indivíduo não melhora durante NRTERM gerações.
NRGEN	20	Usado no critério de parada "numgen". Número de gerações a ser executado pelo AG.
IS	10	Tamanho inicial de cada indivíduo.
PS	15	Tamanho da população.
PC	0.40	Probabilidade de *crossover*.
PM	0.01	Probabilidade de mutação.

Tabela 10: Parâmetros do AG proposto

Quando a classe que implementa o AG é instanciada, é verificado se existe no diretório onde estão os arquivos de regras e os arquivos de dados um arquivo denominado GA.conf. O formato deste arquivo é ilustrado na Figura 15. Caso esse arquivo não exista, o AG é executado com todos os valores *default*. Mesmo se o arquivo existir, se algum parâmetro, ou todos eles, não estiverem citados no arquivo, o parâmetro fica com seu valor *default*.

```
GA CONFIGURATION FILE

STPCRIT <stop_criterion>
PLOTCHART <plot_chart>
LOADCLASS <load_classifiers>
NREND <nr_end>
NRGEN <nr_generations>
IS <individual_size>
PS <population_size>
PC <crossover_probability>
PM <mutation_probability>
```

Figura 15: Arquivo de configurações do AG

Referências

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *Proceedings of the 20th International Conference on Very Large Data Bases — VLDB'94*, pages 487–499. Morgan Kaufmann Publishers. Citado na página 11.

Baranauskas, J. A. (2001). Extração automática de conhecimento por múltiplos indutores. Tese de Doutorado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08102001-112806/. Citado na página 26.

Baranauskas, J. A. and Batista, G. E. A. P. A. (2000). O projeto Discover: Idéias iniciais (comunicação pessoal). Citado nas páginas 1 e 25.

Batista, G. E. A. P. A. (2001). Sintaxe padrão do arquivo de exemplos do projeto Discover. http://www.icmc.sc.usp.br/~gbatista/Discover/SintaxePadraoFinal.htm. Citado na página 26.

Batista, G. E. A. P. A. (2003). Pré-processamento de dados em aprendizado de máquina supervisionado. Tese de Doutorado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06102003-160219/. Citado nas páginas 25, 26, e 27.

Batista, G. E. A. P. A. and Monard, M. C. (2003). Descrição da arquitetura e do projeto do ambiente computacional Discover Learning Environment - DLE. Technical Report 187, ICMC/USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.PDF. Citado nas páginas 26, 27, 28, 29, e 48.

Batista, G. E. A. P. A. and Monard, M. C. (2006). The Discover Object Library (DOL) User's Manual. Technical Report em preparação, ICMC/USP. Citado na página 26.

Bernardini, F. C. (2002). Combinação de classificadores simbólicos para melhorar o poder preditivo e descritivo de *ensembles*. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-29042003-091747/. Citado na página 26.

Bernardini, F. C. and Monard, M. C. (2004). Descrição da arquitetura e do projeto do sistema computacional para construção de ensembles de classificadores simbólicos ELE. Technical Report 243, ICMC/USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_243.pdf. Citado nas páginas 1, 26, 32, 33, e 48.

Bernardini, F. C. and Monard, M. C. (2005a). Methods for constructing symbolic ensembles from symbolic classifiers. In *Frontiers in Artificial Intelligence and Applications*, volume 132, pages 161–168, Hyogo, Japan. Netherlands: IOS Press. Citado na página 1.

Bernardini, F. C. and Monard, M. C. (2005b). Uma proposta para a construção de ensembles simbólicos que explicam suas decisões. In *Conferencia Latinoamericana de Informatica — CLEI 2005*, volume 1, pages 151–162, Cali, Colômbia. Cali: Feriva. Citado na página 1.

Bernardini, F. C., Monard, M. C., and Prati, R. C. (2005). Constructing ensembles of symbolic classifiers. In *International Conference on Hybrid Intelligent Systems — HIS 2005*, volume 1, pages 315–320, Rio de Janeiro. California: IEEE Computer Society. Citado na página 1.

Blake, C., Keogh, E., and Merz, C. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html. Citado na página 29.

Booch, G., Jacobson, I., Rumbaugh, J., and Rumbaugh, J. (1998). *The Unified Modeling Language User Guide*. Addison Wesley. Citado na página 42.

Borgelt, C. and Kruse, R. (2002). Induction of association rules: A priori implementation. In *Proceedings of 15th Conference on Computational Statistics*. Citado nas páginas 21 e 34.

Boswell, T. (1990). Manual for NewId version 4.1. Technical Report TI/P2154/RAB/4/2.3, The Turing Institute. Citado nas páginas 27 e 30.

Cabena, P., Hadjinian, P., Stadler, R., Verhees, J., and Zanasi, A. (1998). *Discovering Data Mining: from Concept to Implementation*. Prentice Hall. Citado na página 1.

Carvalho, A. C. P. L. F., Braga, A. P., and Ludermir, T. B. (2003). *Computação Evolutiva*, chapter 9, pages 225–248. In Rezende (2003). Citado na página 15.

Chiara, R. (2003). Aplicações de técnicas de *data mining* em *logs* de servidores web. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19012004-093205/. Citado na página 26.

Clark, P. and Boswell, R. (1991). Rule induction with $\mathcal{CN}2$: Some recent improvements. In Kodratoff, Y., editor, *Proceedings of the 5th European Working Session on Learning (EWSL 91)*, pages 151–163. Citado nas páginas 27, 30, e 45.

Clark, P. and Niblett, T. (1989). The $\mathcal{CN}2$ induction algorithm. *Machine Learning*, 3(4):261–283. Citado nas páginas 21 e 34.

De Jong, K. A., Spears, W., and Gordon, D. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188. Citado na página 20.

Dietterich, T. G. (1989). Limitations on inductive learning (extended abstract). In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 124–128, Ithaca, NY. Morgan Kaufmann Publishers. Citado na página 34.

Dosualdo, D. G. (2003). Investigação de regressão no processo de mineração de dados. Dissertação de Mestrado, ICMC/USP. Citado na página 26.

Ferro, M. (2004). Aquisição de conhecimento de conjuntos de exemplos no formato atributo valor utilizando aprendizado de máquina relacional. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-16112004-095938/. Citado na página 26.

Ferro, M. (2006). Aquisição de conhecimeno utilizando aprendizado de máquina relacional. Qualificação de Doutorado, ICMC/USP. Citado na página 26.

Fischer, S., Klinkenberg, R., Mierswa, I., and Ritthoff, O. (2002). Yale: Yet another learning environment — tutorial. Technical Report CI-136/02, University of Dortmund, Collaborative Research Center 531. Citado na página 25.

Freitas, A. A. (1998a). A multi-criteria approach for the evaluation of rule interestingness. In *Proceedings of the International Conference on Data Mining*, pages 7–20, Rio de Janeiro, RJ. Citado na página 10.

Freitas, A. A. (1998b). On objective measures of rule surprisingness. In *Proc. 2nd European Symp. on Principles of Data Mining and Knowledge Discovery (PKDD-98). Lecture Notes in Artificial Intelligence*, volume 1510, pages 1–9. Springer Verlag. Citado na página 10.

Freitas, A. A. (2002a). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer Verlag. Citado nas páginas 1, 11, 12, 15, 19, e 22.

Freitas, A. A. (2002b). A survey of evolutionary algorithms for data mining and knowledge discovery. In Ghosh, A. and Tsui, S., editors, *Advances in Evolutionary Computation*. Citado na página 19.

Giordana, A. and Neri, F. (1995). Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416. Citado na página 20.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley. Citado na página 13.

Gomes, A. K. (2002). Medidas de avaliação de regras. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-04072002-144610/. Citado nas páginas 26 e 44.

Greene, D. and Smith, S. (1993). Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257. Citado na página 20.

Hilderman, R. J. and Hamilton, H. J. (1999). Knowledge discovery and interestingness measures: A survey. Technical Report s4s, University of Regina. Citado na página 10.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. The MIT Press. Citado na página 13.

Horst, P. and Monard, M. C. (2000). Um sistema computacional para avaliação de regras induzidas por algoritmos de aprendizado de máquina. In *Proceedings IBERAMIA-SBIA 2000 Open Discussion Track*, pages 167–176, Atibaia, SP, Brasil. Citado na página 44.

Janikow, C. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13:189–228. Citado na página 20.

Kibler, D. and Langley, P. (1988). Machine learning as an experimental science. *Machine Learning*, 3(1):5–8. Citado na página 34.

Kohavi, R., Sommerfield, D., and Dougherty, J. (1994). *$\mathcal{MLC}$++: A Machine Learning Library in C++*. IEEE Computer Society Press. Citado na página 25.

Lavrac, N., Flach, P., and Zupan, B. (1999). Rule evaluation measures: a unifying view. In *Proc. 9th International Workshop on Inductive Logic Programming. Lecture Notes in Artificial Intelligence*, volume 1634, pages 74–185. Springer Verlag. Citado nas páginas 7, 10, 11, e 44.

Lee, H. D. (2005). Seleção de atributos importantes para a extração de conhecimento de bases de dados. Tese de Doutorado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-22022006-172219/publico/tese_huei.pdf. Citado na página 26.

Mahfoud, S. (1995). *Niching Methods for Genetic Algorithms*. PhD thesis, Univ. of Illinois at Urbana-Champaign. IlliGAL Report N° 95001. Citado na página 20.

Mannila, H. (1996). Data mining: Machine learning, statistics, and database. Technical Report FIN-00014, Department of Computer Science, University of Helsinki, Helsinki, Finland. http://www.cs.helsinki.fi/~mannila. Citado na página 1.

Martins, C. A. (2003). Uma abordagem para pré-processamento de dados textuais em algoritmos de aprendizado. Tese de Doutorado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08032004-164855/. Citado na página 26.

Matsubara, E. T. (2004). O algoritmo de aprendizado semi-supervisionado co-training e sua aplicação na rotulação de documentos. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19082004-092311/. Citado na página 26.

Matsubara, E. T., Martins, C. A., and Monard, M. C. (2003). *PRETEX*: Uma ferramenta para pré-processamento de textos utilizando a abordagem *Bag-of-Words*. Technical Report 209, ICMC/USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_209.pdf. Citado na página 26.

Melanda, E. (2004). Pós-processamento de conhecimento de regras de associação. Tese de Doutorado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-13012006-171753/publico/TeseEdsonMelandaVersaoCorrigida.pdf. Citado na página 26.

Michalewicz, Z. (1999). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag. Citado nas páginas 15 e 19.

Milaré, C. R. (2003). Extração de conhecimento de redes neurais artificiais utilizando sistemas de aprendizado simbólico e algoritmos genéticos. Tese de Doutorado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-11082004-004358/publico/claudia_milare.pdf. Citado nas páginas 15, 26, 31, 32, e 33.

Mitchell, M. (1997a). *An Introduction to Genetic Algorithms*. The MIT Press. Citado nas páginas 13 e 17.

Mitchell, T. (1997b). *Machine Learning*. McGraw Hill. Citado nas páginas 1 e 3.

Mitchell, T. (1999). Machine learning and data mining. *Communications of the ACM*, 42(11):30–36. http://cs.cmu.edu/user/mitchell/ftp/tomhome.html. Citado na página 1.

Monard, M. C. and Baranauskas, J. A. (2003). *Indução de Regras e Árvores de Decisão*, chapter 5, pages 115–140. In Rezende (2003). Citado na página 3.

Noda, E., Freitas, A. A., and Lopes, H. S. (1999). Discovering interesting prediction rules with a genetic algorithm. In *Proc. Conference on Evolutionary Computation – 1999 (CEC-99)*, pages 1322–1329, Washington D.C., USA. Citado na página 19.

Paula, M. F. (2003). Ambiente para disponibilização de conhecimento. Dissertaçao de Mestrado, ICMC/USP. Citado na página 26.

Pei, M., Goodman, E., and Punch, W. (1997). Pattern discovery from data using genetic algorithms. In *Proc. 1st Pacific-Asia Conf. Knowledge Discovery & Data Mining (PAKDD-97)*. Citado na página 20.

PERL (1999). *Programming in PERL*. Morgan Kaufmann Publishers, Inc. Citado nas páginas 25 e 31.

Pila, A. D. (2003). Algoritmos genéticos para construção de regras de conhecimento com propriedades específicas. Exame de Qualificação de Doutorado, ICMC/USP. Citado na página 26.

Prati, R. C. (2003). O framework de integração do sistema DISCOVER. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20082003-152116/. Citado na página 25.

Prati, R. C., Baranauskas, J. A., and Monard, M. C. (1999). BIBVIEW: Um sistema para auxiliar a manutenção de registros para o BIBTEX. Technical Report 95, ICMC/USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_95.ps.zip. Citado na página iii.

Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2001a). Extração de informações padronizadas para a avaliação de regras induzidas por algoritmos de aprendizado de máquina simbólico. Technical Report 145, ICMC/USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip. Citado na página 26.

Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2001b). Uma proposta de unificação da linguagem de representação de conceitos de algoritmos de aprendizado de máquina simbólicos. Technical Report 137, ICMC/USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. Citado nas páginas 26, 31, e 41.

Prati, R. C., Baranauskas, J. A., and Monard, M. C. (2002). Padronização da sintaxe e informações sobre regras induzidas a partir de algoritmos de aprendizado de máquina simbólico. *Revista Eletrônica de Iniciação Científica*, 2(3). http://www.sbc.org.br/reic/edicoes/2002e3. Citado nas páginas 21 e 26.

Pugliesi, J. B. (2004). O pós-processamento em extração de conhecimento de bases de dados. Tese de Doutorado, ICMC/USP. Citado na página 26.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1:81–106. Citado nas páginas 27 e 30.

Quinlan, J. (1987). Generating production rules from decision trees. In *Proceedings os the Tenth International Joint Conference in Artificial Intelligence*, pages 304–307, Italy. Citado nas páginas 27 e 30.

Quinlan, J. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers. Citado nas páginas 21, 27, 30, e 34.

Rezende, S. O. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*. Editora Manole, Barueri, SP, Brasil. Citado nas páginas 52, 54, e 55.

Rezende, S. O., Pugliesi, J. B., Melanda, E. A., and Paula, M. F. (2003). *Mineração de Dados*, chapter 12, pages 307–336. In Rezende (2003). Citado na página 1.

Rozante, T. A. A. (2003). Implantação do reuso de componentes no processo de desenvolvimento de software. Dissertação de Mestrado, ICMC/USP. Citado na página 25.

Sanches, M. K. (2003). Aprendizado de máquina semi-supervisionado: Proposta para rotular exemplos a partir de poucos exemplos rotulados. Dissertação de Mestrado, ICMC/USP. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-12092003-101358/. Citado na página 26.

Schaffer, C. (1994). A conservation law for generalization performance. In Cohen, W. W. and Hirsh, H., editors, *Eleventh International Conference on Machine Learning*, pages 259–265, New Brunswick, New Jersey. Morgan Kaufmann Publishers. Citado na página 34.

Todorovski, L., Flach, P., and Lavrac, N. (2000). A report on experiments with weighted relative accuracy in CN2. Technical Report CSTR-00-003, Department of Computer Science, University of Bristol. http://www.compsci.bristol.ac.uk/Tools/Reports/Ps/. Citado na página 45.

Weiss, G. M. and Hirsh, H. (1998). Learning to predict rare events in event sequences. In *Proc. 4th Int. Conf. KnowledgeDiscovery and Data Mining*, pages 359–363. AAAI Press. Citado na página 19.

Weiss, S. and Indurkhya, N. (1998). *Predictive Data Mining: a Practical Guide*. Morgan Kaufmann Publishers. http://www.data-miner.com.or. Citado na página 1.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2. edition. Citado na página 25.