Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Uma Proposta de Unificação da Linguagem de Representação de Conceitos de Algoritmos de Aprendizado de Máquina Simbólicos

Ronaldo Cristiano Prati
José Augusto Baranauskas
Maria Carolina Monard/ILTC

Nº 137

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Março/2001

Uma Proposta de Unificação da Linguagem de Representação de Conceitos de Algoritmos de Aprendizado de Máquina Simbólicos*

Ronaldo Cristiano Prati
José Augusto Baranauskas
Maria Carolina Monard/ILTC

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística
Laboratório de Inteligência Computacional
Caixa Postal 668, 13560-970 - São Carlos, SP, Brasil
e-mail: {prati, jaugusto, mcmonard}@icmc.sc.usp.br

Resumo

O trabalho com Aprendizado de Máquina - AM - é necessariamente empírico, ou seja, trabalha com a experimentação e comparação de diversos algoritmos na tentativa de se descobrir um que melhor se adapte a um domínio específico.

No entanto, a comparação entre esses algoritmos, para outros indicadores que não a precisão, não é uma tarefa trivial ou automática, pois cada algoritmo adota uma forma de representação de conhecimento diferente.

O objetivo deste trabalho é descrever a implementação de uma biblioteca de *scripts* PERL que convertem a descrição da hipótese induzida por alguns dos principais algoritmos de AM simbólicos para um formato específico de regras de produção: *if* <condição> *then* <class = C_i >, e, dessa forma, facilitar a análise, compreensão e comparação do conhecimento induzido por esses algoritmos.

A biblioteca desenvolvida converte atualmente nove algoritmos de AM — $\mathcal{ID}3$, $\mathcal{C}4.5$, $\mathcal{C}4.5rules$, $\mathcal{CN}2$, $\mathcal{OC}1$, $\mathcal{R}ipper$, $\mathcal{C}5.0$ /See5, $\mathcal{T}2$ e $\mathcal{MC}4$ — para um formato unificado, definido neste trabalho.

Palavras-Chave: Aprendizado de Máquina, Linguagem de Representação de Conceitos, Formato Padrão de Regras.

Março 2001

*Trabalho realizado com auxílio da CAPES, CNPq e FAPESP

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

[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

Com o advento da área de Aprendizado de Máquina – AM – em Inteligência Artificial, muitos pesquisadores implementaram algoritmos para aquisição automática de conhecimento similares, isoladamente, cada qual possuindo um formato próprio para os dados de entrada. Assim, quase sempre foi necessária uma transformação na estrutura e formato dos dados quando se escolhia um ou outro algoritmo. A razão de se utilizar mais de um algoritmo está no fato de que não existe um único bom algoritmo de AM (ou *indutor*) para todas as tarefas, como foi monstrado experimentalmente em (Kohavi et al., 1997).

O projeto $\mathcal{MLC}$++ (Machine Learning Library in C++), desenvolvido na *Stanford University*, e que hoje está sobre a reponsabilidade de *Silicon Graphics*, disponibiliza uma biblioteca de domínio público de classes e funções em C++, reunindo os algoritmos mais comuns de AM. Além de padronizar o formato dos dados de entrada, a biblioteca $\mathcal{MLC}$++ foi projetada para ajudar na seleção de algoritmos para realizar tarefas específicas e/ou no desenvolvimento de novos algoritmos mais convenientes para outras tarefas. Ela tem um papel dual, servindo tanto como um sistema para usuários finais quanto para projetistas de algoritmos. Essa biblioteca conta atualmente com mais de 30 algoritmos (Kohavi et al., 1994).

Pelas suas características de orientação a objeto (algoritmos $\mathcal{MLC}$++ são tratados como objetos C++), a biblioteca $\mathcal{MLC}$++ possui a habilidade de juntar classes e exemplos para a exploração de novos algoritmos ou variantes de algoritmos existentes, com a transformação automática dos formatos de arquivos de entrada para os diferentes algoritmos. $\mathcal{MLC}$++ possibilita ainda a obtenção de estatísticas de desempenho, comparação de diferentes algoritmos em diferentes conjuntos de exemplos, visualização gráfica das estruturas e conceitos aprendidos, entre outras funcionalidades (Kohavi et al., 1996).

Todavia, a biblioteca $\mathcal{MLC}$++ facilita principalmente a comparação de indutores com relação à precisão da classificação obtida. Basicamente, ela permite a análise de cada classificador gerado (por um indutor) como uma caixa preta. Mesmo assim, a obtenção dos dados necessários para essa comparação deve ser feita individualmente para cada indutor sendo avaliado.

Dessa forma, não são triviais e nem automáticas a análise e comparação dos classificadores simbólicos gerados, ou seja, através da verificação das regras (ou outra forma de linguagem de representação de conceitos) induzidas com relação à qualidade e interesse para o usuário final (Freitas, 1998a; Freitas, 1998b). Nesse contexto, um classificador simbólico é definido como todo e qualquer classificador que pode ser transformado num conjunto de regras do tipo *if* <condição> *then* <class = C_i >.

Este trabalho tem como objetivo formular uma proposta de unificação da linguagem de representação de conceitos para os algoritmos de AM simbólicos mais comuns presentes na $\mathcal{MLC}$++, bem como a sua implementação, através de uma biblioteca de *scripts* para a conversão da forma de representação do conhecimento induzido, gerado por esses indutores, para o formato padrão de regras $\mathcal{PBM}$— Prati, Baranauskas e Monard —, definido neste trabalho.

Quanto à organização deste relatório, na Seção 2 é dada uma introdução às formas de representação de conhecimento mais comuns em AM; na Seção 3, é feita uma descrição do formato padrão para a representação dos conceitos induzidos usado na conversão de classificadores definidos neste projeto. Na Seção 4 tem-se uma descrição, através de exemplos, da biblioteca de conversão desenvolvida e, finalmente, algumas considerações adicionais são apresentadas na Seção 5.

2 Formas de Representação do Conhecimento

Para se representar um conceito aprendido por um algoritmo de AM é necessária uma linguagem de representação para o conhecimento. De uma forma geral, as linguagens para esse tipo de representação se dividem em dois tipos: baseadas em atributos (proposicionais) e relacionais.

Os sistemas baseados em atributo são mais comuns na área de AM. Sistemas que utilizam linguagens de representação como regras de produção ou árvores de decisão podem ser tratados como variantes de linguagens proposicionais. Todos os algoritmos utilizados neste trabalho usam representação de conhecimento baseadas em atributo.

A forma de representação do conhecimento induzido geralmente depende da maneira pela qual o algoritmo faz indução. Algoritmos da família TDIDT — Top Down Induction of Decision Trees — como o $\mathcal{ID}3$ ou o $\mathcal{C}4.5$ geram como saída classificadores no formato de árvores de decisão. Já algoritmos que induzem regras de produção, como $\mathcal{CN}2$ ou $\mathcal{R}ipper$, diretamente, a partir dos conjuntos de dados, fornecem como saída classificadores na forma *if* <condição> *then* <class = C_i >. A Tabela 1 mostra a forma de representação do conhecimento induzido para os indutores utilizados neste trabalho.

Algoritmo	**Forma de Representação**
$\mathcal{C}4.5$	árvore de decisão
$\mathcal{C}4.5rules$	regras de produção
$\mathcal{C}5.0$	árvore de decisão e regras de produção
$\mathcal{ID}3$	árvore de decisão
$\mathcal{CN}2$	regras de produção
$\mathcal{OC}1$	árvore de decisão (oblíqua)
$\mathcal{R}ipper$	regras de produção (cláusulas prolog)
$\mathcal{MC}4$	árvore de decisão
$\mathcal{T}2$	árvore de decisão

Tabela 1: Formas de Representacao de Alguns Algoritmos de AM

As duas seções seguintes descrevem, de modo simplificado, árvores de decisão e regras de produção. Para isso, é conveniente salientar que um conjunto de dados T é um conjunto de n exemplos classificadas (rotuladas) e m atributos. Uma linha i se refere ao i-ésima exemplo ($i = 1, 2, \ldots, n$) e uma entrada x_{ij} se refere ao valor do j-ésima ($j = 1, 2, \ldots, m$) atributo X_j do exemplo i.

Os exemplos são tuplas $T_i = (x_{i1}, x_{i2}, \ldots, x_{im}, y_i) = (\mathbf{x}_i, y_i)$, também referenciadas como $(\mathbf{X}, Y)$ onde a última coluna, Y, é a que desejamos predizer com base nos outros $\mathbf{X}$ atributos, isto é, $Y = f(\mathbf{X})$. Cada $\mathbf{X}$ é um elemento do conjunto $X_1 \times X_2 \times \ldots \times X_m$ onde X_j é o domínio do j-ésimo atributo e Y pertence a uma das k classes, isto é, $Y \in \{C_1, C_2, \ldots, C_k\}$.

2.1 Árvores de Decisão

Uma Árvore de Decisão – DT – é uma estrutura de dados recursivamente definida como:

- um nó folha, que indica uma classe, ou

Instance No.	Outlook	Temperature	Humidity	Windy	Voyage?
T_1	sunny	25	72	yes	go
T_2	sunny	28	91	yes	dont_go
T_3	sunny	22	70	no	go
T_4	sunny	23	95	no	dont_go
T_5	sunny	30	85	no	dont_go
T_6	overcast	23	90	yes	go
T_7	overcast	29	78	no	go
T_8	overcast	19	65	yes	dont_go
T_9	overcast	26	75	no	go
T_{10}	overcast	20	87	yes	go
T_{11}	rain	22	95	no	go
T_{12}	rain	19	70	yes	dont_go
T_{13}	rain	23	80	yes	dont_go
T_{14}	rain	25	81	no	go
T_{15}	rain	21	80	no	go

Tabela 2: O Arquivo de Dados *voyage*

- um nó de decisão, que contém um teste sobre o valor de um atributo. Para cada um dos possíveis valores do atributo tem-se um ramo para uma outra árvore de decisão (subárvore). Cada subárvore contém a mesma estrutura de uma árvore.

Todos os exemplos mostrados neste trabalho estão baseados no conjunto de dados *voyage* (ver Tabela 2), adaptado de (Quinlan, 1988) por (Baranauskas and Monard, 2000b). Este conjunto de dados contém informações sobre medições diárias das condições meteorológicas e uma classificação se o dia é apropriado a uma visita à fazenda ("go") ou não ("dont_go"). Cada exemplo é composto pelos atributos:

- outlook: assume os valores *"sunny"*, *"overcast"* ou *"rain"*;
- temperature: um valor numérico que indica a temperatura em graus *Celsius*;
- humidity: outro valor numérico que indica a umidade relativa do ar;
- windy: assume os valores *"yes"* ou *"no"*.

Uma representação gráfica de uma DT gerada pelo indutor $\mathcal{C}4.5$ para o conjunto de dados *voyage* pode ser encontrada na Figura 1 na próxima página. Os círculos representam os nós de decisão, nas setas estão os possíveis resultados dos testes e os quadrados representam os nós folhas.

Uma DT pode ser usada para classificar novos exemplos, começando a partir da raíz da árvore e ir descendo pelos nós de decisão até encontrar um nó folha. Quando um nó folha é encontrado, a classe para o novo exemplo é prevista com o rótulo do nó folha.

2.2 Regras de Produção

Regras de produção geralmente são representadas por um conjunto de regras na forma

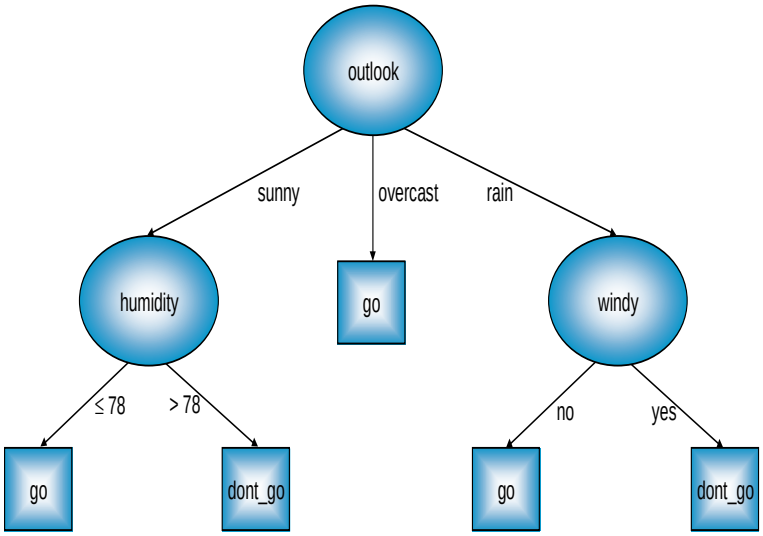

Figura 1: Representação Gráfica da Árvore de Decisão Gerada pelo Indutor $\mathcal{C}4.5$

$$\textit{if } \text{<condição>} \quad \textit{then } \text{<class} = C_i >$$

onde <condição> é uma disjunção de conjunções de testes para os atributos da forma

$$X_i \text{ op valor}$$

e C_i é um dos possíveis valores para a classe.

As regras de produção são divididas em ordenadas e não ordenadas. Em regras de produção ordenadas, a ordem de avaliação das regras é importante, ao contrário das regras de produção não ordenadas.

Um exemplo de regras de produção para a mesma hipótese induzida representada pela DT na Figura 1 pode ser vista na Figura 2 na próxima página.

3 O Formato Padrão $\mathcal{PBM}$

Um formato específico de regras de produção — o formato $\mathcal{PBM}$ — foi adotado neste trabalho para o qual são convertidos os classificadores gerados pelos indutores que fazem parte da biblioteca de conversão desenvolvida, descrita com mais detalhes na Seção 4 na página 6.

O formato padrão $\mathcal{PBM}$ adotado para o desenvolvimento da blibliotaca de conversão baseia-se no seguinte formato de regras:

$$\textit{if } \text{<condição>} \quad \textit{then } \text{<class} = C_i\text{>}$$

e é definido formalmente pela gramática $\mathcal{G} = < \vartheta_1, \Sigma_1, \delta_1, \varsigma >$, onde:

```
IF outlook = overcast
THEN CLASS = go

IF outlook = sunny
        AND humidity <= 78
THEN CLASS = go

IF outlook = sunny
        AND humidity > 78
THEN CLASS = dont_go

IF outlook = rain
        AND windy = yes
THEN CLASS = dont_go

IF outlook = rain
        AND windy = no
THEN CLASS = go
```

Figura 2: Exemplo de Regras de Produção

- ϑ_1 é o vocabulário dos símbolos não terminais: <regra>, <condição>, <classificação>, <fator>, <termo>, <comparação>, <combinação linear> e <somas>, <operador>, <atributo>, <atributo numérico> e <valor>.
- $\mathbf{\Sigma_1}$ é o vocabulário dos símbols terminais: *if*, *then*, *class*, *and*, *or*, $X_1, X_2, ..., X_m, Y, \leq, \geq, \in, <, >, \neq$ e $=$.
- δ_1 é o conjunto das leis de formação da gramátida $\mathcal{G}$, como mostra a Tabela 3.
- ς é o símbolo inicial da gramática: <regra>.

$\delta_1 =$	<regra>	$\Rightarrow$	`IF` <condição> `THEN` <classificação>
	<condição>	$\Rightarrow$	<fator> |
			<fator> `OR` <condição>
	<fator>	$\Rightarrow$	<termo> |
			<termo> `AND` <fator>
	<termo>	$\Rightarrow$	<comparação> |
			<combinação linear>
	<comparação>	$\Rightarrow$	<atributo> <operador> <valor>
	<combinação linear>	$\Rightarrow$	<somas> <operador> <valor>
	<somas>	$\Rightarrow$	<constante> $\times$ <atributo numérico> |
			<constante> $\times$ <atributo numérico> $+$ <somas>
	<operador>	$\Rightarrow$	$\leq, \geq, \in, =, <, >, \neq$
	<classificação>	$\Rightarrow$	`CLASS` = <valor>

Tabela 3: Conjunto de Transições da Gramática $\mathcal{G}$

Dessa forma, o classificador gerado pelo indutor, depois de transformado no formato padrão, se resume a um conjunto de regras *if* <condição> *then* <class = C_i>. O arquivo de saída que contém o resultado da conversão apresenta essas regras, enumerando-as (Figura 3 na página seguinte). Deve se notar que a ordem original das regras extraídas é mantida bem como a ordem dos testes de atributos em cada regra.

```
Standart Rules Conversor        Copyright (c) Ronaldo C. Prati
Inducer: <Inducer Name>         Input File: <Input File Name>
Date: <Date>

R0001 IF ...
    THEN ...

R0002 IF ...
    THEN ...

...
```

Figura 3: Formato do Arquivo de Saída Padrão

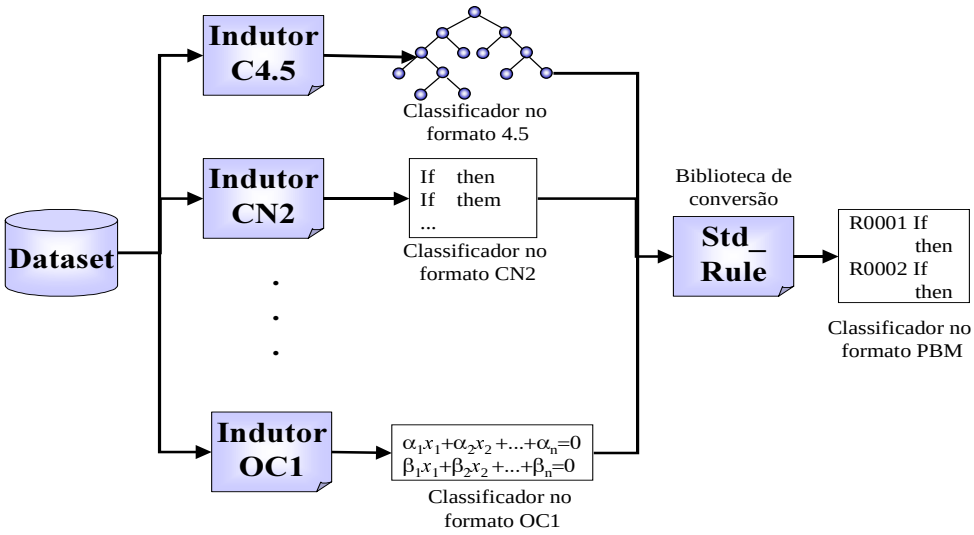

Figura 4: A biblioteca de conversão

4 A Biblioteca de Conversão

A biblioteca de conversão para a implementação do formato padrão foi desenvolvida utilizando-se a linguagem PERL (PERL, 1999). Ela trabalha a partir dos arquivos de saída gerados pelos indutores. A saída, já no formato padrão, é armazenada em outro arquivo ou apresentada na tela. Como descrito anteriormente, a biblioteca converte para o formato padrão nove algoritmos de AM — $\mathcal{ID}3$, $\mathcal{C}4.5$, $\mathcal{C}4.5rules$, $\mathcal{CN}2$, $\mathcal{OC}1$, $\mathcal{R}ipper$, $\mathcal{C}5.0$ /See5, $\mathcal{T}2$ e $\mathcal{MC}4$ —. Uma visão geral da biblioteca está representada na Figura 4.

Há, ainda, uma opção para se habilitar, junto ao arquivo de saída (já no formato padrão) algumas informações adicionas que são geradas por alguns indutores, tais como número de exemplos cobertos corretamente, número de exemplos cobertos incorretamente, etc.

Para facilitar a utilização da biblioteca de conversão pelo usuário foi criado um *script* adicional com o objetivo de atuar como interface para a chamada dos *scripts* da biblioteca, através da linha de comando do sistema operacional, ocultando do usuário detalhes da programação

PERL. Esse *script* chama a biblioteca de conversão com os parâmetros corretos, a partir dos parâmetros passados pela linha de comando. A utilização desse *script* se dá pelo comando:

std_rule.pl <indutor> [-f<arquivo de entrada>] [-s<arquivo de saída>] [-i]

onde:

<**indutor**> é o nome de um dos indutores presentes na biblioteca (`c4.5, c4.5rules, c5, id3, cn2, oc1, t2, mc4` ou `ripper`).

-f<arquivo de entrada> é o nome do arquivo de contém o classificador gerado pelo indutor, isto é, o nome do arquivo de saída do indutor, e que é usado como entrada para a biblioteca de conversão.

-s<arquivo de saída> é o nome do arquivo no qual será gravado o classificador no formato padrão, isto é, o nome do arquivo que conterá a saída da biblioteca de conversão.

-i habilita, junto ao arquivo de saída, reter as informações adicionais do classificador original (quando este as tiver).

Os parâmetros entre colchetes –[ ]– são opcionais e os comandos [**-f<arquivo de entrada>**] e [**-s<arquivo de saída>**] podem ser substituídos por "*pipes*" para facilitar a integração com outros sistemas. Se o nome do arquivo de entrada não for passado como parâmetro ele deve ser susbstituído por um *pipe* de entrada; já, se o nome do arquivo de saída não for passado como parâmetro, a saída padrão (geralmente o monitor de vídeo) é utilizada.

Por exemplo, o comando

std_rule.pl c4.5 -i <voyage.c45.tree

chama a biblioteca de conversão para o indutor $\mathcal{C}4.5$, habilitando a saída as informações adicionais geradas pelo $\mathcal{C}4.5$, tendo como entrada o arquivo voyage.c45.tree e a saída, no formato padrão, é apresentada na saída padrão.

No restante desta seção será mostrada a conversão dos classificadores gerados por vários indutores, para o formato padrão, utilizando a biblioteca de conversão desenvolvida nesse projeto, a partir de exemplos, baseados no conjunto de dados *voyage*(Tabela 2 na página 3)

4.1 $\mathcal{C}4.5$

$\mathcal{C}4.5$ foi desenvolvido por J. Ross Quinlan (Quinlan, 1988). Este algoritmo de AM é membro da família TDIDT – *Top Down Induction of Decision Tree* –, isto é, $\mathcal{C}4.5$ cria árvores de decisão para representar regras de classificação.

O $\mathcal{C}4.5$ é um dos mais famosos indutores que usam árvore de indução. Em geral, seu código é robusto e pode ser executado em várias plataformas. Pode ser obtido em:

`http://www.sce.unsw.edu.au/~quinlan`.

```
C4.5 [release 8] decision tree generator        Thu Mar  2 15:30:20 2000
----------------------------------------

    Options:
        File stem <voyage>
Read 15 cases (4 attributes) from voyage.data
Decision Tree:

outlook = overcast: go (5.0/1.0)
outlook = sunny:
|   humidity <= 78 : go (2.0)
|   humidity > 78 : dont_go (3.0)
outlook = rain:
|   windy = yes: dont_go (2.0)
|   windy = no: go (3.0)

Tree saved
Evaluation on training data (15 items):
         Before Pruning           After Pruning
        ----------------   ---------------------------
        Size      Errors   Size      Errors   Estimate
           8    1( 6.7%)      8    1( 6.7%)    (43.3%)   <<
```

Figura 5: Árvore de Decisão Gerada pelo Indutor $\mathcal{C}4.5$

A Figura 5 mostra o classificador gerado pelo indutor $\mathcal{C}4.5$ no conjunto de dados *voyage*. A seqüencia de caracteres – *string* – antes do sinal de : (dois pontos) representa um nó de decisão. A *string* após o sinal (se existir) representa um nó folha; se essa *string* não existir, tem-se uma subárvore na próxima linha. O sinal de | (barra) representa a profundidade do nó na árvore. Os dois números entre parênteses representam, respectivamente, o número de exemplos cobertos pelo ramo e o número de exemplos onde a classe prevista não é a real (erro).

A transformação do formato de saída gerado pelo $\mathcal{C}4.5$ para o formato padrão se dá percorrendo a árvore de decisão. A profundidade atual da árvore é mantida em uma pilha (empilha-se o nó atual ao se aprodundar na árvore e desempilha-se nós ao retroceder). O Algoritmo 1 implementa essa idéia. A listagem do arquivo contendo as regras no formato padrão pode ser vista na Figura 6 na próxima página. A conversão foi realizada utilizando a linha de comando:

std_rule.pl c4.5 -fvoyage.c45.tree -svoyage.c45.std

Como mencionado anteriormente, é possível manter as informações adicionais geradas pelo indutor. Por exemplo, o formato padrão para o $\mathcal{C}4.5$ com informações adicionais pode ser visto na Figura 7 na página 10 (números ao final de cada regra). Essa saída foi obtida chamando a biblioteca de conversão com os seguintes parametros:

std_rule.pl c4.5 -fvoyage.tree -svoyage.std -i

Esta funcionalidade pode ser utilizada para todos os indutores que tenham informações adicionais que a biblioteca reconhece. Por questões de espaço, os demais exemplos mostram apenas o classificador sem as informações adicionais.

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: C4.5                          Input File: voyage.c45.tree
Date: Thu Mar  2 15:35:43 2000

R0001   IF outlook = overcast
        THEN CLASS = go

R0002   IF outlook = sunny
                AND humidity <= 78
        THEN CLASS = go

R0003   IF outlook = sunny
                AND humidity > 78
        THEN CLASS = dont_go

R0004   IF outlook = rain
                AND windy = yes
        THEN CLASS = dont_go

R0005   IF outlook = rain
                AND windy = no
        THEN CLASS = go
```

Figura 6: Classificador Gerado pelo $\mathcal{C}4.5$ no Formato Padrão

Algoritmo 1 $\mathcal{C}4.5$ to $\mathcal{PBM}$ standard rule

Require: File containing $\mathcal{C}4.5$ tree classifier

```
 1: procedure c45_to_stdrule
 2: for all nodes in tree do
 3:    this_node_level ← count( '|') in node
 4:    while this_node_level >= tree_level do
 5:       go up one level on tree {delete a node on stack}
 6:    end while
 7:    go down one level on tree {store a node on stack}
 8:    if node has a leaf then
 9:       print rule {if stack then class = leaf}
10:    end if
11: end for
12: end c45_to_stdrule
```

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: C4.5                          Input File: voyage.c45.tree
Date: Thu Mar  2 15:35:43 2000

R0001   IF outlook = overcast
        THEN CLASS = go (5.0/1.0)

R0002   IF outlook = sunny
                AND humidity <= 78
        THEN CLASS = go (2.0)

R0003   IF outlook = sunny
                AND humidity > 78
        THEN CLASS = dont_go (3.0)

R0004   IF outlook = rain
                AND windy = yes
        THEN CLASS = dont_go (2.0)

R0005   IF outlook = rain
                AND windy = no
        THEN CLASS = go (3.0)
```

Figura 7: Classificador Gerado pelo $\mathcal{C}4.5$ no Formato Padrão com as Informações Adicionais

4.2 $\mathcal{C}4.5rules$

$\mathcal{C}4.5rules$ é um algoritmo de AM que cria regras de produção extraídas de árvores de decisão geradas pelo $\mathcal{C}4.5$ (Quinlan, 1988). Desta forma, para utilizar o $\mathcal{C}4.5rules$ é necessário executar o $\mathcal{C}4.5$ antes.

No $\mathcal{C}4.5rules$, cada regra assume a forma:

$$<\text{condição}> \rightarrow <\text{classe} = C_i > [A\%]$$

que pode ser entendida como *if* <condição> *then* <class = C_i>, com uma precisão de $A\%$. O Algoritmo 2 implementa a idéia da conversão do formato de saída do $\mathcal{C}4.5rules$ para o formato padrão $\mathcal{PBM}$.

Algoritmo 2 $\mathcal{C}4.5rules$ to $\mathcal{PBM}$ standad rule

Require: File containing $\mathcal{C}4.5rules$ rule set classifier

```
1: for all rules infile do
2:    if is a default rule then
3:       print default rule {DEFAULT CLASS = C_i}
4:    else
5:       search for body's part
6:       search for tail's part {-> part}
7:       print rule {if body then tail}
8:    end if
9: end for
```

```
C4.5 [release 8] rule generator Sat Oct 14 14:12:32 2000
-------------------------------
    Options:
        File stem <../Data/voyage>
Read 15 cases (4 attributes) from ../Data/voyage
------------------
Processing tree 0
Final rules from tree 0:

Rule 3:
        outlook = rain
        windy = no
        ->  class go  [63.0%]
Rule 1:
        outlook = sunny
        humidity > 78
        ->  class dont_go  [63.0%]
Rule 2:
        outlook = rain
        windy = yes
        ->  class dont_go  [50.0%]
Default class: go

Evaluation on training data (15 items):
Rule  Size  Error  Used  Wrong            Advantage
----  ----  -----  ----  -----            ---------
   3     2  37.0%     3      0 (0.0%)        0 (0|0)    go
   1     2  37.0%     3      0 (0.0%)        3 (3|0)    dont_go
   2     2  50.0%     2      0 (0.0%)        2 (2|0)    dont_go
Tested 15, errors 1 (6.7%)   <<
          (a)  (b)      <-classified as
          ---- ----
            9           (a): class go
            1    5      (b): class dont_go
```

Figura 8: Regras de Produção Geradas pelo Indutor $\mathcal{C}4.5rules$

A Figura 8 mostra o classificador gerado pelo $\mathcal{C}4.5rules$ e a Figura 9 na página seguinte mostra o classificador reduzido ao formato padrão. A linha de comando abaixo foi utilizada para a conversão dos arquivos:

std_rule.pl c4.5rules -fvoyage.rules -svoyage.rules.std

4.3 $\mathcal{ID}3$

O $\mathcal{ID}3$ é um algoritmo de árvore de decisão básico sem poda. A versão do $\mathcal{ID}3$ utilizada neste trabalho é a que vem agregada à $\mathcal{MLC}$++ (Kohavi et al., 1994).

Um exemplo da árvore de indução gerada pelo $\mathcal{ID}3$ pode ser vista na Figura 10 na página 13. É interessante notar a maior complexidade para a leitura (humana) deste tipo de linguagem de

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: C4.5                          Input File: voyage.c45.tree
Date: Thu Mar  2 15:35:43 2000

R0001   IF outlook = overcast
        THEN CLASS = go

R0002   IF outlook = sunny
                AND humidity <= 78
        THEN CLASS = go

R0003   IF outlook = sunny
                AND humidity > 78
        THEN CLASS = dont_go

R0004   IF outlook = rain
                AND windy = yes
        THEN CLASS = dont_go

R0005   IF outlook = rain
                AND windy = no
        THEN CLASS = go
```

Figura 9: Classificador Gerado pelo $\mathcal{C}4.5$*rules* no Formato Padrão

representação, a qual é bem facilitada no formato padrão.

Os nós são rotulados por números entre colchetes; em seguida vem a descrição do nó entre parênteses: a *string* que aparece antes da vírgula é o atributo de teste do nó, a profundidade na árvore é dada pelo valor de *level*. A *string* que vem após o sinal de : representa os ramos da árvore; seu formato é um conjunto de ramos do tipo [`i`]`--(teste)-->`[`j`] (a partir do nó `i`, aplique `teste` para o atributo e se for verdade, vá para `j`); se essa *string* não existir, o nó é um nó folha.

A transformação do formato gerado pelo $\mathcal{ID}3$ para o formato padrão se dá percorrendo a árvore de decisão gerada pelo mesmo, utilizando-se um algoritmo recursivo. O Algoritmo 3 implementa essa idéia. Na Figura 11 na página 14 é mostrado o formato padrão do exemplo da Figura 10 na próxima página. A transformação para o formato padrão foi feita pela linha de comando:

std_rule.pl id3 -fvoyage.id3 -svoyage.id3.std

4.4 $\mathcal{S}$ee5 e $\mathcal{C}5.0$

$\mathcal{C}5.0$ é a versão comercial do $\mathcal{C}4.5$. Na verdade, exitem dois programas: $\mathcal{C}5.0$ para plataformas Unix e $\mathcal{S}$ee5, que contém as mesmas funcionalidades que o $\mathcal{C}5.0$, mas utiliza uma interface gráfica para a plataforma Windows™. Maiores informações sobre o $\mathcal{C}5.0$ e o $\mathcal{S}$ee5 podem ser encontradas no endereço:

`http://www.rulequest.com`.

Algoritmo 3 $\mathcal{ID}3$ to $\mathcal{PBM}$ standad rule

Require: A file with $\mathcal{ID}3$ tree

```
1: procedure id3_to_stdrule
2: root_node ← 0
3: rule = {}
4: for all branchs in root_node do
5:    node_label, node_test, node_goal ← [label]–(test)–>[goal]
6:    node_branch ← node_label node_test
7:    search(rule + node_branch, node_goal)
8: end for
9: end id3_to_stdrule
10: procedure search(rule, goal)
11: find node [goal] on tree
12: if node goal is a leaf then
13:    print rule {if rule then class = leaf}
14: else
15:    for all branchs in goal_node do
16:       node_label, node_test, node_goal ← [label]–(test)–>[goal]
17:       node_branch ← node_label node_test
18:       search(rule + node_branch, node_goal)
19:    end for
20: end if
21: end search
```

```
Rooted Decision Graph Categorizer ID3
Root: temperature

[0](temperature, level 0) : [0]--(?)-->[1][0]--(<= 19.5)-->[2]
[0]--(> 19.5)-->[3]
[1](go, level 1) :
[2](dont_go, level 1) :
[3](temperature, level 1) : [3]--(?)-->[4][3]--(<= 22.5)-->[5]
[3]--(> 22.5)-->[6]
[4](go, level 2) :
[5](go, level 2) :
[6](humidity, level 2) : [6]--(?)-->[7][6]--(<= 79)-->[8][6]--(> 79)-->[9]
[7](go, level 3) :
[8](go, level 3) :
[9](outlook, level 3) : [9]--(?)-->[10][9]--(sunny)-->[11]
[9]--(overcast)-->[12][9]--(rain)-->[13]
[10](dont_go, level 4) :
[11](dont_go, level 4) :
[12](go, level 4) :
[13](temperature, level 4) : [13]--(?)-->[14][13]--(<= 24)-->[15]
[13]--(> 24)-->[16]
[14](dont_go, level 5) :
[15](dont_go, level 5) :
[16](go, level 5) :
```

Figura 10: Árvore de Decisão Gerado pelo Indutor $\mathcal{ID}3$

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: ID3                    Input File: voyage.id3
Date: Thu Mar  2 16:03:29 2000

R0001   IF temperature = ?
        THEN CLASS = go

R0002   IF temperature  <= 19.5
        THEN CLASS = dont_go

R0003   IF temperature > 19.5
                AND temperature = ?
        THEN CLASS = go

R0004   IF temperature > 19.5
                AND temperature <= 22.5
        THEN CLASS = go

R0005   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity = ?
        THEN CLASS = go

R0006   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity <= 79
        THEN CLASS = go

R0007   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity > 79
                AND outlook = ?
        THEN CLASS = dont_go

R0008   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity > 79
                AND outlook = sunny
        THEN CLASS = dont_go

R0009   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity > 79
                AND outlook = overcast
        THEN CLASS = go

R0010   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity > 79
                AND outlook = rain
                AND temperature = ?
        THEN CLASS = dont_go

R0011   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity > 79
                AND outlook = rain
                AND temperature <= 24
        THEN CLASS = dont_go

R0012   IF temperature > 19.5
                AND temperature > 22.5
                AND humidity > 79
                AND outlook = rain
                AND temperature > 24
        THEN CLASS = go
```

Figura 11: Classificador Gerado pelo $\mathcal{ID}3$ no Formato Padrão

```
See5 [Release 1.11]     Thu Mar 02 15:52:53 2000
-------------------
Read 15 cases (4 attributes) from voyage.data
Decision tree:

outlook = overcast: go (5/1)
outlook = sunny:
:...humidity <= 78: go (2)
:   humidity > 78: dont_go (3)
outlook = rain:
:...windy = yes: dont_go (2)
    windy = no: go (3)

Evaluation on training data (15 cases):
            Decision Tree
          ----------------
          Size      Errors
             5    1( 6.7%)   <<
           (a)   (b)    <-classified as
          ----  ----
             9          (a): class go
             1     5    (b): class dont_go
Time: 0.1 secs
```

Figura 12: Árvore de Decisão Gerado pelo Indutor $\mathcal{C}5.0$

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: C5.0                   Input File: voyage.c50
Date: Thu Mar  2 15:57:31 2000

R0001   IF outlook = overcast
        THEN CLASS = go

R0002   IF outlook = sunny
                AND humidity <= 78
        THEN CLASS = go

R0003   IF outlook = sunny
                AND humidity > 78
        THEN CLASS = dont_go

R0004   IF outlook = rain
                AND windy = yes
        THEN CLASS = dont_go

R0005   IF outlook = rain
                AND windy = no
        THEN CLASS = go
```

Figura 13: Classificador Gerado pelo $\mathcal{C}5.0$ no Formato Padrão

A saída gerada pelo indutor pode ser encontrada na Figura 12 na página anterior. Como no $\mathcal{C}4.5$, a *string* que antecede o sinal de : representa um nó de decisão. A *string* após o sinal representa um nó folha (se existir). O sinal de :... indica um novo ramo na árvore de decisão o nível de paragrafação indica a profundidade na árvore.

O classificador no formato padrão pode ser visto na Figura 13 na página precedente. Como pode ser observado, o processo de transformação do formato gerado pelo $\mathcal{C}5.0$ para o formato padrão é semelhante ao do $\mathcal{C}4.5$, apesar de uma pequena diferença sintática. O comando utilizado para a conversão foi:

std_rule.pl c5.0 -fvoyage.c50 -svoyage.c50.std

4.5 $\mathcal{CN}2$

O $\mathcal{CN}2$ é um indutor externo que faz interface com a $\mathcal{MLC}$++; está descrito em (Clark and Boswell, 1989; Clark and Boswell, 1991). O $\mathcal{CN}2$ é um algoritmo de indução de regras. É um algoritmo não-incremental que recebe como entrada um conjunto de exemplos e gera como saída um conjunto de regras de produção para classificar esses exemplos, além de permitir avaliar a exatidão desse conjunto. Esse algoritmo foi projetado para uma eficiente geranção de regras de indução do tido *if* <condição> *then* <class = C_i > simples e compreensíveis em domínios onde ruídos nos dados de entrada podem estar presentes. O $\mathcal{CN}2$ pode ser obtido em

`http://www.cs.utexas.edu/users/pclark/software.html`

ou contactar `pclark@cs.utexas.edu`, para maiores informações sobre o $\mathcal{CN}2$.

A Figura 14 na página seguinte mostra o exemplo do conjunto de regras gerado pelo $\mathcal{CN}2$ para o conjunto de dados *voyage*. Os números entre colchetes representam o número de exemplos cobertos pela regra, divididos em um vetor cuja posição corresponde à classe real do exemplo. Ressalta-se que o formato padrão adotado para este trabalho foi baseado no formato das regras geradas pelo $\mathcal{CN}2$. O Algoritmo 4 implementa a idéia da conversão. A Figura 15 na página 18 mostra o conjunto de regras do $\mathcal{CN}2$ transformadas para o formato padrão. O comando usado na transformação foi:

std_rule.pl cn2 -fvoyage.rul -svoyage.cn2.std

Algoritmo 4 $\mathcal{CN}2$ to $\mathcal{PBM}$ standad rule

Require: File containing $\mathcal{CN}2$ rule set classifier

```
1: for all rules in file do
2:    if is a default rule then
3:       print default rule {DEFAULT CLASS = C_i}
4:    else
5:       search for if's part
6:       search for then's part
7:       print rule {if ... then ...}
8:    end if
9: end for
```

```
**RULE FILE**
@
Time: [ Sat Mar 11 15:57:31 2000 ]
Examples: voyage.exs
Algorithm: UNORDERED
Error_Estimate: LAPLACIAN
Threshold: 0.00
Star: 5
@
*UNORDERED-RULE-LIST*

IF    humidity < 83.00
  AND windy = no
THEN  voyage = go  [5 0]

IF    outlook = overcast
  AND temperature > 19.50
THEN  voyage = go  [4 0]

IF    outlook = sunny
  AND humidity < 76.00
THEN  voyage = go  [2 0]

IF    outlook = rain
  AND humidity > 87.50
THEN  voyage = go  [1 0]

IF    outlook = sunny
  AND humidity > 83.00
THEN  voyage = dont_go  [0 3]

IF    temperature < 24.00
  AND humidity < 80.50
  AND windy = yes
THEN  voyage = dont_go  [0 3]

(DEFAULT) voyage = go  [9 6]
```

Figura 14: Classificador Gerado pelo Indutor $\mathcal{CN}2$

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: CN2                        Input File: voyage.rul
Date: Sat Mar 11 18:47:20 2000

R0001   IF humidity < 83.00
                AND windy = no
        THEN CLASS = go

R0002   IF outlook = overcast
                AND temperature > 19.50
        THEN CLASS = go

R0003   IF outlook = sunny
                AND humidity < 76.00
        THEN CLASS = go

R0004   IF outlook = rain
                AND humidity > 87.50
        THEN CLASS = go

R0005   IF outlook = sunny
                AND humidity > 83.00
        THEN CLASS = dont_go

R0006   IF temperature < 24.00
                AND humidity < 80.50
                AND windy = yes
        THEN CLASS = dont_go

R0007   DEFAULT CLASS = go
```

Figura 15: Classificador Gerado pelo $\mathcal{CN}2$ no Formato Padrão

4.6 $\mathcal{R}ipper$

O algoritmo $\mathcal{R}ipper$ — *Repeated Incremental Pruning to Produce Error Reduction* — é um sistema para induzir regras a partir de um conjunto de instâncias (Cohen, 1995). Um ponto interessanto sobre $\mathcal{R}ipper$ é que ele permite que o usuário incorpore algum conhecimento já existente ao processo de indução, na tentativa de tornar o processo mais preciso.

As regras induzidas pelo $\mathcal{R}ipper$ assumem a notação de cláusulas Prolog:

$$C_i = X_r\ op_r\ Valor_r,\ X_s\ op_s\ Valor_s,\ ...\ X_z\ op_z\ Valor_z\ (\text{P/N})$$

onde X_i é o atributo, op é um operador pertencente ao conjunto $\{=, \neq, <, \leq, >, \geq\}$ e $Valor$ é um valor válido para o atributo X_i. As vírgulas unem cada teste como um `and` lógico. Os valores entre parênteses P e N indicam, respectivamente, o número de instâncias cobertas pela regra que satifazem e não satisfazem a condição C_i. Maiores informações sobre $\mathcal{R}ipper$ podem ser obtidas no endereço

`http://www.research.att.com/~wcohen/ripperd.html`

O Algoritmo 5 implementa a idéia da conversão do formato de saída do $\mathcal{R}ipper$ para o formato $\mathcal{PBM}$.A Figura 16 na página seguinte mostra o classificador gerado pelo indutor $\mathcal{R}ipper$ e a Figura 17 na próxima página mostra o classificador transformado para o formato padrão. Para a transformação foi usada a linha de comando:

std_rule.pl ripper -fvoyage.ripper -svoyage.ripper.std

Algoritmo 5 $\mathcal{R}ipper$ to $\mathcal{PBM}$ standad rule

Require: File containing $\mathcal{R}ipper$ rule set classifier
1: **for all** rules in file **do**
2: print default rule {DEFAULT CLASS = C_i}
3: **if** is a default rule **then**
4: print default rule {DEFAULT CLASS = C_i}
5: **else**
6: (tail, body) ← tail :- body
7: print rule {if body then tail}
8: **end if**
9: **end for**

4.7 $\mathcal{OC}1$

Oblique Classifier 1 — $\mathcal{OC}1$— é um sistema de indução de árvores de decisão projetado para aplicações que tenham seus atributos como valores numéricos contínuos (Murthy et al., 1994). O $\mathcal{OC}1$ constrói árvores de decisão que contenham combinação linear de um ou mais atributos para cada nó. Essas árvores particionam o espaço em planos oblíquos e não paralelos aos eixos. $\mathcal{OC}1$ têm sido usado para classificação de dados em diversos domínios, incluindo astronomia (Salzberg et al., 1995) e seqüenciamento de DNA (Salzberg, 1995).

Árvores induzidas pelo $\mathcal{OC}1$ assumen a forma:

```
option: data is clean
Final hypothesis is:
dont_go :- humidity>=85, outlook=sunny (3/0).
dont_go :- windy=yes, outlook=rain (2/0).
dont_go :- temperature<=20, temperature>=20 (1/0).
default go (9/0).
================================ summary ================================
Train erros rate:  0.00% +/- 0.00% (15 datapoints)     <<
Hypothesis size:   3 rules, 9 conditions
Learning time:     0.01 sec
```

Figura 16: Classificador Gerado pelo Indutor $\mathcal{R}ipper$

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: ripper                    Input File: voyage.ripper
Date: Mon Jul 10 19:28:56 2000


R0001   IF humidity>=85
           AND outlook=sunny
        THEN CLASS = dont_go

R0002   IF windy=yes
           AND outlook=rain
        THEN CLASS = dont_go

R0003   IF temperature<=20
           AND temperature>=20
        THEN CLASS = dont_go

R0004   DEFAULT CLASS =  go
```

Figura 17: Classificador Gerado pelo $\mathcal{R}ipper$ no Formato Padrão

$$a_1x[1] + a_2x[2] + ... + a_mx[m] + a_{m+1} = 0$$

onde a_1 é uma constante e x_i é a i-ésima feature.

Como o $\mathcal{OC}1$ só trabalha com dados numéricos, os valores discretos presentes no conjunto de dados *voyage*, exemplificado na Tabela 2 na página 3 devem serem convertidos para valores numéricos. Uma descrição mais detalhada de como converter atributos discretos para numéricos podem ser encontrada em (Weiss and Indurkhya, 1998). A classe "`go`" correspode à nova classe `1` e a classe "`dont_go`" à nova classe `2`.

As Figuras 18 e 19 mostram, respectivamente, as árvores de decisão geradas pelo $\mathcal{OC}1$ com podas e sem podas. Como podemos observar, o $\mathcal{OC}1$ traça hiperplanos para a classificação. Os valores maiores que 0 (à esquerda) e menores que 0 (à direita) fazem parte de outros semi-espaços. A divisão dos semi-espaços em novos semi-espaços (à direta: r, rr, rl, ... e à esquerda: l, lr, lf, ...), com novos semi-planos, continua até que o respectivo semi-espaço só contenha valores de uma classe (ou um numero n $< \alpha$, para um α pequeno, de exemplos de outras classes). Esse semi-espaço corresponde, então, à classe dos exemplos correspondentes ao semi-espaço. O Algoritmo 6 implementa a idéia da conversão do formato de saídado do $\mathcal{OC}1$ para o formato $\mathcal{PBM}$.

Algoritmo 6 $\mathcal{OC}1$ to $\mathcal{PBM}$ standad rule

Require: File containing $\mathcal{OC}1$ tree classifier

```
1: procedure oc1_to_stdrule
2: search for root hyperplane
3: rule ← hyperplane equation
4: find(rule, l) {left subhyperplane}
5: find(rule, r) {right subhyperplane}
6: end oc1_to_stdrule
7: procedure find(rule, hyperplane)
8: search for hyperplane
9: if not found hyperplane then
10:    print rule
11: else
12:    rule ← rule + hyperplane equation
13:    find(rule, hyperplane + l) {left subhyperplane}
14:    find(rule, hyperplane + r) {right subhyperplane}
15: end if
16: end find
```

Os classificadores transformados no formato padrão são mostrados nas Figuras 20 e 21 para a DT podada e não podada, respectivamente. Os comandos utilizados na transformação, para a árvore podada e não podada foram, respectivamente:

std_rule.pl oc1 -foc1.tree -svoyage.oc1.std
std_rule.pl oc1 -foc1.tree.unpruned -svoyage.oc1.un.std

4.8 $\mathcal{MC}4$ e $\mathcal{T}2$

O $\mathcal{MC}4$ é um dos sucessores do $\mathcal{ID}3$. Ele inclui uma poda semelhante ao $\mathcal{C}4.5$. O $\mathcal{MC}4$ apresenta, normalmente, resultados semelhantes ao $\mathcal{C}4.5$, com a exceção da manipulação de

```
Training set: ../voyage.data, Dimensions: 6, Categories: 2


Root Hyperplane: Left = [0,2], Right = [9,3]
1.000000 x[4] + -19.500000 = 0
```

Figura 18: Árvore de Decisão "Podada" Gerada pelo Indutor $\mathcal{OC}1$

```
Training set: ../voyage.data, Dimensions: 6, Categories: 2


Root Hyperplane: Left = [0,2], Right = [9,3]
1.000000 x[4] + -19.500000 = 0

r Hyperplane: Left = [7,0], Right = [2,3]
1.000000 x[1] + -0.500000 = 0

rr Hyperplane: Left = [2,0], Right = [0,3]
1.000000 x[5] + -78.500000 = 0
```

Figura 19: Árvore de Decisão "Sem Podas" Gerada pelo Indutor $\mathcal{OC}1$

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: OC1                        Input File: voyage.dt
Date: Sat Ago  19 16:05:03 2000

R0001   IF 1.000000 x[4] + -19.500000  < 0
        THEN CLASS = 2

R0002   IF 1.000000 x[4] + -19.500000  > 0
        THEN CLASS = 1
```

Figura 20: Classificador Gerado pelo $\mathcal{OC}1$ a Partir da DT "Podada"

```
Standard Rules Conversor v1.1.0 Copyright (c) Ronaldo C. Prati
Inducer: OC1                        Input File: voyage.dt.unpruned
Date: Sat Ago  19 16:10:53 2000

R0001   IF 1.000000 x[4] + -19.500000  < 0
        THEN CLASS = 2

R0002   IF 1.000000 x[4] + -19.500000  > 0
                AND 1.000000 x[1] + -0.500000  < 0
        THEN CLASS = 1

R0003   IF 1.000000 x[4] + -19.500000  > 0
                AND 1.000000 x[1] + -0.500000  > 0
                AND 1.000000 x[5] + -78.500000  < 0
        THEN CLASS = 1

R0004   IF 1.000000 x[4] + -19.500000  > 0
                AND 1.000000 x[1] + -0.500000  > 0
                AND 1.000000 x[5] + -78.500000  > 0
        THEN CLASS = 2
```

Figura 21: Classificador Gerado pelo $\mathcal{OC}1$ a Partir da DT "Sem Podas"

valores desconhecidos, que é diferente.

O $\mathcal{T}2$ é um algoritmo de árvore de decisão de dois níveis que minimiza o número de erros e discretiza atributos contínuos (Auer et al., 1995). Requer grande quantidade de memória quando se tem muitas classes.

Ambos algoritmos, $\mathcal{MC}4$ e $\mathcal{T}2$, têm suas saídas semelhantes ao $\mathcal{C}4.5$ (Seção 4.1 na página 7) e $\mathcal{ID}3$ (Seção 4.3 na página 11), respectivamente.

5 Considerações Finais

Este trabalho traz uma proposta de unificação da linguagem de representação do conceito induzido para alguns dos algoritmos de AM simbólicos presentes na $\mathcal{MLC}$++, e a implementação de uma biblioteca de *scripts* para a conversão automática dos classificadores gerados por esses indutores para o formato padrão aqui proposto.

Como resultados futuros deste trabalho, espera-se que outras pesquisas sejam beneficiadas ao poder comparar os classificadores "por dentro", ou seja, com relação à qualidade do classificador sob diversas considerações e não apenas através da precisão obtida num conjunto de teste.

Uma das propostas de continuidade deste trabalho é o de, dado um conjunto de dados, avaliar cada regra individualmente, obtendo medidas padrões, tais como o número do exemplos cobertos correta e incorretamente pela regra (como é possível notar, cada classificador exibe informações sobre precisão das regras, mas cada um com a sua própria medida). Com isso, a comparação entre diversos classificadores torna-se-á facilitada.

A Codigos Fonte

A.1 std_rule.pl

```
#!/usr/bin/perl

#-----------------------------------------------------------------------
# This script calls the standard rules conversor script
#
# pre-condition
#      A file contend a classifier induced for id3, mc4, oc1, c4.5,
#      c4.5rules, c5.0, cn2, ripper or t2.
#
# pos-condition
#      A file contend the classifier in standard output form
#
# 10-14-2000 modifications by Ronaldo C. Prati
# 03-26-2001 review by Ronaldo C. Prati
#
# arguments
#       0 - <inducer name>
#
# optionally
#       1 - -f<input file>
#       2 - -s<output file>
#       3 - -i (for additional information)
#
# You can use UNIX pipes for input or output files
#-----------------------------------------------------------------------

use std_rule_lib;

if ($#ARGV < 0)
{#inducer name is needed
   die "usage: std_rule <inducer> [-f<input file>] [-s<output file>] [-i]\n";
}

foreach (@ARGV)
{#parameters
   if (/-f/)
   {  s/-f//;
      $in = $_;
   }
   if (/-s/)
   {  s/-s//;
      $out = $_;
   }
   if (/-i/)
   {  $info = $_;
   }
}
#call library
$std = std_rule_lib->new($in,$out,$ARGV[0],$info);
$std->convert;
```

A.2 std_rule_lib.pm

```
#-----------------------------------------------------------------
# Module Standard rule conversor
#-----------------------------------------------------------------

package std_rule_lib;

#-----------------------------------------------------------------
# This script transform inducer's output in to standard output form
#
# pre-condition:
#      A file contend a classifier induced for id3, mc4, oc1, c4.5,
#      c4.5rules, c5.0, cn2, ripper or t2.
#
# pos-condition
#      A file contend the classifier in standard output form
#
# 10-14-2000 modifications by Ronaldo C. Prati
# 03-07-2001 review by Ronaldo C. Prati
#
# arguments
#       0 - input file
#       1 - output file
#       2 - inducer name
#       3 - -i aditional information (optional)
#-----------------------------------------------------------------


#-----------------------------------------------------------------
# create a new object
sub new
{ my $type = shift;
  my $self =
     { # $_[0] = input file,- = STDIN
       'inputfile'   => $_[0] ? $_[0] : "-",
         # $_[1] = output file, >- = STDOUT
       'outputfile'  => $_[1] ? $_[1] : ">-",
       #temporaty file
       'workfile'    => "$main::$$.".time().".output.tmp",
         # inducer name
       'inducer'     => $_[2],
         # -i if addcional information is chose
       'information' => $_[3] eq "-i"   ? "-i" : undef(),
         #script version
       'version'     => "1.1.2"
     };
  return bless $self,$type;
}
#-----------------------------------------------------------------
```

```
#-------------------------------------------------------------------
# start conversion, calling the correct procedure
sub convert
{ my $self;
  $self = shift;

  if ($self->{inducer} eq "c4.5")       # c4.5?
  {
     $self->c45_clean;                  # clear input file
     $self->c45_to_stdrule;             # call correct procedure
  }
  elsif ($self->{inducer} eq "cn2")
  {
     $self->cn2_clean;
     $self->cn2_to_stdrule;
  }
  elsif ($self->{inducer} eq "id3")
  {
     $self->id3_clean;
     $self->ID3_to_stdrule;
  }
  elsif ($self->{inducer} eq "c5.0")
  {
     $self->t2_clean;
     $self->c50_to_stdrule;
  }
  elsif ($self->{inducer} eq "ripper")
  {
     $self->ripper_clean;
     $self->ripper_to_stdrule;
  }
  elsif ($self->{inducer} eq "mc4")
  {
     $self->ID3_to_stdrule;
  }
  elsif ($self->{inducer} eq "t2")
  {
     $self->t2_clean;
     $self->C45_to_stdrule;
  }
  elsif ($self->{inducer} eq "oc1")
  {
     $self->oc1_to_stdrule;
  }
  elsif ($self->{inducer} eq "c4.5rules")
  {
     $self->c45rules_to_stdrule;
  }
  else
  {
     die "$self->{inducer}: unknow inducer";    # inducer not present
  }
unlink $self->{workfile};                       # erase workfile
}
#-------------------------------------------------------------------
```

```
#--------------------------------------------------------------------
# printer a header contents general information in output file
# commun for all inducers
sub print_header
{ my $self;
  $self = shift;

  open(OUT,">$self->{outputfile}") || die "Could not create output file\n";

  # copyright information
  print OUT "Standard Rules Conversor v$self->{version}\tCopyright
             (c)Ronaldo C. Prati\n";

  # files information
  print OUT "Inducer: $self->{inducer}\t\t\tInput File: $self->{inputfile}\n";

  # date and time information
  print OUT "Date: ".gmtime(time())."\n\n";
}
#--------------------------------------------------------------------


#--------------------------------------------------------------------
# Convert from c4.5 tree to standard format
# c4.5 output needs to be cleaned
#--------------------------------------------------------------------
sub c45_to_stdrule
{ my ($self, @node, $tree_level, $node_level,
      @stack, $rules, $rl_number, $buf);
  $self = shift;
  $tree_level = 0;
  $rl_number = 1;

  open(IN,"$self->{workfile}") || die "Could not open input file";
  $self->print_header();

  while(<IN>)
  {
    chop;
    $node_level = s/\|//g;      # count number of | (tree level)
    s/^\s+//;                   # white spaces at begin
    s/\s+$//;                   # white spaces at end
    @node = split /:/;          # @node[0] node , @node[1] leaf
    if ($node_level < $tree_level)      # go up on tree
    {
      for(;$tree_level > $node_level ; $tree_level--)
      {
        pop @stack;  # go back tree_level - node_level levels on tree
      }
    }
    $tree_level++;
    push (@stack,$node[0]);     # add node
    if ($node[1] =~ /\S+/)
```

```
    { # if is a leaf
       $rules = join("\n\t\tAND ",@stack);  #print rule
       if ($self->{information} ne "-i")
       {
          $node[1] =~ s/\([^\)]+\)//;
       }
       $buf = sprintf("R%04d",$rl_number++);
       print OUT "\n$buf\tIF $rules\n\tTHEN CLASS = $node[1]\n";
    }
  }

close IN;
close OUT;
}
#-------------------------------------------------------------------

#-------------------------------------------------------------------
# Convert from c4.5rule to standard format
sub c45rules_to_stdrule
{ my ($self, $rl_number, $in, $and);
  $self = shift;

  open(IN,"$self->{inputfile}") || die "Could not open input file";
  $self->print_header();

  while (<IN>)
  {
    s/^\s+//;
    if ($in eq "rule")
    {# in a rule
      if (/->/)
      {# then part
         s/->  class//;
         if ($self->{information} ne "-i")
         {# print aditional information?
            s/\([^\)]+\)//;
         }
         print OUT "\tTHEN CLASS = $_\n\n";
         $in = "not"; # find another rule
      }
      else
      {
         if ($and eq "false")
         {# first disjunct
            print OUT "$_";
            $and = "true";
         }
         else
         {# Ands ...
            print OUT "\t\tAND $_";
         }
      }
    }

    if (/Default/)
```

```
    {# default rule
      s/Default class://;
      $rl_number = sprintf("R%04d",$r++);
      print OUT "$rl_number\tDefault CLASS = $_";
      last;
     }

    if (/Rule/)
    {# new rule
      $in = "rule";
      $rl_number = sprintf("R%04d",$r++);
      print OUT "$rl_number\tIF ";
      $and = "false";
    }
  }
  close IN;
  close OUT;
}


#---------------------------------------------------------------------
# Convert from ID3 tree to standard format
sub ID3_to_stdrule
{ my ($self, @root_node, @root, @nodes, $buf, $tmp);
  $self = shift;

  open(IN,"$self->{workfile}") || die "Could not open input file";
  $self->print_header();

  @tree = <IN>; # read input file
  close IN;

  foreach (@tree)
  {
      chop;                              # prepare leaves to split
      s/\]\[/\]\|\[/g;
  }

  @root_node = split(/:/,@tree[0]);     # root node is in line 0

  $tmp = &InParents($root_node[0]);  # in parents = (atribuite, level)
  @root = split(/\,/,$tmp);

  @nodes = split(/\|/,$root_node[1]);   # next levels
  $rl_number = 1;
  foreach (@nodes)                      # find levels
  {
     my @aux, $x;
     @aux = split /\-\->|\-\-/ ;
     $buf = "\tIF @root[0] ".&compare(&InParents(@aux[1]));
     $x = &InBraces(@aux[2]);
     &eval($buf,$x);
  }
close OUT;
}
```

```
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# eval branchs from ID3 trees
sub eval
{ my ($aux, @this_node, @sub_root, $tmp, @tmp, $n, $if);

  ($if, $n) = @_;
  @this_node = split(/:/,@tree[$n]);    #what's this node?
  $tmp = &InParents($this_node[0]);
  @sub_root = split(/\,/,$tmp);

  if ($this_node[1] =~ /\S+/)
  {#it isn't a leaf
        @sub_nodes = split(/\|/,$this_node[1]);
        $aux = "$if\n\t\tAND $sub_root[0] ";
        foreach (@sub_nodes)
        {#go down on tree
           @tmp = split /\-\->|\-\-/ ;
           &eval($aux.&compare(&InParents(@tmp[1])),&InBraces(@tmp[2]));
        }
  }
  else
  {#it is a leaf
   #print rule
        $tmp = sprintf("R%04d",$rl_number++);
        print OUT "$tmp $if\n\tTHEN CLASS = $sub_root[0]\n\n";
  }

}
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# get in parents() contents
sub InParents
{
  $_[0] =~ /(\()([^\)]+)(\))/;
  $2;
}
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# get in braces [] contents
sub InBraces
{
  $_[0] =~ /(\[)([^\]]+)(\])/;
  $2
}
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# test a leaf
sub compare
{
  if ($_[0] =~ /[<>=]/)
```

```
  {
     "$_[0]";
  }
  else
  {
     "= $_[0]";
  }
}
#-------------------------------------------------------------


#-------------------------------------------------------------------
# Convert from ripper rules to standard format
sub ripper_to_stdrule
{ my ($self, @lines, @rule, $rl_count);
  $self = shift;
  $rl_count=1;

  open(IN,"$self->{workfile}") || die "Could not open input file";
  $self->print_header();

  @lines = <IN>;   #read input file
  close IN;

  foreach (@lines)
  {
    my $tmp = sprintf("\nR%04d",$rl_count++);
    chop;
    if (/default/)
    {# is default rule
      s/\(([^\)]+)\)./;
      s/default//;
      $add_info = $1;
      if ($self->{information} eq "-i")
      {
        print OUT "$tmp\tDEFAULT CLASS = $_ ($add_info)\n";
      }
      else
      {
        print OUT "$tmp\tDEFAULT CLASS = $_\n";
      }
    }
    else
    { #is a normal rule
      @rule = split(/:-/,$_);           #separe if ... then terms
      $rule[1] =~ s/\(([^\)]+)\)./;    #get adittional information
      $add_info = $1;
      $rule[1] =~ s/,/\n\t\tAND/g;      #puts if term in standard format
      print OUT "$tmp\tIF$rule[1]";
      if ($self->{information} eq "-i") #print then term
      {
        print OUT "\n\tTHEN CLASS = $rule[0] ($add_info)\n";
      }
      else
      {
```

```
        print OUT "\n\tTHEN CLASS = $rule[0]\n";
      }
    }
  }
close OUT;
}
#--------------------------------------------------------------------

#--------------------------------------------------------------------
# Convert from cn2 rules to standard format
sub cn2_to_stdrule
{ my ($self, $rl_count, $add_info, @lines);
  $self = shift;
  $rl_count = 1;

  open(IN,"$self->{workfile}") || die "Could not open input file";
  $self->print_header();

  @lines = <IN>;        #read input file
  close IN;

  foreach(@lines)
  {
     chop;
     if(/IF/)
     {#if term, put it in standard form
        my $tmp = sprintf("\nR%04d",$rl_count++);
        s/    / /;
        print OUT "$tmp\t$_\n";
     }
     if(/THEN/)
     {#them term
        s/  / /;
        s/Class/CLASS/;
        s/\[([^\]]+)\]//;
        $add_info = $1;
        if ($self->{information} eq "-i")
        {
           print OUT "\t$_ ($add_info)\n";
        }
        else
        {
           print OUT "\t$_\n";
        }
     }
     if (/AND/)
     {#and term
        s/  /'\t'/;
        print OUT "\t$_\n";
     }
     if (/\(DEFAULT\)/)
     {#default rule
        s/Class/CLASS/;
        s/\[([^\]]+)\]//;
        s/[\(\)]//g;
```

```
        $add_info = $1;
        my $tmp = sprintf("\nR%04d",$rl_count++);
        if ($self->{information} eq "-i")
        {
           print OUT "$tmp\t$_ ($add_info)\n";
        }
        else
        {
           print OUT "$tmp\t$_\n";
        }
     }
  }
}
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# convert from c5.0/see5.0 tree to standard format
sub c50_to_stdrule
{ my ($self, @node, $tree_level, $node_level, @stack, $rules, $rl_number, $buf);
  $self = shift;
  $tree_level = 0;
  $rl_number = 1;

  open(IN,"$self->{workfile}") || die "Could not open input file";
  $self->print_header();

  while(<IN>)
  {
    chop;
    s/:\.\.\./|/g;              # convert to c4.5 format
    s/:   /|/g;
    s/    /|/g;
    $node_level = s/\|//g;      # count a number of | (tree level)
    s/$\s+//;                   # white spaces at end
    @node = split /:/;          # @node[0] node , @node[1] leave
    if ($n < $x)
    {
      for(;$tree_level > $node_level ;$tree_level--)
      {
        pop @stack; # go back tree_level - node_level levels on tree
      }
    }
    $tree_level++;
    push (@stack,$node[0]); # add node
    if ($node[1] =~ /\S+/)
    { # if has a leave
       $rules = join("\n\t\tAND ",@stack);  #print rule
       if ($self->{information} ne "-i")
       {
          $node[1] =~ s/\([^\)]+\)//;
       }
       $buf = sprintf("R%04d",$rl_number++);
       print OUT "\n$buf\tIF $rules\n\tTHEN CLASS = $node[1]\n";
    }
  }
```

```
close IN;
close OUT;
}
#--------------------------------------------------------------------

#--------------------------------------------------------------------
# convert from OC1 tree to standard format
sub oc1_to_stdrule()
{ my ($n, $self);
  $self = shift;
  $n = 1;
  $self->print_header();

  open (IN,"$self->{inputfile}"); #read input file
  @tmp = <IN>;
  close IN;

  &find("Root");                #go down on tree
  close OUT;

}
#--------------------------------------------------------------------

#--------------------------------------------------------------------
# finds OC1's subtrees
sub find()
{  my ($aux, $found, $complex, $right, $left);
   $aux = @_[0];
   $found = 0;

   for($i = 0; $i <= $#tmp; $i++)
   {#seek in file
      if ($tmp[$i] =~ m/^$aux/m)
      {#seek a subtree
         $found = $i; #found on line i
         last;
      }
   }
   if (!$found)
   {# is a leaf
      printf OUT ("R%04d\tIF %s\n\tTHEN CLASS = %s\n\n",$n++,@_[1],@_[2]);
      return 0;
   }

   chop ($tmp[$found+1]);
   $tmp[$found] =~ /\[([^\]]+)\]/;
   $left = $1;
   $tmp[$found] =~ /Right = \[([^\]]+)\]/;
   $right = $1;

   #find subtrees
   if ($aux eq "Root")
   {
      $complex = "$tmp[$found+1]";
```

```
      &find("l","$complex < 0",&max($left));
      &find("r","$complex > 0",&max($right));
   }
   else
   {
      $complex = "@_[1]\n\t\tAND $tmp[$found+1]";
      &find($aux.'l',"$complex < 0",&max($left));
      &find($aux.'r',"$complex > 0",&max($right));
   }
}
#---------------------------------------------------------------------


#---------------------------------------------------------------------
# find max value in a vector
sub max()
{
   my (@array, $return, $max);
   @array = split(/,/,@_[0]);
   $max = 0;
   for ($i = 0; $i <= $#array; $i++)
   {
      if ($array[$i] > $max)
      {
         $max = $array[$i];
         $return = $i;
      }
   }
   ($return+1);
}
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# clear t2 and c5.0 files
sub t2_clean()
{ my ($self,$on);
  $self = shift;

  open(IN,"$self->{inputfile}");
  open(AUX,">$self->{workfile}");
  $on="false";

  while(<IN>)
  {
    if (/\S/)
    {
      if (/^(Evaluation on training data)/)
      {
        $on="false";
      }
      if ($on eq "true")
      {
        print AUX $_;
      }
      if (/^(Decision Tree:)/)
```

```
      {
        $on="true";
      }
    }
  }
}
#--------------------------------------------------------------------

#--------------------------------------------------------------------
# clear ripper files
sub ripper_clean()
{ my ($self,$on);
  $self = shift;

  open(IN,"$self->{inputfile}");
  open(AUX,">$self->{workfile}");
  $on="false";

  while(<IN>)
  {
    if (/\S/)
    {
      if ($on eq "true")
      {
        print AUX $_;
      }
      if (/^default/)
      {
        $on="false";
      }
      if (/^(Final hypothesis is:)/)
      {
        $on="true";
      }
    }
  }
  close IN;
  close AUX;
}
#--------------------------------------------------------------------

#--------------------------------------------------------------------
# clear c4.5 files
sub c45_clean()
{ my ($on,$self);
  $self = shift;

  open(IN,"$self->{inputfile}");
  open(AUX,">$self->{workfile}");
  $on="false";

  while(<IN>)
  {
    if (/\S/)
    {
```

```
      if (/^(Tree saved)/)
      {
        $on="false";
        last;
      }
      if ($on eq "true")
      {
        print AUX $_;
      }
      if (/^(Simplified Decision Tree:)/)
      {
        close AUX;
        open(AUX,">$self->{workfile}");
      }
      if (/^(Decision Tree:)/)
      {
        $on="true";
      }
    }
  }

  close IN;
  close AUX;
}
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# clear id3 files
sub id3_clean()
{ my ($self);
  $self = shift;

  open(IN,"$self->{inputfile}");
  open(AUX,">$self->{workfile}");

  while (<IN>){
    if (/(\[)([^\]]+)(\])/)
    {
       print AUX $_;
    }
  }
  close IN;
  close AUX;
}
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# clear cn2 files
sub cn2_clean()
{ my ($on,$self);
  $self = shift;

  open(IN,"$self->{inputfile}");
  open(AUX,">$self->{workfile}");
  $on="false";
```

```
  while(<IN>)
  {
    if (/\S/)
    {
      if (/^(IF)/)
      {
        $on="true";
      }
      if ($on eq "true")
      {
        print AUX $_;
      }
      if (/^(\(DEFAULT\))/ )
      {
        $on="true";
      }
    }
  }
  close IN;
  close AUX;
}

1;

__END__
```

Referências

Auer, P., Holte, R., and Maass, W. (1995). Theory and applications of agnostic pac-learning with small decision trees. In *Machine Learning: Procedings of the Twelfth International Conference*, Morgan Kaufmann. A. Prieditis & S. Russel.

Baranauskas, J. A. and Monard, M. C. (2000a). Reviewing some machine learning concepts and methods. Technical Report 102, ICMC-USP. `fpt://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_102.ps.zip`.

Baranauskas, J. A. and Monard, M. C. (2000b). An unified overview of six supervisioned symbolic machine learning inducers. Technical Report 103, ICMC-USP. `fpt://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_103.ps.zip`.

Clark, P. and Boswell, R. (1989). The $\mathcal{CN}2$ induction algorothm. *Machine Learning*, 3(4):261–283.

Clark, P. and Boswell, R. (1991). Rule induction with $\mathcal{CN}2$: Some recent improvements. In *Procedings of the Fifth European Conference*, pages 151–163, Springer Verlag. Y. Kodratoff. Avaliable at `http://www.cs.utexas.edu/users/pclark/papers/newcn.ps`.

Cohen, W. W. (1995). Fast effective rule induction. In *Machine Learning: Procedings of the Twelfth International Conference*, pages 115–123, San Francisco, CA. Morgan Kaufmam.

Félix, L. C. and et. all (1998). $\mathcal{MLC}$++ biblioteca de aprendizado de máquima em *C++*. Technical report, ICMC-USP. Technical Report 72.

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.

Freitas, A. A. (1998b). On objective measures of rule surprisingness. In *Principles of Data Mining & Knowledge Discovery: Proceedings of the Second European Symp. Lecture Notes in Artificial Intelligence*, volume 1510, pages 1–9.

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

Kohavi, R., Sommerfield, D., and Dougherty, J. (1996). Data mining using $\mathcal{MLC}$++: A machine learning library in *C++*. *Tools with Artificial Intelligence*, pages 234–245.

Kohavi, R., Sommerfield, D., and Dougherty, J. (1997). Data mining using $\mathcal{MLC}$++: A machine learning library in *C++*. *International Journal on Artificial Intellegence Tools*.

Merz, C. J. and Murphy, P. M. (1998). UCI repository of machine learning datasets. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Murthy, S. K., Kasif, S., and Salzberg, S. L. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2(1):1–32. http://www.cs.jhu.edu/∼salzberg/jair94.ps.

MySQL (2000). The $\mathcal{MySQL}$ server. http://www.mysql.com.

PERL (1999). *Programming in PERL*. Morgan Kaufmann Publishers, Inc.

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.

Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann, CA.

Quinlan, J. R. (1996). *Induction of Decision Trees*, volume 1, pages 81–106. Shavlik and Dietterich.

Salzberg, S. L. (1995). Locating protein coding regions in human dna using a decision tree algorithm. *Journal of Computational Biology*, 2(3):473–485.

Salzberg, S. L., Chandar, R., Ford, H., Murthy, S., and White, R. (1995). Decision trees for automated identification of cosmic ray hits in Hubble space telescope images. *Publications of the Astronomical Society of the Pacific*, 107:1–10.

Weiss, S. M. and Indurkhya, N. (1998). *Predictive Data Mining: A Practical Guide*. Morgan Kaufmann, San Francisco, CA.