Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Uma Sintaxe Padrão Prolog para Classificadores Simbólicos

Alan Keller Gomes
Flávia Cristina Bernadini
Maria Carolina Monard
Gustavo E. A. P. A. Batista

Nº 154

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Janeiro/2002

Uma Sintaxe Padrão Prolog para Classificadores Simbólicos *

Alan Keller Gomes
Flávia Cristina Bernardini
Maria Carolina Monard
Gustavo E. A. P. A. Batista

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: {akgomes, fbernard, gbatista, mcmonard}@icmc.sc.usp.br

Resumo

A aplicação de diferentes algoritmos de Aprendizado de Máquina supervisionado sobre um conjunto de exemplos de treinamento induz a descrição de um conceito (classificador) implícito nesses exemplos.

Quando um classificador é obtido por meio de um algoritmo de aprendizado simbólico, o conceito ali descrito é diretamente interpretável por seres humanos e, geralmente, pode ser transformado em conjunto de regras. Um classificador simbólico, descrito sob a forma de regras, pode ser avaliado por outros indicadores que não só a precisão de classificação. De um modo mais específico, é possível avaliar cada uma das regras individualmente, utilizando informações que permitam derivar facilmente medidas que facilitem a análise e a interpretação das regras que constituiem esse classificador.

Com o objetivo de unificar o formato do conhecimento descrito por diferentes classificadores simbólicos, foi proposto em (Prati et al., 2001b; Prati et al., 2001a) um formato padrão para regras, denominado de formato $\mathcal{PBM}$ para, entre outros, facilitar a avaliação, utilizando diversas medidas de cada uma das regras do conjunto de regras que constituem um classificador.

O trabalho aqui apresentado tem por objetivo propor uma sintaxe equivalente ao formato padrão de regras e de exemplos, de acordo com especificações da linguagem Prolog. A conversão do formato padrão $\mathcal{PBM}$ para a sintaxe padrão Prolog é implementada através de *s*cripts Perl.

Palavras-Chave: Aprendizado de Máquina, Conhecimento Simbólico, Regras de Conhecimento.

Janeiro 2002

*Trabalho realizado com auxílio da CAPES, CNPq e FAPESP

Sumário

1 Introdução **1**

2 Sintaxe Padrão de Exemplos **2**

3 Sintaxe Padrão de Regras **4**

4 Sintaxe Padrão Prolog de Exemplos **9**

5 Sintaxe Padrão Prolog de Regras **11**

6 Considerações Finais **12**

A Conversor de Regras **16**

Referências Bibliográficas **21**

Lista de Figuras

1 Conversores do Formato Padrão $\mathcal{PBM}$ para Sintaxe Padrão Prolog 2

2 Gramática do Arquivo de Nomes (Prati et al., 2001a) 3

3 O Arquivo `voyage.names` (Prati et al., 2001a) 5

4 O Arquivo `voyage.data` (Prati et al., 2001a) 5

5 Gramática do Formato Padrão $\mathcal{PBM}$ (Prati et al., 2001b) 6

6 Exemplo de um Arquivo no Formato $\mathcal{PBM}$ (Prati et al.,2001b) 6

7 Gramática do Formato Padrão de Regras Estendido (Prati et al., 2001a) 7

8 Exemplo de um Arquivo no Formato $\mathcal{PBM}$ Estendido (Prati et al.,2001a) 8

9 Gramática da Sintaxe Padrão Prolog para Exemplos 9

10 Representação Prolog do Conjunto de Exemplos *Voyage* 10

11 Gramática da Sintaxe Padrão Prolog para Regras 14

12 Exemplo de um Arquivo com um Conjunto de Regras na Sintaxe Prolog 15

13 Exemplo de um Arquivo com um Conjunto de Regras na Sintaxe Prolog 15

Lista de Tabelas

1 O Conjunto de Exemplos Voyage (Prati et al., 2001a) 4

2 Valores Nominais de Operadores 13

[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

Encontra-se em desenvolvimento no Laboratório de Inteligência Computacional —LABIC[1]— um projeto de grande porte denominado DISCOVER, inicialmente proposto por Baranauskas e Batista (Baranauskas and Monard, 2000b). O projeto DISCOVER tem como objetivo fornecer um ambiente integrado para apoiar as etapas do processo de descoberta de conhecimento, oferecendo funcionalidades voltadas para Aprendizado de Máquina (AM) (Batista, 1997; Caulkins, 2000; Milaré, 2000; Martins, 2001), Mineração de Dados[2](MD) (Batista, 2000; Horst, 1999; Lee, 2000; Monard et al., 1999; Nagai, 2000; Pila and Monard, 2001; Pugliesi, 2001; Baranauskas, 2001) e Mineração de Textos[3](MT) (Imamura, 2001).

As funcionalidades voltadas para AM consideram, entre outros, um formato padrão para as regras induzidas por algoritmos de AM simbólico (Prati et al., 2001b; Prati et al., 2001a) bem como formato padrão para os exemplos utilizados (Batista, 2001), denominado $\mathcal{PBM}$.

No intuito de testar algumas idéias que poderão futuramente ser implementadas no ambiente DISCOVER, foram propostos em (Bernardini, 2001) e (Gomes, 2001) dois sistemas computacionais com o objetivo de investigar alguns métodos para construir *ensembles* de classificadores (Bauer and Kohavi, 1999; Dietterich, 1997; Dietterich, 2000), bem como analisar diversas medidas propostas na literatura para avaliar regras induzidas por algoritmos de AM simbólico (Lavrac et al., 1999; Freitas, 1998b; Freitas, 1998a; Hilderman and Hamilton, 1999).

Foi decidido implementar ambos sistemas na linguagem de programação lógica Prolog (Bratko, 1990) mais especificamente em LPA-PROLOG (Westwood, 2000b; Westwood, 2000a). A escolha da linguagem Prolog deve-se ao fato de ser uma linguagem apropriada para o desenvolvimento de protótipos nos quais novas idéias possam ser facilmente tratadas antes de serem implementadas em um sistema de grande porte. Uma outra característica importante de Prolog é a possibilidade de realizar consultas para obter informações que não ficam restritas àquelas pré-definidas no sistema.

A entrada para esses sistemas consiste de:

1. regras induzidas por algoritmos de AM simbólico;
2. um conjunto de exemplos.

ambos na sintaxe padrão $\mathcal{PBM}$ adotada no projeto DISCOVER.

O trabalho aqui apresentado tem por objetivo propor uma sintaxe equivalente à sintaxe padrão $\mathcal{PBM}$ de regras e exemplos, de acordo com as especificações da linguagem Prolog. Essa sintaxe será referida, ao longo deste trabalho, como **sintaxe padrão Prolog**. A conversão do formato padrão $\mathcal{PBM}$ para a sintaxe padrão Prolog é implementada através de *scripts* Perl. A Figura 1 ilustra a interação desses *scripts* na conversão da sintaxe de diferentes conjuntos de regras e de exemplos, no formato padrão do DISCOVER ($\mathcal{PBM}$), para a sintaxe padrão Prolog aqui proposta.

[1]http://labic.icmc.sc.usp.br
[2]Data Mining
[3]Text Mining

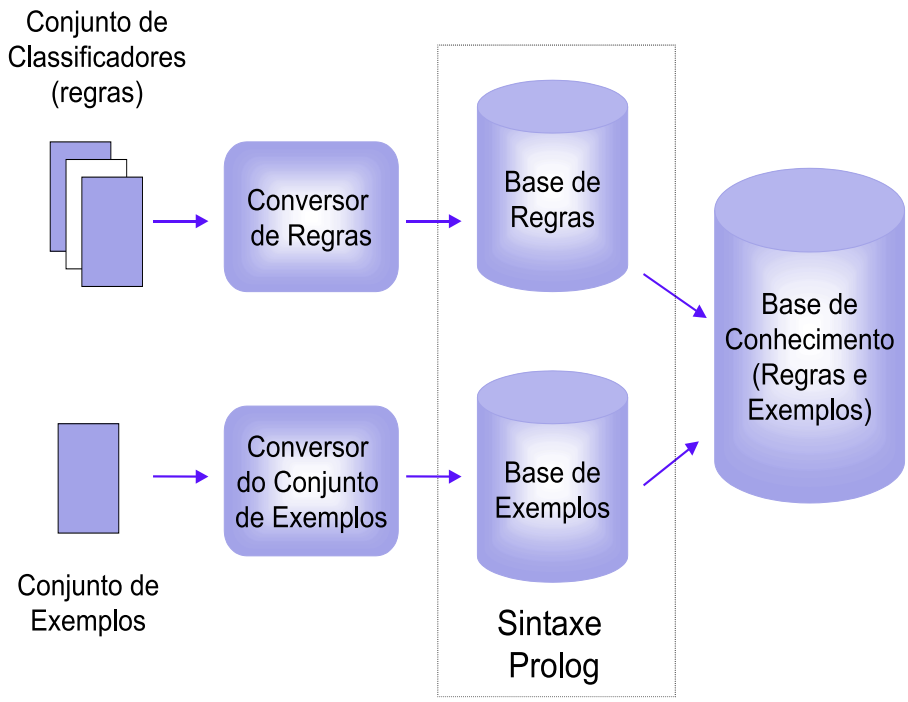

Figura 1: Conversores do Formato Padrão $\mathcal{PBM}$ para Sintaxe Padrão Prolog

O trabalho está organizado da seguinte maneira: na Seção 2 é apresentada a sintaxe padrão de exemplos e na Seção 3 a sintaxe padrão de regras utilizada no ambiente DISCOVER. Na Seção 4 é apresentada a sintaxe padrão Prolog de exemplos e na Seção 5 a sintaxe padrão Prolog de regras proposta nesse trabalho. No apêndice A é apresentado o *script* Perl que realiza a conversão da sintaxe padrão de regras para a sintaxe padrão Prolog. Para realizar a conversão da sintaxe padrão de exemplos para a sintaxe padrão Prolog foi utilizado um *script* Perl integrado a uma biblioteca de manipulação de conjuntos de dados. Futuramente poderão ser obtidas maiores informações sobre essa biblioteca, a medida que forem concluídos os trabalhos mencionados em (Batista, 2001).

2 Sintaxe Padrão de Exemplos

A sintaxe padrão de exemplos proposta em (Batista, 2001) é uma extensão do formato dos arquivos de nomes e dados do $\mathcal{C}4.5$ (Quinlan, 1988) e utiliza arquivos do tipo texto para declarar os nomes dos atributos (e seus respectivos domínios) bem como os valores que esses atributos assumem no conjunto de exemplos.

Os nomes dos atributos são declarados em um arquivo de nomes, designado com a extensão `.names`. A Figura 2 mostra a gramática que define a sintaxe do arquivo de nomes. Os valores que esses atributos assumem no conjunto de exemplos são declarados no arquivo de dados, designados com a extensão `.data`. Um conjunto de exemplos somente está na sintaxe padrão se estiverem

```
S ::= <class-defs> | <feature-defs>

<class-defs> ::= <feature-name>. | null.

<feature-name> ::= <identifier>

<feature-defs> ::= <feature-name> : <feature-type> .
                 | <feature-name> : <feature-type> : <extended-defs> .
                 | <feature-name> : <feature-type> := <expression> : <extended-defs> .

<feature-type> ::= real | integer | boolean | nominal
                 | nominal(<list>) | enumerated (<list>)
                 | date | time | string

<extended-defs> ::= <extended-def>
                  | <extended-def> : <extended-defs>

<extended-def> ::= <identifier> | <identifier> (<list>)

<list> ::= <identifier> | <identifier> , <list>
```

Figura 2: Gramática do Arquivo de Nomes (Prati et al., 2001a)

presentes os dois arquivos (`.names` e `.data`) que devem possuir o mesmo nome, se diferenciando apenas pela extensão.

Os atributos declarados em um arquivo de nomes podem assumir qualquer um dos seguintes tipos de dados:

Nominal: O tipo de dado `nominal` é utilizado para declarar um atributo discreto, ou seja que pode assumir um conjunto finito de valores.

Enumerated: O tipo de dado `enumerated` é muito semelhante ao tipo de dado `nominal`. A principal diferença é que com o tipo `enumerated` é possível identificar uma ordem entre os valores que o atributo pode assumir. Entretanto, não existe uma definição clara de distância entre esses valores. Um exemplo de tipo `enumerated` é um atributo que pode assumir, por exemplo, os valores *pequeno*, *médio* e *grande*.

Integer: O tipo de dado `integer` é utilizado para declarar um atributo que pode assumir valores inteiros.

Real: O tipo de dado `real` é semelhante ao tipo de dado `integer`, com a diferença que um atributo `real` pode assumir números com ou sem parte fracionária.

String: Um atributo `string` pode assumir como valor um string de tamanho indefinido o qual pode conter quaisquer caracteres incluindo quebra de linha (**n**). Para identificar os limites de um string é necessário inserir o símbolo de aspas (“) antes e depois do string.

Date: O tipo de dado `date` permite declarar um atributo que pode conter uma data (dia, mês e ano). A princípio, os valores das datas devem estar no formato "aaaa/mm/dd" (formato utilizado pela maioria dos sistemas de gerenciamento de bancos de dados).

Time: O tipo de dado `time` permite declarar um atributo que pode conter um horário (hora, minuto e segundo). A princípio, os valores dos horários devem estar no formato "hh:mm:ss".

A primeira declaração em um arquivo de nomes define qual deve ser o nome do atributo classe. O atributo classe pode ser qualquer atributo presente no conjunto de exemplos. Após a declaração do atributo classe, são declarados os demais atributos.

O arquivo de dados correspondente a um arquivo de nomes contém n exemplos, um exemplo em cada linha. Cada exemplo i consiste nos valores dos atributos desse exemplo, separados por vírgula, que devem pertencer ao domínio correspondente aos atributos especificados no arquivo de nomes. Valores "*desconhecidos*" são representados por um ponto de interrogação (?) e valores "*não-se-aplica*" são representados com um ponto de exclamação (!).

O conjunto de exemplos na Tabela 1 será utilizado para ilustrar o conteúdo do arquivo de nomes e de dados na sintaxe padrão $\mathcal{PBM}$ do sistema DISCOVER.

Example No.	Outlook	Temperature	Humidity	Windy	Voyage?
E_1	sunny	25	72	yes	go
E_2	sunny	28	91	yes	dont_go
E_3	sunny	22	70	no	go
E_4	sunny	23	95	no	dont_go
E_5	sunny	30	85	no	dont_go
E_6	overcast	23	90	yes	go
E_7	overcast	29	78	no	go
E_8	overcast	19	65	yes	dont_go
E_9	overcast	26	75	no	go
E_{10}	overcast	20	87	yes	go
E_{11}	rain	22	95	no	go
E_{12}	rain	19	70	yes	dont_go
E_{13}	rain	23	80	yes	dont_go
E_{14}	rain	25	81	no	go
E_{15}	rain	21	80	no	go

Tabela 1: O Conjunto de Exemplos Voyage (Prati et al., 2001a)

As figuras 3 e 4 mostram respectivamente, os arquivos de nomes e de dados correspondentes a esse conjunto de exemplos.

3 Sintaxe Padrão de Regras

No intuito de unificar a linguagem de representação do conhecimento induzido (classificador) por diferentes algoritmos de AM simbólico, foi proposto em (Prati et al., 2001b) o formato padrão de regras $\mathcal{PBM}$. Esse formato é definido pela gramatíca da Figura 5.

```
voyage          | Class Attribute

| Attributes
outlook:       nominal (sunny, overcast, rain).
temperature:   integer.
humity:        integer.
windy:         nominal (yes, no).
voyage:        nominal (go, dont_go).
```

Figura 3: O Arquivo `voyage.names` (Prati et al., 2001a)

```
sunny,25, 72, yes, go
sunny,28, 91, yes, dont_go
sunny,22, 70, no,  go
sunny,23, 95, no,  dont_go
sunny,30, 85, no,  dont_go
overcast,23, 90, yes, go
overcast,29, 78, no,  go
overcast,19, 65, yes, dont_go
overcast,26, 75, no,  go
overcast,20, 87, yes, go
rain,22, 95, no,  go
rain,19, 70, yes, dont_go
rain,23, 80, yes, dont_go
rain,25, 81, no,  go
rain,21, 80, no,  go
```

Figura 4: O Arquivo `voyage.data` (Prati et al., 2001a)

Um classificador simbólico é qualquer classificador que possa ser transformado num conjunto de regras do tipo *if* <condição> *then* <class = C_i >. Para realizar essa transformação, foi implementada uma biblioteca composta por uma série de *scripts* Perl (Prati et al., 2001b). Na Figura 6 é mostrado um arquivo no formato $\mathcal{PBM}$.

Logo após, derivadas de uma tabela de contingência, foi definido um conjunto mínimo de informações que permitem calcular medidas de avaliação de regras. Esse conjunto de informações é obtido através da avaliação de cada uma das regras do conjunto de regras que constituem o classificador, no formato padrão de regras $\mathcal{PBM}$, utilizando um conjunto de exemplos fornecido pelo usuário. Como uma extensão do formato $\mathcal{PBM}$, após serem calculadas, essas informações são adicionadas ao final de cada uma das regras (Prati et al., 2001a).

Denotando uma regra como $Body \rightarrow Head$ ou resumidamente $B \rightarrow H$, o formato padrão proposto

```
S ::= <rule>

<rule> ::= IF <complex> THEN <class>

<complex> ::= <fator> | <fator> OR <condition>

<fator> ::= <term> | <term> AND <fator>

<term> ::= <comparison> | <linear combination>

<comparison> ::= <atribute><operator><value>

<linear combination> ::= <combination><operator><value>

<combination> ::= <constant> x <numeric attribute>
                | <constant> x <numeric attribute> + <combination>

<operator> ::= <=,>=,in,<,>,!=,=

<class> ::= CLASS = <value>
```

Figura 5: Gramática do Formato Padrão $\mathcal{PBM}$ (Prati et al., 2001b)

```
Standard Rules Conversor v1.1.4 Copyright (c)Ronaldo C. Prati
Inducer: cn2-ordered     Input File: voyage.cn2.ordered_output
Date: Thu Jun 14 21:15:06 2001

Rules Evaluated as ORDERED
Names File: voyage.names   Data File: voyage.test

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

R0002   IF outlook = sunny
            AND humidity > 78.50
        THEN CLASS = dont_go

R0003   IF humidity > 83.50
        THEN CLASS = go

R0004   IF temperature < 24.00
        THEN CLASS = dont_go

R0005   DEFAULT CLASS = go
```

Figura 6: Exemplo de um Arquivo no Formato $\mathcal{PBM}$ (Prati et al., 2001b)

```
S ::= <rule>

<rule> ::= IF <complex> THEN <class> <extension>

<complex> ::= <fator> | <fator> OR <condition>

<fator> ::= <term> | <term> AND <fator>

<term> ::= <comparison> | <linear combination>

<comparison> ::= <atribute><operator><value>

<linear combination> ::= <combination><operator><value>

<combination> ::= <constant> x <numeric attribute>
                | <constant> x <numeric attribute> + <combination>

<operator> ::= <=,>=,in,<,>,!=,=

<class> ::= CLASS = <value>

<extension> ::= <know values> <unknow values>

<know values> ::= [<value>, <value>, <value>, <value>, <value>]

<unknow values> ::= ?[<value>, <value>, <value>, <value>, <value>]
```

Figura 7: Gramática do Formato Padrão de Regras Estendido (Prati et al., 2001a)

para as informações adicionais é o seguinte:

$$\left[f_{bh}, f_{b\overline{h}}, f_{\overline{bh}}, f_{\overline{b}h}, n\right]$$

onde n é o número total de exemplos e f representa a freqüência relativa de:

hb número de exemplos para os quais B é verdade e H é verdade.

$b\overline{h}$ número de exemplos para os quais B é verdadeiro e H é falso.

$\overline{bh}$ número de exemplos para os quais B é falso e H é falso.

$\overline{b}h$ número de exemplos para os quais B é falso e H é verdade.

O formato $\mathcal{PBM}$ estendido é definido pela gramática na Figura 7. Esse formato padrão de regras contém as freqüências relativas f_{bh}, $f_{b\overline{h}}$, $f_{\overline{bh}}$, e $f_{\overline{b}h}$, além do número de exemplos, n, utilizado no cálculo dessas métricas. Deve ser observado que tanto valores conhecidos como valores desconhecidos nos exemplos são considerados no cálculo dessas métricas. Valores absolutos das medidas também podem ser facilmente obtidos multiplicando as freqüências relativas por n.

Na Figura 8 é mostrado um arquivo no formato $\mathcal{PBM}$ estendido.

```
Standard Rules Conversor v1.1.4 Copyright (c)Ronaldo C. Prati
Inducer: cn2-ordered     Input File: voyage.cn2.ordered_output
Date: Thu Jun 14 21:15:06 2001

Rules Evaluated as ORDERED
Names File: voyage.names   Data File: voyage.test

R0001   IF humidity < 83.00
            AND windy = no
        THEN CLASS = go  [0.250,0.083,0.417,0.250,12] ?[0.333,0.333,0.333,0.000,3]

R0002   IF outlook = sunny
            AND humidity > 78.50
        THEN CLASS = dont_go  [0.154,0.000,0.462,0.385,13] ?[0.500,0.000,0.500,0.000,2]

R0003   IF humidity > 83.50
        THEN CLASS = go  [0.231,0.000,0.538,0.231,13] ?[0.500,0.500,0.000,0.000,2]

R0004   IF temperature < 24.00
        THEN CLASS = dont_go  [0.200,0.000,0.467,0.333,15] ?[0.000,0.000,0.000,0.000,0]

R0005   DEFAULT CLASS = go
```

Figura 8: Exemplo de um Arquivo no Formato $\mathcal{PBM}$ Estendido (Prati et al., 2001a)

4 Sintaxe Padrão Prolog de Exemplos

Partindo da sintaxe padrão $\mathcal{PBM}$ do projeto DISCOVER para a conversão de um conjunto de exemplos nessa sintaxe padrão para a sintaxe padrão Prolog é feita através de um *script* Perl. Por sua vez, esse *script* utiliza uma biblioteca que implementa um grupo de sub-rotinas de manipulação de conjuntos de dados na sintaxe padrão. A Figura 9 apresenta a gramática que define a sintaxe padrão Prolog para um conjunto de exemplos.

```
S::= <example> | <feature> | <class_feature>

<example> ::= ex(<example_position>, <list_attribute_values>).

<feature> ::= feature(<feature_position>, <feature_identifier>, <feature_type>).

<class_feature> ::= classFeature(<feature_identifier>).

<feature_type> ::= nominal(<list_feature_type_values>) | numeric.
```

Figura 9: Gramática da Sintaxe Padrão Prolog para Exemplos

Como visto anteriormente, um conjunto específico de exemplos na sintaxe padrão está definido através das informações contidas nos arquivos `.data` e `.names`.

Assim, o predicado `ex/2` (Figura 10) é utilizado para representar em Prolog cada um dos exemplos contidos no arquivo `.data`. Os dois argumentos do predicado `ex/2` são:

<**arg-1**> posição (ou número) do exemplo no arquivo `.data`, começando a partir de 0 (zero);

<**arg-2**> lista com os valores que cada atributo pode assumir. A ordem desses valores na lista está dada pela ordem em que os atributos estão declarados no arquivo `.names`.

Os predicados `feature/3` e `classFeature/1` (Figura 10) são utilizados para representar em Prolog as informações sobre os atributos contidos no arquivo `.names`.

Os três argumentos do predicado `feature/3` são:

<**arg-1**> posição desse atributo no conjunto de dados. A ordem dos atributos é a mesma que a declarada no arquivo `.names`, começando em zero.

<**arg-2**> nome do atributo. O conversor da sintaxe padrão para a sintaxe padrão Prolog é responsável por verificar se o nome do atributo é um identificador Prolog válido. Por exemplo, no caso do nome do identificador iniciar com uma letra maiúscula (o que tornaria o nome do atributo em uma variável Prolog), o conversor é responsável por substituir por uma letra minúscula.

<**arg-3**> tipo do atributo. Existem dois tipos básicos na sintaxe padrão Prolog: `numeric` e `nominal`. O tipo de dados `numeric` abrange os tipos de dados inteiro e real da sintaxe

padrão. O tipo de dados *nominal* abrange os tipo de dados `nominal` e `enumerated` da sintaxe padrão. O tipo de dados `nominal` é declarado na forma de um functor Prolog. Os argumentos do functor são os valores que um atributo `nominal` pode assumir. O conversor da sintaxe padrão $\mathcal{PBM}$ para a sintaxe padrão Prolog é responsável por verificar quais valores um atributo `nominal` pode assumir. Os tipos `date` e `time` da sintaxe padrão de exemplos não são suportados na sintaxe padrão Prolog. Desta forma, o conversor entre as duas sintaxes é responsável por converter os atributos previamente dos tipos `date` e `time` em atributos do tipo `numeric`. Atributos do tipo `date` são convertidos no número de dias desde o primeiro dia do ano zero. Os atributos do tipo `time` são convertidos no número de segundos desde o primeiro segundo do dia. Obviamente, essas conversões automáticas podem não ser as melhores conversões para um determinado domínio. Entretanto, o usuário é livre para converter os atributos dos tipos `date` e `time` em quaisquer outros tipos antes que o conversor da sintaxe padrão para a Sintaxe Prolog seja executado.

O argumento do predicado `classFeature/1` é simplesmente o nome da feature que corresponde à classe.

A Figura 10 mostra a representação na sintaxe padrão Prolog, do conjunto de exemplos *Voyage* mostrado na Tabela 1.

```
ex(0, [sunny, 25, 72, yes, go]).
ex(1, [sunny, 28, 91, yes,dont_go]).
ex(2, [sunny, 22, 70, no, go]).
ex(3, [sunny, 23,95, no,  dont_go]).
ex(4, [sunny,30, 85, no,  dont_go]).
ex(5, [overcast, 23, 90, yes, go]).
ex(6, [overcast, 29, 78, no,go]).
ex(7, [overcast, 19, 65,yes, dont_go]).
ex(8, [overcast, 26, 75, no,  go]).
ex(9, [overcast, 20, 87, yes, go]).
ex(10,[rain, 22, 95, no,  go]).
ex(11,[rain, 19, 70, yes, dont_go]).
ex(12,[rain, 23, 80,yes, dont_go]).
ex(13,[rain, 25, 81, no,  go]).
ex(14,[rain, 21, 80, no, go]).

feature(0, outlook, nominal(overcast,rain,summy)).
feature(1,temperature, numeric).
feature(2, humidity, numeric).
feature(3, windy, nominal(yes,no)).
feature(4, voyage, nominal(go,dont_go)).

classFeature(voyage).
```

Figura 10: Representação Prolog do Conjunto de Exemplos *Voyage*

5 Sintaxe Padrão Prolog de Regras

A sintaxe padrão Prolog de regras por nós proposta é definida pela gramática mostrada na Figura 11.

Foi implementado um *script* Perl, ao qual denominamos Conversor de Regras, que tem como entrada um arquivo contendo regras de classificação no formato $\mathcal{PBM}$ ou $\mathcal{PBM}$ estendido pode ser convertido para cláusulas Prolog, segundo a gramática da Sintaxe Prolog mostrada na Figura 11. Deve ser ressaltado que a sintaxe padrão Prolog não contempla as regras induzidas por $\mathcal{OC}1$ (Murthy et al., 1994), as quais estão definidas através do símbolo não terminal `<linear combination>` na Figura 7.O código Perl dessa implementação encontra-se no Apêndice A na página 16.

Como mencionado anteriormente, diversos conjuntos de regras podem ser lidos simultaneamente. Assim, é necessário identificar cada um desses conjuntos que provêm de diferentes experiências. Isso é realizado de forma automática pelo Conversor de Regras através do identificador `<id>::=id_X` utilizado como primeiro argumento nas cláusulas unitárias Prolog `inducer/2`, `inputFile/2`, `dAte/3`, `evaluatedAs/3`, `namesFile/2`, `dataFile/2` e `rule/6` na Figura 11. O valor de `X` é gerado como o valor da seguinte Função Hash

$$\texttt{X} = (Sum1 + Sum2)\ mod\ 211$$

onde

$$Sum1 = \texttt{<month>} + \texttt{<day>} + \texttt{<year>}.$$

e

$$Sum2 = \texttt{<hour>} + \texttt{<minute>} + \texttt{<second>}.$$

Os valores de `<month>`,`<day>`,`<year>`,`<hour>`,`<minute>`,`<second>` são retirados da terceira linha do arquivo no formato $\mathcal{PBM}$ estendido que formam as informações que identificam quando foi realizada essa experiência.

Os símbolos terminais correspondentes a `<week_day>` e `<month>` são primeiramente mapeados em números inteiros da seguinte forma:

`<week_day>`	*valor*	`<month>`	*valor*
'Sun'	1	'Jan'	1
'Mon'	2	'Feb'	2
'Thu'	3	'Mar'	3
'Wed'	4	'Apr'	4
'Wen'	5	'May'	5
'Fri'	6	'Jun'	6
'Sat'	7	'Jul'	7
		'Aug'	8
		'Sep'	9
		'Oct'	10
		'Nov'	11
		'Dec'	12

para finalmente calcular o valor de `X`, Equação 5.

A Figura 12 mostra o arquivo da Figura 8 no formato $\mathcal{PBM}$ estendido na sintaxe padrão Prolog por nós porposta, após a execução do *script* Conversor de Regras.

Como pode ser observado o predicado `inducer/2`, informa o indutor utilizado nessa experiência para induzir as regras enquanto que `inputFile/2` informa o arquivo q contém as regras no formato $\mathcal{PBM}$.

O predicado `dAte/3` informa quanto as experiências foram realizadas e predicado `evaluatedAs/2` informa a forma na qual deverm ser avaliadas as regras (Baranauskas and Monard, 2000a).

Os predicados `nameFile/2` e `dataFile/2` informam os os arquivos `.names` e `.data` utilizados.

O predicado `rule/6` que representa as regras na sintaxe padrão Prolog, possui os seguintes argumentos:

<**arg-2**> número da regra.

<**arg-3**> lista de condições da regra. No formato padrão $\mathcal{PBM}$, o `<complex>` de uma regra é uma disjunção de conjunções de testes de atributos na forma:

```
<atributo><operador><valor>
```

o qual é convertido, na sintaxe padrão Prolog, para:

```
<condition>(<feature_name>,<feature_value>).
```

onde `<atributo>` é mapeado para `<feature_name>`, `<operador>` é mapeado, conforme a Tabela 2, para `<condition>` e `<valor>` é mapeado para `<feature_value>`.

<**arg-4**> classe da regra.

<**arg-5**> lista de infomações adicionais calculadas sobre exemplos que possuem valores conhecidos.

<**arg-6**> lista de infomações adicionais calculadas sobre exemplos que possuem valores desconhecidos.

Para a regra DEFAULT, os argumentos <**arg-3**>, <**arg-5**> e <**arg-6**> são sempre listas vazias.

Conforme mencionado anteriormente, além do formato $\mathcal{PBM}$ estendido, o Conversor de Regras também suporta regras no formato $\mathcal{PBM}$ (Prati et al., 2001b). Na Figura 13 é mostrado o arquivo da Figura 6 no formato $\mathcal{PBM}$ convertido para a sintaxe padrão Prolog por nós proposta. Deve ser observado que <**arg-5**> e <**arg-6**> são sempre listas vazias na conversão de regras no formato padrão $\mathcal{PBM}$ para a sintaxe padrão Prolog.

6 Considerações Finais

Neste trabalho foi apresentada proposta de uma sintaxe padrão Prolog equivalente ao formato padrão $\mathcal{PBM}$ de regras de classificação (Prati et al., 2001a) e de exemplos (Batista, 2001) utilizado no projeto Discover. Foi também realizada a implementação, na linguagem Perl, que

operador	condition
=	equal
>=	greaterORequal
<=	lessORequal
>	greater
<	less
in	pertence
<>	notEqual

Tabela 2: Valores Nominais de Operadores

faz a transformação de arquivos contendo informações na sintaxe $\mathcal{PBM}$ estendida para arquivos contendo essas mesmas informações na sintaxe padrão Prolog por nós proposta. Esses arquivos na sintaxe padrão prolog são utilizados como entrada pelos dois sistemas propostos por (Gomes, 2001) e (Bernardini, 2001).

A sintaxe padrão Prolog aqui proposta pretende ser robusta o suficiente para que outros sistemas, implementados no intuito de testar idéias relacionadas com aquisição automática de conhecimento, possam utilizá-la explorando algumas características de prototipagem rápida da linguagem Prolog.

Isso vai de encontro às diretrizes utilizadas no desenvolvimento do projeto Discover, no qual novas funcionalidades são integradas após terem sido exaustivamente validadas.

```
S ::= <header> | <rule>

<header> ::= <inducer>
             <input_file>
             <date>
             <evaluated_as>
             <name_file>
             <data_file>

<inducer> ::= inducer(<id>,<inducer_name>).

<input_file> ::= inputFile(<id>,<input_file_name>).

<date> ::= dAte(<id>,day(<week_day>,<month>,<day>,<year>),time(<hour>:<minute>:<second>)).

<evaluated_as> ::= evaluatedAs(<id>,<evaluated_form>).

<name_file> ::= nameFile(<id>,<file>).

<data_file> ::= dataFile(<id>,<file>).

<rule>::= rule(<id>,<rule number>,<list_of_conditions>,
      class(<class_feature>),<extension>).

<list_of_conditions>::= [<condition>(<feature_name>,<feature_value>),
             <condition>(<feature_name>,<feature_value>), ...].

<extension>::= [<know values>],[<unknow values>]

<condition>::=equal|notEqual|greater|less|lessORequal|greaterORequal|pertence.

<id>::=id_X.

<week_day>::='Sun'|'Mon'|'Thu'|'Wed'|'Wed'|'Fri'|'Sat'

<month>::='Jan'|'Feb'|'Mar'|'Apr'|'May'|'Jun'|'Jul'|'Aug'|'Sep'|'Oct'|'Nov'|'Dec'

<evaluated_form>::='UNORDERED'|'ORDERED'|'INTER-CLASS ORDERED'
```

Figura 11: Gramática da Sintaxe Padrão Prolog para Regras

```
inducer(id_129,'cn2-ordered').
inputFile(id_129,'voyage.cn2.ordered_output').
dAte(id_129,day('Thu','Jun',14,2001),time(20:54:28)).
evaluatedAs(id_129,'ORDERED').
nameFile(id_129,'voyage.names').
dataFile(id_129,'voyage.data').

rule(id_129,0001,[greater(humidity,83),equal(windy,no)],class(go),
           [0.250,0.083,0.417,0.250,12],[0.333,0.333,0.333,0.000,3]).
rule(id_129,0002,[equal(outlook,sunny),greater(humidity,78.50)],class(dont_go),
           [0.154,0.000,0.462,0.385,13],[0.500,0.000,0.500,0.000,2]).
rule(id_129,0003,[greater(humidity,83.50)],class(go),
           [0.231,0.000,0.538,0.231,13],[0.500,0.500,0.000,0.000,2]).
rule(id_129,0004,[greater(temperature,24.00)],class(dont_go),
           [0.200,0.000,0.467,0.333,15],[0.000,0.000,0.000,0.000,0]).
rule(id_129,0005,[],class(go),[],[]).
```

Figura 12: Exemplo de um Arquivo com um Conjunto de Regras na Sintaxe Prolog

```
inducer(id_129,'cn2-ordered').
inputFile(id_129,'voyage.cn2.ordered_output').
dAte(id_129,day('Thu','Jun',14,2001),time(20:54:28)).
evaluatedAs(id_129,'ORDERED').
nameFile(id_129,'voyage.names').
dataFile(id_129,'voyage.data').

rule(id_129,0001,[greater(humidity,83),equal(windy,no)],class(go),[],[]).
rule(id_129,0002,[equal(outlook,sunny),greater(humidity,78.50)],class(dont_go),[],[]).
rule(id_129,0003,[greater(humidity,83.50)],class(go),[],[]).
rule(id_129,0004,[greater(temperature,24.00)],class(dont_go),[],[]).
rule(id_129,0005,[],class(go),[],[]).
```

Figura 13: Exemplo de um Arquivo com um Conjunto de Regras na Sintaxe Prolog

A Conversor de Regras

```
#!/usr/local/bin/perl

#-------------------------------------------------------------------------------
# This script transform PBM Rule Standard Form to
# standard syntax Prolog #
# pre-condition:
#      A file contend rules in PBM Rule Standard Form #
# pos-condition:
#      A file contend rules in syntax Prolog
#
# 11-06-2001 modifications by Alan K. Gomes and Flavia C. Bernardini
# 07-01-2001 modifications by Flavia C. Bernardini
#
# arguments:
#   0 - input file
#   1 - output file
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#
# Procedure Declarations
#
#-------------------------------------------------------------------------------

#-------------------------------------------------------------------------------
#Procedure conv_cond: convert comparison simbols
#(=,>,<,>=,<=,in,<>) to  strings
#(equal,greater,less,lessORequal,greaterORequal,pertence,notEqual)
#-------------------------------------------------------------------------------

sub conv_cond { if($cond eq "="){$cond="equal";} elsif($cond eq
">="){$cond="greaterORequal";} elsif($cond eq
"<="){$cond="lessORequal";} elsif($cond eq ">"){$cond="greater";}
elsif($cond eq "<"){$cond="less";}
}
#-------------------------------------------------------------------------------
#Procedure ident_type: Identify the "line" type, what can beginning
# with IF, AND or OR, or THEN tokens
#-------------------------------------------------------------------------------

sub ident_type{
$i=0;
```

```
$ch=substr($words[$i],0,1); #take the first character in the line
if($ch=~/R/){ #next string expected IF
    #take the rule number
        ($lixo,$num_rule)= split(/R/,$words[$i]);
        while(($words[$i] ne "IF")&&($words[$i] ne
"DEFAULT")&&($words[$i] ne "Default"))
        {$i++;} #find IF or DEFAULT
        if($words[$i] eq "IF") {$tipo="I";}
        elsif(($words[$i] eq "DEFAULT")||($words[$i] eq "Default")){$tipo="C";}
        else{die("Problemas na sintaxe das regras");}
        $i++;
        }
else { #line with THEN CLASS or AND or OR
        while(($i <= @words) &&($tipo eq "")){
            if($words[$i] eq "AND"){$tipo="A";}
                elsif($words[$i] eq "OR"){$tipo="O";}
                elsif(($words[$i] eq "CLASS") or ($words[$i] eq "class"))
                 {$tipo="C";}#found THEN - the next string is CLASS
                elsif($i == @words){$tipo="UNK";}
                $i++;
            }
    }

}
#-------------------------------------------------------------------------------
#Procedure mount_rule: takes the rule of the input file and writes it in the
#output file in Prolog sintax
#-------------------------------------------------------------------------------

sub mount_rule{

if($tipo=~/I/||$tipo=~/A/||$tipo=~/O/){
    while(($i<=@words)&&(($words[$i] ne "=")&& ($words[$i] ne ">")&&
    ($words[$i] ne "<")&&
($words[$i] ne"<=")&&($words[$i] ne ">=")){
            #$letter=substr($words[$i],0,1);
            if($words[$i]=~/[A-Z].*/){$words[$i] =~ tr/[A-Z]/[a-z]/;}
                                # change uppercase to lowercase letters
            $feature_name.=$words[$i]; #take everything that comes
                        #before the comparison signal
             $i++;
            }

      $cond=$words[$i]; #take the comparison signal
      conv_cond; #convert in string the comparison signal and put it in $cond
    $i++;
```

```
        while($i<=@words){
           $feature_value.=$words[$i];
             $i++;
                }
        if($tipo=~/A/){$list_cond.=",";}
        if($tipo=~/O/){$list_cond.=";";}
        $list_cond.=$cond."(".$feature_name.",".$feature_value.")";
        $tipo= $feature_name= $feature_value=  "";
        }
if($tipo=~/C/){ #Found the class value
    while(($i<=@words)&&($words[$i] ne"=")){$i++;}
        $i++;
        $ch=substr($words[$i],0,1);
        until(($ch =~/\[/)||($i>@words)){
        $class.=$words[$i];
                $i++;
                $ch=substr($words[$i],0,1);
                }
    $eval="[],[]";
        if($i < @words){
           $list1=$list2="";
           until($ch=~/\?/){
        $list1.=$words[$i];
             $i++;
             $ch=substr($words[$i],0,1);
             }
            ($lixo,$sec_list)= split(/\?/,$words[$i]);
            $list2=$sec_list;
            $i++;
            until($i>@words){
        # delete "?"
            $list2.=$words[$i];
             $i++;
             }
        $eval=$list1.",".$list2;
                }
        print ARQ2
("rule(id_$id,$num_rule,[$list_cond],class($class),$eval).\n");
        $tipo= $list_cond = $class = $eval = "";
        }
$tipo=""; }
#-------------------------------------------------------------------------------
# Procedure to find the correspondent month number
#-------------------------------------------------------------------------------

sub equal_month{
```

```
if($words3[2] eq "Jan"){$month=1;}
elsif($words3[2] eq "Fev"){$month=2;}
elsif($words3[2] eq "Mar"){$month=3;}
elsif($words3[2] eq "Apr"){$month=4;}
elsif($words3[2] eq "May"){$month=5;}
elsif($words3[2] eq "Jun"){$month=6;}
elsif($words3[2] eq "Jul"){$month=7;}
elsif($words3[2] eq "Aug"){$month=8;}
elsif($words3[2] eq "Sep"){$month=9;}
elsif($words3[2] eq "Oct"){$month=10;}
elsif($words3[2] eq "Nov"){$month=11;}
elsif($words3[2] eq "Dec"){$month=12;}
}

#-------------------------------------------------------------------------------
#Mount the Header Output File
#-------------------------------------------------------------------------------

sub header{
$linha2 = <ARQ1>;
@words2 = split(/[\s+]/,$linha2);
$linha3 = <ARQ1>;
@words3 = split(/[\s+]/,$linha3);
$linha4 = <ARQ1>;
$linha = <ARQ1>;
@words5 = split(/[\s+]/,$linha);

($time1,$time2,$time3) = split(/:/,$words3[5]);
equal_month;
$id=($month+$words3[4]+$words3[6]+$time1+$time2+$time3)%211;#hashing function
print ARQ2
("inducer(id_$id,'$words2[1]').\ninputFile(id_$id,'$words2[6]').\n");
print ARQ2
("date(id_$id,day('$words3[1]','$words3[2]',$words3[3],$words3[5]),
  time($words3[4])).\n");
if($words5[1] eq "Evaluated"){
      $i=@words5;
      if($i > 4){print ARQ2 ("evaluatedAs(id_$id,'$words5[3] $words5[4]').\n");}
    else {print ARQ2 ("evaluatedAs(id_$id,'$words5[3]').\n")};
    $linha = <ARQ1>;
    @words6 = split(/[\s+]/,$linha);
    if($words6[0] eq "Names"){
        print ARQ2
        ("nameFile(id_$id,'$words6[2]').\ndataFile(id_$id,'$words6[5]').\n");
            }
    elsif($words6[0] eq "Data"){
```

```
        print ARQ2 ("dataFile(id_$id,'$words6[2]').\n");
            }
    $linha = <ARQ1>;
    }
print ARQ2 ("\n"); }
#-------------------------------------------------------------------------------
#
# Main Program
#
#-------------------------------------------------------------------------------

$arq1 = $ARGV[0];
$arq2 = $ARGV[1];

#Open the input and output files - if it can't be done, the program dies

if (!(open (ARQ1,$arq1)) || !(open(ARQ2,">$arq2")) ) {
        die ("Stdprolog.pl: Impossible to open files ".$arq1." and/or ".$arq2);
}
print("
##############################################################################
Transforming Rules in Rule Standard Format in $arq1 to Prolog Syntax
Rules in $arq2...

Attention: be aware of the pre-conditions that $arq1 must be in PBM rule
standard form as defined by Pratti in Tecnincal Report 137* or Tecnincal
Report 145*, ICMC-USP.
##############################################################################
");
$linha = <ARQ1>;
header;
while($linha ne ""){
    @words = split(/[\s+\t+]/,$linha);
    ident_type;
    mount_rule;
    $linha = <ARQ1>;
} print(" Finish. "); close(ARQ1); close(ARQ2);
#-------------------------------------------------------------------------------
# End of Program
#-------------------------------------------------------------------------------
```

Referências

Baranauskas, J. A. (2001). Extração automática de conhecimento por múltiplos indutores. Tese de Doutorado, ICMC-USP.

Baranauskas, J. A. and Monard, M. C. (2000a). Reviewing some machine learning concepts and methods. Technical Report 102, ICMC-USP. ftp://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 supervised symbolic machine learning inducers. Technical Report 103, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_102.ps.zip.

Batista, G. E. A. P. A. (1997). Um ambiente de avaliação de algoritmos de aprendizado de máquina utilizando exemplos. Dissertação de Mestrado, ICMC-USP.

Batista, G. E. A. P. A. (2000). Pré-processamento de dados em aprendizado de máquina supervisionado. Monografia do Exame de Qualificação de Doutorado, ICMC-USP.

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

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1/2):105–139.

Bernardini, F. C. (2001). Combinação de classificadores para melhorar o poder preditivo e descritivo de ensembles. Monografia do Exame de Qualificação de Mestrado, ICMC-USP.

Bratko, I. (1990). *Prolog Programming for Artificial Intelligence*. Addison-Wesley.

Caulkins, C. W. (2000). Aquisição de conhecimento utilizando aprendizado de máquina relacional. Dissertação de Mestrado, ICMC-USP.

Dietterich, T. (1997). *Machine Learning Research: Four Current Directions (draft)*. Oregon State University. http://www.cs.orst.edu/~tgd/ [date: Sep/2000].

Dietterich, T. (2000). *Ensemble Methods in Machine Learning (draft)*. Oregon State University. http://www.cs.orst.edu/~tgd/ [date: Oct/2000].

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 *Proceedings of the Second European Symp. LNAI*, volume 1510, pages 1–9.

Gomes, A. K. (2001). Medidas de avaliação de regras. Exame de Qualificação de Mestrado, ICMC-USP.

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

Horst, P. S. (1999). Avaliação do conhecimento adquirido por algoritmos de aprendizado de máquina utilizando exemplos. Dissertação de Mestrado, ICMC-USP.

Imamura, C. Y. (2001). Pré-processamento para extração de conhecimento de bases textuais. Dissertação de Mestrado, ICMC-USP.

Lavrac, N., Flach, P., and Zupan, B. (1999). Rule evaluation measures: a unifying view. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming. LNAI*, volume 1634, pages 74–185.

Lee, H. D. (2000). Seleção e construção de features relevantes para o aprendizado de máquina. Dissertação de Mestrado, ICMC-USP.

Martins, C. A. (2001). Clustering conceitual em aprendizado de máquina. Exame de Qualificação de Doutorado, ICMC-USP.

Milaré, C. R. (2000). Extração de conhecimento de redes neurais. Exame de Qualificação de Doutorado, ICMC-USP.

Monard, M. C., Caulkins, C. W., Baranauskas, J. A., Oliveira, R. B. T., and Rezende, S. O. (1999). Data preparation, reduction and prediction in the context of data mining: A case study with insurance policies. Technical Report 81, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_81.ps.zip`.

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`.

Nagai, W. A. (2000). Avaliação do conhecimentos extraído de problemas de regressão. Dissertação de Mestrado, ICMC-USP.

Pila, A. D. and Monard, M. C. (2001). Rough sets reducts as a filter approach for feature subset selection: An empirical comparison with wrapper and other filter approaches. Technical Report 134, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_134.ps.zip.

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.

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`.

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`.

Pugliesi, J. B. (2001). O pós-processamento em extração de conhecimento de bases de dados. Exame de Qualificação de Doutorado, ICMC-USP.

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

Westwood, D. (2000a). LPA-PROLOG for windows programming guide.

Westwood, D. (2000b). LPA-PROLOG technical reference.