UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2569

Sintaxe Padrão para Representar Regras de Regressão no Discover

Jaqueline Brigladori Pugliesi

Daniel Gomes Dosualdo

Solange Oliveira Rezende

Nº 193

RELATÓRIOS TÉCNICOS

São Carlos - SP
Abril/2003

Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Sintaxe Padrão para Representar Regras de Regressão no Discover

Jaqueline Brigladori Pugliesi
Daniel Gomes Dosualdo
Solange Oliveira Rezende

Nº 193

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Abril/2003

Sintaxe Padrão para Representar Regras de Regressão no Discover *

Jaqueline Brigladori Pugliesi
Daniel Gomes Dosualdo
Solange Oliveira Rezende

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: {jbpuglie, dosualdo, solange}@icmc.usp.br

Resumo

O processo de Mineração de Dados inicia-se com a escolha das fontes de dados a serem utilizadas e a definição dos objetivos. Uma parcela desses dados é selecionada, pré-processada e submetida aos métodos e ferramentas adequados com o objetivo de encontrar padrões e/ou modelos que representem o conhecimento obtido. Depois de extraídos, os padrões são pós-processados e o conhecimento adquirido é avaliado quanto à sua qualidade e/ou utilidade para determinar a viabilidade de sua utilização no apoio a algum processo de tomada de decisão.

Neste relatório técnico é apresentada a sintaxe padrão utilizada para converter os modelos gerados por algoritmos de regressão em um formato único, baseado nessa sintaxe. Essa conversão auxilia a avaliação de regras provenientes de diferentes algoritmos, além de permitir comparar os modelos por eles gerados. Além disso, são apresentadas duas classes que foram implementadas: uma que converte os regressores Cubist, RT e WEKA para o formato padrão de regras de regressão e a outra que realiza um tratamento semântico dessa regras.

Palavras-Chave: Mineração de Dados, Pós-Processamento, Regras de Regressão, Sintaxe Padrão, DISCOVER.

Abril 2003

*Trabalho realizado com auxílio da FAPESP.

Sumário

1 Introdução **1**

2 O Projeto Discover **2**

3 Gramática da Sintaxe Padrão **3**

4 Padronização dos Regressores **6**

5 Conversão dos Regressores para o Formato Padrão **8**
5.1 Cubist 9
5.2 RT 10
5.3 WEKA 12

6 Considerações Finais **18**

A Código Fonte da Classe de Conversão para a Sintaxe Padrão de Regras de Regressão **19**

B Código Fonte da Classe de Tratamento Semântico para Regras de Regressão **30**

Referências **38**

Lista de Figuras

1 Funcionalidades do Projeto DISCOVER 2
2 Produções da Gramática para Regras de Regressão 5
3 Regra de Regressão Gerada pelo Algoritmo Cubist 6
4 Regra de Regressão da Figura 3 na Sintaxe Padrão 6
5 Exemplo de um Conjunto de Regras Padronizadas 7
6 Regra de Regressão com Condições Desnecessárias sobre Atributos Numéricos 8
7 Regra de Regressão com Tratamento Semântico sobre Atributo Numérico . 8
8 Regra de Regressão com Condições Desnecessárias sobre Atributo Nominal 8
9 Regra de Regressão com Tratamento Semântico sobre Atributo Nominal . . 8
10 Módulo para Conversão das Saídas dos Algoritmos de Regressão Utilizados no DISCOVER 9
11 Algumas Regras Geradas pelo Cubist Utilizando o Conjunto de Dados Abalone 11
12 Regras da Figura 11 Convertidas para o Formato Padrão de Regras de Regressão 12
13 Conjunto de Regras Geradas pelo RT Utilizando o Conjunto de Dados mpg 13
14 Conjunto de Regras da Figura 13 Convertidas para o Formato Padrão de Regras de Regressão 14
15 *Model Tree* Gerada pelo $\mathcal{M}5$ Utilizando o Conjunto de Dados Housing . . 15
16 Algumas Regras da Figura 15 Convertidas para o Formato Padrão de Regras de Regressão 16
17 Regras da Figura 16 Convertidas para o Formato Padrão de Regras de Regressão com Tratamento Semântico 17

Lista de Tabelas

1 Algoritmos de Regressão Utilizados Neste Trabalho 9

[1]Este documento foi elaborado com o LaTeXe sua bibliografia gerada automaticamente pelo BibTeX.

1 Introdução

A extração de padrões e modelos de grandes Bases de Dados geralmente é feita utilizando metodologias, técnicas e algoritmos de Aprendizado de Máquina. Além disso, outros aspectos devem ser considerados, como o estudo de métodos para avaliação da precisão, qualidade e compreensibilidade do conhecimento obtido de uma ou mais amostras submetidas a algoritmos de aprendizado.

O processo de Mineração de Dados é composto por uma série de etapas, sendo que as etapas de pré e pós-processamento consomem a maior parte do esforço despendido ao longo do processo (Rezende, Pugliesi, Melanda, & Paula 2003).

Na fase de pré-processamento, uma série de atividades são necessárias, como adequações de formato, tratamento de valores ausentes e transformações. Em especial, a falta de um formato único para os conjuntos de exemplos utilizados causa um grande transtorno às comunidades de Mineração de Dados e Aprendizado de Máquina. Por exemplo, com relação aos métodos de regressão e classificação simbólicos, cada algoritmo utilizado para induzir uma hipótese possui um formato diferente para seus dados de entrada. Dessa forma, cada vez que é preciso executar algum desses algoritmos, é necessário converter o conjunto de dados para o formato de entrada do algoritmo, além de efetuar outras operações, como limpeza nos dados, tratamento de valores desconhecidos e outras inconsistências.

Já na fase de pós-processamento (ou avaliação do conhecimento), o panorama não é muito diferente. Ainda com relação ao aprendizado simbólico, são distintas as hipóteses induzidas pelos algoritmos. Em conseqüência, qualquer operação que necessite ser feita sobre a hipótese induzida (como cálculo de medidas e construção de tabela de contingência) precisa ser implementada diversas vezes, uma para cada formato de hipótese diferente.

No mundo real, problemas de classificação são mais encontrados do que problemas de regressão, o que explica a maior atenção dada a classificação. Porém, muitos problemas importantes são de regressão. Além do fato de que problemas de regressão são relevantes por si próprios, outra razão da necessidade em focar em regressão deve-se ao fato de que métodos de regressão podem ser usados para resolver problemas de classificação. Por exemplo, redes neurais são muitas vezes aplicadas a problemas de classificação (Weiss & Indurkhya 1995).

A avaliação do conhecimento leva em consideração algumas particularidades dos diferentes tipos de problema. Regras de regressão, apesar de serem semelhantes às regras de classificação, não predizem um valor categórico definido e sim, um valor numérico, envolvendo um grau de incerteza quanto à classe predita (Cheng 1998).

Em regressão, a cada exemplo é atribuído um valor numérico ao invés de um conjunto finito de valores discretos. Fornecendo um conjunto de atributos de entrada $x_1, ..., x_n$ e o atributo-meta y, tenta-se fazer uma aproximação da função mostrada na Equação 1.

$$y = f(x_1, ..., x_n) \tag{1}$$

O objetivo deste relatório técnico é apresentar uma sintaxe padrão para representar regras de regressão em um ambiente que está sendo desenvolvido em nosso laboratório, denominado DISCOVER, bem como algumas classes que convertem a saída dos algoritmos de regressão Cubist, WEKA e RT para um formato baseado nessa sintaxe.

2 O Projeto Discover

Em nosso laboratório de pesquisa — LABIC (Laboratório de Inteligência Computacional)[1] — sempre foram realizadas implementações nas áreas de Aprendizado de Máquina e Mineração de Dados por parte de professores e alunos, conforme as necessidades de cada um. Porém, devido à falta de documentação dos programas já implementados, ou ainda porque seus autores se desligavam do laboratório, normalmente esses implementações não eram reaproveitadas. Dessa forma, um mesmo programa era implementado diversas vezes por diferentes pessoas.

Esses fatores impulsionaram então pesquisadores do LABIC a criarem um projeto que envolvesse membros do laboratório. Esse projeto, que recebeu o nome de DISCOVER, deveria fornecer um ambiente para apoiar as etapas do processo de descoberta do conhecimento, oferecendo funcionalidades voltadas para Aprendizado de Máquina, Mineração de Dados e Textos, como apresentado na Figura 1.

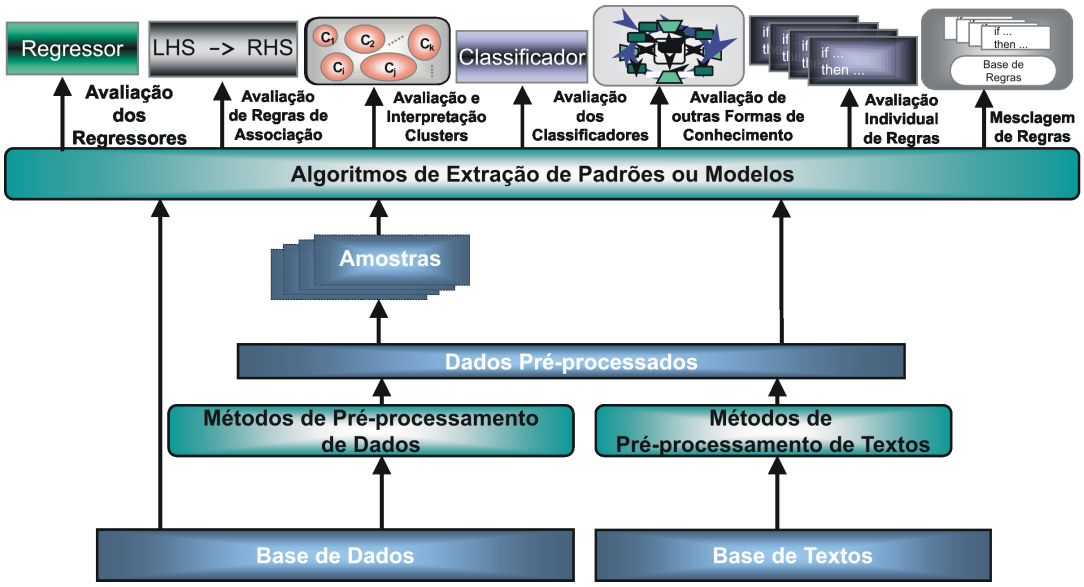

Figura 1: Funcionalidades do Projeto DISCOVER

Em princípio, o projeto DISCOVER consistiria apenas de um repositório de *scripts*[2], que seriam implementados na linguagem PERL (Till 1996) — *Practical Extraction and Report Language* — e posteriormente reutilizados de forma a facilitar a execução de experimentos.

[1] http://labic.icmc.usp.br/

[2] Scripts são pequenos programas que realizam tarefas atômicas.

Portanto, a priori, o projeto DISCOVER consistiria de um conjunto de *scripts* independentes que poderiam ser combinados conforme as necessidades dos usuários, e reunidos em um repositório compartilhado entre esses usuários. Assim, os usuários poderiam combinar esses *scripts* para executar experimentos mais complexos.

Posteriormente, surgiu a proposta de se criar um ambiente integrado, em que os *scripts* seriam substituídos por bibliotecas de classes e essas classes empacotadas como componentes, com a composição deles sendo feita por meio de uma interface gráfica. Dessa forma, os programas desenvolvidos pelos diversos pesquisadores são os componentes e o DISCOVER um ambiente computacional que possibilita a construção de experimentos através da composição desses componentes.

Essa nova abordagem facilitou a utilização do ambiente e permitiu obter um melhor desempenho do mesmo. Isso porque não é possível compartilhar memória através da utilização de *scripts*, causando uma sobrecarga com operações de leitura e escrita. No entanto, para tornar o DISCOVER um ambiente integrado, sua implementação tornou-se bem mais complexa. Surgiu a necessidade de uma maior comunicação entre os participantes e de interação entre os diferentes componentes. Com relação à arquitetura do ambiente, em (Prati 2003) é proposto um *framework* para a integração dos componentes do ambiente DISCOVER baseado em *software patterns*, no qual os componentes são integrados por meio de uma linguagem baseada em XML. Além disso, a interface gráfica responsável pela composição dos componentes está sendo implementada de acordo com a arquitetura proposta para o ambiente. Essa interface é apresentada em (Geromini 2002).

De maneira geral, o DISCOVER pode ser entendido como um conjunto de métodos aplicados sobre os dados ou sobre o conhecimento extraído a partir dos dados. Dessa forma, é muito importante que o DISCOVER ofereça uma base sólida para manipular dados e conhecimento. Essa base é composta por sintaxes padrões utilizadas para representar dados e conhecimento, e por bibliotecas que oferecem um conjunto de funcionalidades básicas de manipulação de dados e conhecimento. O pré-processamento e a sintaxe padrão para representação de dados DSX (*Discover Dataset Sintax*) são apresentados em (Batista 2003). Com relação à forma de representação do conhecimento, destacam-se trabalhos com regras de decisão (Prati, Baranauskas, & Monard 2001b; Prati, Baranauskas, & Monard 2001a), regras de regressão (Pugliesi 2001; Dosualdo 2002), regras de associação (Melanda 2000) e agrupamentos (Martins 2001).

3 Gramática da Sintaxe Padrão

Os métodos de regressão simbólicos geralmente induzem hipóteses no formato de regras e árvores. As regras são diferente das árvores, e mesmo as árvores geradas por diferentes algoritmos são distintas umas das outras: elas diferem no modo como é particionado o conjunto de dados, na escolha dos atributos e valores utilizados nos nós internos da árvore, no modo como predizem o valor do atributo-meta em seus nós-folha (por exemplo, uma *Regression Tree* calcula a média dos exemplos e uma *Model Tree* possui uma equação linear), nos mecanismos de poda e no formato em que a árvore pode ser visualizada, fator esse que nos interessa.

Devido aos diferentes formatos de representação das hipóteses geradas por esses algoritmos, qualquer medida ou estatística que precise ser calculada sobre as hipóteses tem de ser implementada várias vezes, uma para cada formato de hipótese. Dessa forma, é definida neste relatório técnico uma sintaxe padrão para representação de regras de regressão para o ambiente DISCOVER. Baseado nessa sintaxe, padroniza-se os regressores convertendo-os para um formato único em que as regras são não ordenadas. Além de facilitar a implementação de medidas e cálculos a serem efetuados sobre as hipóteses, isso permite comparar os regressores gerados pelos diferentes algoritmos.

Para possibilitar a integração e avaliação de conhecimentos extraídos por diferentes algoritmos houve a necessidade de se definir uma sintaxe padrão para regras de regressão, e para tanto foi especificada a gramática a ser utilizada na conversão das regras geradas pelos algoritmos de regressão para essa sintaxe. A definição de tal gramática levou em consideração a padronização definida pelo grupo do LABIC que trabalha com Extração de Conhecimento de Dados para o ambiente Discover.

A gramática, que define a sintaxe padrão para regras de regressão, é definida como uma quádrupla $G = (V_n, V_t, P, S)$:

S: <regra regressão>

V_n: { <regra regressão>, <nro regra>, <regra_r>, <condições>, <atrib_meta>, <regressão>, <fator>, <termo>, <comparação>, <expressão>, <atributo>, <operador relacional>, <valor numérico>, <valor categórico>, <conjunto>, <identificador>, <soma>, <constante>, <sinal>, <equação linear>, <dígitos>, <dígito>, <letra> }

V_t: { IF, THEN, OR, AND, IN, {, }, <=, >=, =, <, >, <>, +-, [,], +, -, ., ,, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, _ }

P: é descrito usando a notação EBNF e representado na Figura 2.

$[\alpha] = \alpha|\lambda$

$\{\alpha\}$ = repetição da cadeia α zero ou mais vezes

$\alpha|\beta = \alpha$ ou β devem ser escolhidos

Não terminais aparecem entre < e >, e terminais aparecem em negrito.

Por exemplo, a regra apresentada na Figura 3 foi gerada pelo algoritmo de regressão Cubist e armazenada, juntamente com as demais, em um arquivo-texto.

As regras geradas são convertidas para a sintaxe padrão de regras de regressão utilizando a gramática de conversão apresentada anteriormente. Um exemplo da regra apresentada na Figura 3 convertida para essa sintaxe pode ser visto na Figura 4. Todas as regras convertidas também são armazenadas em um outro arquivo-texto.

```
<regra regressão> ::= <nro regra> <regra_r>
<regra_r> ::= [ IF <condições> THEN ] <atrib_meta> = <regressão>
<condições> ::= <fator> | <fator> OR <condições>
<fator> ::= <termo> | <termo> AND <fator>
<termo> ::= <comparação> | <expressão>
<comparação> ::= <atributo> <operador relacional> <valor numérico>
                 | <atributo> <operador relacional> <valor categórico>
                 | <atributo> IN { <conjunto> }
<atributo> ::= <identificador>
<expressão> ::= <soma> <operador relacional> <valor numérico>
<soma> ::= <constante> <atributo numérico> |
           <constante> <atributo numérico> <sinal> <soma>
<operador relacional> ::= <= | >= | = | < | > | <>
<regressão> ::= <equação linear> | <valor numérico> |
                <valor numérico> +- <valor numérico>
<equação linear> ::= <constante> <sinal> <soma>
<conjunto> ::= <valor categórico> | <valor categórico> , <conjunto>
<nro regra> ::= [ R <dígito> <dígito> <dígito> <dígito> ]
<sinal> ::= + | -
<constante> ::= <valor numérico>
<valor numérico> ::= [ <sinal> ] <dígitos> [ . <dígitos> ]
         | [ <sinal> ] <dígitos> [ . <dígitos> ] e <sinal> <dígitos>
<dígitos> ::= <dígito> { <dígito> }
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<valor categórico> ::= <identificador>
<identificador> ::= <letra> | <dígito> { <letra> | <dígito> }
<letra> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
          | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D
          | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S
          | T | U | V | W | X | Y | Z | _
```

Figura 2: Produções da Gramática para Regras de Regressão

Observe que as informações fornecidas pela regra original (por exemplo: [898 cases, mean 7.8, range 1 to 21, est err 1.2]) não são armazenadas na sintaxe padrão, uma vez que essas medidas normalmente são distintas para cada algoritmo de regressão. As in-

```
Rule 1: [898 cases, mean 7.8, range 1 to 21, est err 1.2]
  if
    sex = I
  then
    rings = 2.7 + 15.3 shell_weight - 7.1 shucked_weight + 8.8 diameter
            + 15 height + 0.2 whole_weight
```

Figura 3: Regra de Regressão Gerada pelo Algoritmo Cubist

```
[R0001]  IF sex IN {I}
         THEN rings = 2.7 +15.3 shell_weight -7.1 shucked_weight
                      +8.8 diameter +15 height +0.2 whole_weight
```

Figura 4: Regra de Regressão da Figura 3 na Sintaxe Padrão

formações necessárias à avaliação de regras são calculadas de forma padronizada e independe do algoritmo que as geraram.

4 Padronização dos Regressores

O processo de padronização dos regressores funciona da seguinte forma: primeiramente, o conjunto de dados é submetido a algum dos possíveis algoritmos de regressão, e a saída por eles gerada é salva com a extensão `.out`, ou seja, `<dataset>.out`. Em seguida, utiliza-se a classe, por nós implementada, denominada `RegConvert` (cujo código é apresentado no Apêndice A), que contém os métodos que convertem a saída de cada algoritmo para o formato padrão de regras de regressão, para gerar então um arquivo com a extensão `.rules`, que contém o regressor no formato padrão de regras de regressão definido para o ambiente DISCOVER.

Independentemente do algoritmo de regressão utilizado para induzir a hipótese, o conjunto de regras padronizado obtido a partir dessa hipótese possui o formato apresentado na Figura 5.

Muitas vezes, depois que os regressores são convertidos para o formato padrão de regras de regressão, as regras que formam esse regressor podem apresentar condições desnecessárias, principalmente quando o conjunto de regras é convertido a partir de uma árvore. Isso acontece porque em muitos casos um mesmo atributo (numérico ou nominal) é utilizado para particionar o conjunto de dados em diferentes níveis de uma árvore assumindo diferentes valores, o que faz com que uma condição sobreponha a outra, tanto no caso de atributos numéricos quanto nominais. Nesse sentido, foi desenvolvido um programa para realizar um tratamento semântico ao arquivo `.rules`, que contém as regras no formato padrão com o objetivo de excluir as condições desnecessárias de cada regra. Esse tratamento apresenta duas vantagens:

```
[R0001] IF <condition_11>
               AND <condition_12>
        THEN <conclusion_1>

[R0002] IF <condition_21>
               AND <condition_22>
               AND <condition_23>
        THEN <conclusion_2>

[R0003] IF <condition_3>
        THEN <conclusion_3>

[R0004] IF <condition_41>
               AND <condition_42>
               AND <condition_43>
               AND <condition_44>
        THEN <conclusion_4>
```

Figura 5: Exemplo de um Conjunto de Regras Padronizadas

1. melhora a compreensibilidade da regra, visto que quanto maior o número de condições, mais difícil o entendimento da regra. É importante ressaltar que a compreensibilidade é um fator muito importante quando se trabalha com aprendizado simbólico;

2. aumenta o desempenho dos programas que utilizam essas regras como entrada, já que muitos testes desnecessários são evitados. Isso é extremamente importante quando o arquivo de regras é extenso e o número de exemplos de teste é grande.

Para realizar o tratamento semântico utiliza-se a classe `RegSemantic` (código apresentado no Apêndice B) para gerar um arquivo com a extensão `.Srules` (`<dataset>.Srules`), contendo as regras semanticamente tratadas.

O problema com os atributos numéricos acontece quando existem várias condições em uma regra sobre um mesmo atributo numérico com diferentes valores sobre um mesmo operador ($<, <=, >, >=$), fazendo com que tais condições se sobreponham. A solução nesse caso consiste em encontrar o intervalo correto para o atributo numérico na regra em que ele aparece em várias condições desse tipo.

Por exemplo, seja um conjunto de dados com x_1, x_2 e x_3 atributos numéricos, e uma regra de regressão como apresentada na Figura 6.

Logo, percebe-se que as duas primeiras condições da regra não são necessárias, uma vez que a terceira condição ($x_1 <= 0,08$) sobrepõe as duas primeiras. Dessa forma, após a aplicação do tratamento semântico, obtém-se a regra de regressão da Figura 7.

Com os atributos nominais, o problema ocorre quando existe mais de uma condição sobre um mesmo atributo nominal em uma determinada regra. A solução aqui consiste em encontrar a intersecção dos valores nominais entre todas as condições que aparecem naquela regra para o atributo nominal em questão.

IF $x_1 <= 0,17$
AND $x_1 <= 0,10$
AND $x_1 <= 0,08$
AND $x_1 > 0,03$
AND $x_2 > 3,84$
THEN $y = 5,02 + 0,4x_1 + 1,19x_3$

Figura 6: Regra de Regressão com Condições Desnecessárias sobre Atributos Numéricos

IF $x_1 <= 0,08$
AND $x_1 > 0,03$
AND $x_2 > 3,84$
THEN $y = 5,02 + 0,4x_1 + 1,19x_3$

Figura 7: Regra de Regressão com Tratamento Semântico sobre Atributo Numérico

Seja um conjunto de dados sobre automóveis com x_1 e x_2 atributos numéricos, e x_3 um atributo nominal assumindo os valores *gm, ford, fiat, wolksvagem, toyota, peugeot, audi* e *renaux*. Agora considere a regra de regressão apresentada na Figura 8.

IF $x_1 < 8,75$
AND x_3 IN {*gm, ford, fiat, wolksvagem, toyota, peugeot, renaux*}
AND x_3 IN {*toyota, peugeot, renaux*}
AND x_3 IN {*peugeot, renaux*}
THEN $y = 11,2 - 1,57x_1 + 0,44x_2$

Figura 8: Regra de Regressão com Condições Desnecessárias sobre Atributo Nominal

Analisando a regra, nota-se que para um exemplo de teste ser coberto por essa regra, o atributo nominal x_3 deve assumir os valores *toyota* ou *peugeot*. Assim sendo, aplicando o tratamento semântico, obtém-se a regra da Figura 9. Nota-se claramente a maior compreensibilidade da regra apresentada nessa tabela, comparando-se com a regra antes da aplicação do tratamento semântico.

IF $x_1 < 8,75$
AND x_3 IN {*peugeot, renaux*}
THEN $y = 11,2 - 1,57x_1 + 0,44x_2$

Figura 9: Regra de Regressão com Tratamento Semântico sobre Atributo Nominal

5 Conversão dos Regressores para o Formato Padrão

Para a conversão das regras e árvores de regressão para a sintaxe padrão de regras de regressão foram implementados conversores na linguagem PERL, sendo que tanto as regras de regressão quanto árvores de regressão, independente do formato original, são convertidas para a mesma sintaxe de regras de regressão.

Os algoritmos de regressão simbólicos utilizados no DISCOVER, juntamente com a forma como representam suas hipóteses, são apresentados na Tabela 1.

Algoritmo	Forma de Representação
Cubist	Conjunto de regras de regressão
$\mathcal{M}5$ WEKA	*Model Tree* ou *Regression Tree*
RT	*Regression Tree* com desvios-padrões
RETIS RT	*Model Tree*
CART RT	*Regression Tree* com desvios-padrões

Tabela 1: Algoritmos de Regressão Utilizados Neste Trabalho

Um esquema sobre os conversores de dados de algoritmos de regressão pode ser visualizado na Figura 10.

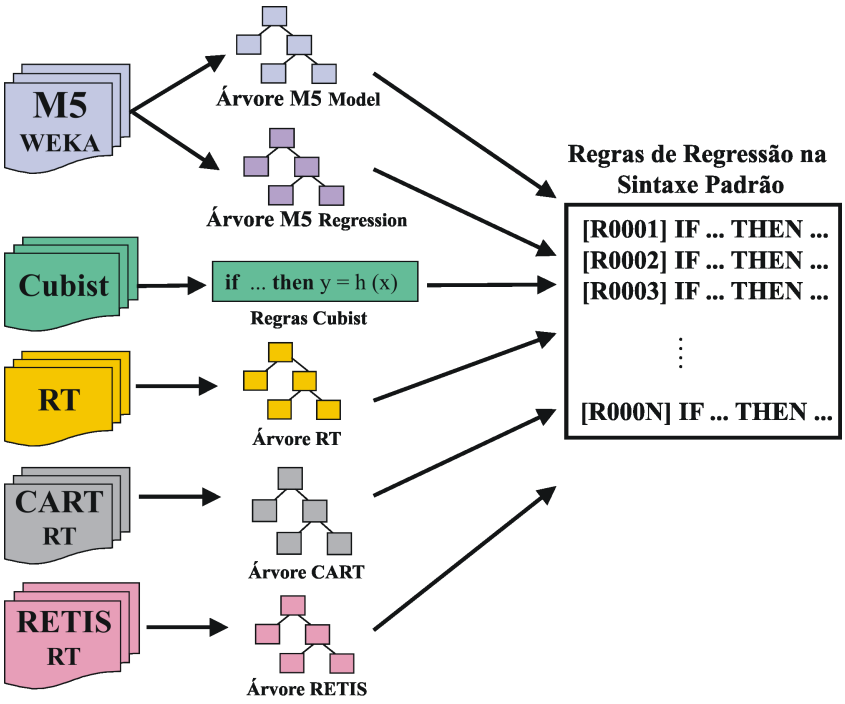

Figura 10: Módulo para Conversão das Saídas dos Algoritmos de Regressão Utilizados no Discover

Nesta seção é apresentada uma descrição dos sistemas e algoritmos de regressão utilizados neste relatório, para os quais foram implementados conversores, bem como exemplos de regras de regressão antes e depois de serem convertidas para a sintaxe padrão. Os sistemas RT e WEKA são *freeware*, e com relação ao Cubist, o LABIC possui uma versão do mesmo.

5.1 Cubist

O Cubist (Rulequest-Research 2000) é uma ferramenta utilizada para a geração de modelos preditivos numéricos baseados em regras a partir de um conjunto de dados fornecido. Esses modelos são utilizados na predição de valores numéricos, ou seja, para problemas de regressão. Cada caso fornecido ao Cubist possui um atributo-meta e os demais atributos

utilizados para predizer o valor desse atributo-meta. O Cubist gera regras não ordenadas, dispostas em ordem crescente do valor médio dos exemplos preditos.

A tarefa do Cubist é tentar estimar o valor do atributo-meta de um novo exemplo em função dos outros atributos. Para isso, ele constrói um modelo contendo uma ou mais regras, na qual cada regra é uma conjunção de condições associadas com uma expressão linear. Portanto, os modelos gerados pelo Cubist são modelos lineares *piecewise*. Porém, o Cubist pode também construir modelos múltiplos e pode combinar modelos baseados em regras com modelos baseados em exemplos.

Depois de executados os algoritmos de regressão, converte-se os diferentes modelos gerados para a sintaxe padrão de regras de regressão. Na Figura 11 são apresentadas algumas regras geradas pelo algoritmo Cubist e na Figura 12 pode-se visualizar essas mesmas regras convertidas para o formato padrão de regras de regressão.

5.2 RT

O RT (*Regression Trees*) foi desenvolvido por Luis Fernando Raínho Alves Torgo, do Departamento de Ciências de Computadores da Faculdade de Ciências da Universidade do Porto (Torgo 2001). Esse algoritmo permite obter modelos de regressão em diferentes tipos de representação baseados em um conjunto de exemplos fornecido. Alguns tipos de modelos que podem ser obtidos são:

- modelo baseado em árvores de regressão, em que os nós-folhas são rotulados com a média dos valores neles mapeados;
- modelo paramétrico global;
- modelo local, através do algoritmo *K-Nearest Neighbour*.

O RT permite ainda, emular os algoritmos de regressão RETIS e CART, descritos a seguir.

RETIS O RETIS (*Regression Tree Induction System*) é um sistema utilizado para induzir árvores de regressão desenvolvido por Aram Karalic. As árvores de regressão são utilizadas para modelar uma relação linear *piecewise* entre atributos nominais ou contínuos e um atributo-meta contínuo (Karalic 1995). Portanto, esse algoritmo gera uma *Model Tree*.

CART O algoritmo CART (*Classification And Regression Trees*) foi desenvolvido por Breiman, Friedman, Olshen e Stone (Breiman, Friedman, Olshen, & Stone 1984).

O algoritmo CART permite a construção de árvores de decisão e árvores de regressão (*Regression Trees*) realizando um particionando recursivo binário do conjunto de dados e associando a cada nó-folha da árvore uma classe, no caso das árvores de decisão, ou um valor contínuo, no caso das árvores de regressão.

```
Cubist [Release 1.10]  Mon Mar 31 16:29:56 2003
---------------------

    Options:
        Application 'abalone'

    Target attribute 'rings'

Read 3762 cases (9 attributes) from abalone.data

Model:

  Rule 1: [42 cases, mean 5.7, range 3 to 10, est err 1.0]

    if
        sex in {M, F}
        diameter <= 0.23
    then
        rings = 3.8 + 67.5 shell_weight + 34.2 diameter - 25.8 length
                - 13.2 shucked_weight + 5.1 whole_weight - 4.6 viscera_weight
                + 4 height

  Rule 2: [809 cases, mean 6.9, range 1 to 16, est err 1.0]

    if
        sex = I
        shell_weight <= 0.1535
    then
        rings = 2.7 + 15.3 shell_weight - 6.5 shucked_weight + 8.8 diameter
                + 14 height

  Rule 3: [111 cases, mean 7.6, range 5 to 10, est err 0.8]

    if
        shucked_weight > 0.233
        shell_weight <= 0.1535
    then
        rings = 6.6 - 7.4 shucked_weight + 2.1 whole_weight + 6.2 shell_weight
                + 2.9 diameter

  Rule 4: [277 cases, mean 9.2, range 5 to 18, est err 1.5]

    if
        sex in {M, F}
        diameter > 0.23
        shucked_weight <= 0.233
        shell_weight <= 0.1535
    then
        rings = 6.9 + 31.6 whole_weight - 53.3 shucked_weight
                - 25 viscera_weight + 3.2 shell_weight + 1.7 diameter
```

Figura 11: Algumas Regras Geradas pelo Cubist Utilizando o Conjunto de Dados Abalone

```
Standard Regression Rules Conversor  Copyright (c) Daniel Gomes Dosualdo
Inducer: cubist              Input File:  abalone.tmp.rules
Date: Mon Mar 31 16:29:59 2003       Output File: abalone.tmp.Rules


[R0001] IF diameter <= 0.23
        AND sex IN {M, F}
    THEN rings = 3.8 +67.5 shell_weight +34.2 diameter -25.8 length
                      -13.2 shucked_weight +5.1 whole_weight
                      -4.6 viscera_weight +4 height

[R0002] IF shell_weight <= 0.1535
        AND sex IN {I}
    THEN rings = 2.7 +15.3 shell_weight -6.5 shucked_weight
                     +8.8 diameter +14 height

[R0003] IF shucked_weight > 0.233
        AND shell_weight <= 0.1535
    THEN rings = 6.6 -7.4 shucked_weight +2.1 whole_weight
                     +6.2 shell_weight +2.9 diameter

[R0004] IF shucked_weight <= 0.233
        AND shell_weight <= 0.1535
        AND diameter > 0.23
        AND sex IN {M, F}
    THEN rings = 6.9 +31.6 whole_weight -53.3 shucked_weight
                     -25 viscera_weight +3.2 shell_weight +1.7 diameter
```

Figura 12: Regras da Figura 11 Convertidas para o Formato Padrão de Regras de Regressão

Na Figura 13 é apresentada a árvore de regressão gerada pelo RT, e as regras geradas na sintaxe padrão de regras de regressão a partir dessa árvore podem ser visualizadas na Figura 14.

5.3 WEKA

O ambiente WEKA (*Waikato Environment for Knowledge Analysis*) disponibiliza vários algoritmos de aprendizado de máquina implementados na linguagem Java, desenvolvido na Universidade de Waikato na Nova Zelândia[3].

No WEKA, cada implementação de um algoritmo de aprendizado é representada por uma classe. A linguagem Java permite que as classes sejam organizadas em pacotes, que são simplesmente diretórios contendo uma coleção de classes relacionadas. Isso é bastante útil porque certos algoritmos compartilham muitas funcionalidades, e dessa forma, várias classes em um pacote podem ser utilizadas por mais de um algoritmo (Witten & Frank 1999).

[3]http://www.waikato.ac.nz/

```
*************************************************************
RT version 4.1.0 - RESEARCH VERSION
*************************************************************
(c) Copyright Luis Torgo, All Rights Reserved.
                                                        05-Jul-01
Problem with 8 attributes.
360 examples were loaded.

SAVING THE UNPRUNED TREE ON FILE mpg.data.tree

PRUNING THE TREE...DONE.
SAVING THE PRUNED TREE ON FILE mpg.data.pruned

RESULTING TREE :
 cylinders In { 8  6  3 }       (mpg =  22.80)
 True   (181 of 360)
        displacement <= 284.50  (mpg = 16.85)
        True    (84 of 181)
               LEAF ::  mpg =  19.47 +/- 0.634 [15.00..32.70]
        False   (97 of 181)
               LEAF ::  mpg =  14.58 +/- 0.472 [9.00..23.00]
 False  (179 of 360)
        horsepower <= 70.50     (mpg = 28.82)
        True    (66 of 179)
               LEAF ::  mpg =  33.13 +/- 1.304 [23.00..46.60]
        False   (113 of 179)
               model In { 70  71  72  73  74  75  76  77  78 }  (mpg =  26.30)
               True     (83 of 113)
                      LEAF ::  mpg =  24.81 +/- 0.705 [18.00..33.50]
               False    (30 of 113)
                      LEAF ::  mpg =  30.44 +/- 1.533 [22.30..41.50]


NUMBER OF NODES 9 [5 are leaves]
```

Figura 13: Conjunto de Regras Geradas pelo RT Utilizando o Conjunto de Dados mpg

O pacote **weka.classifiers** contém implementações dos algoritmos de classificação e predição numérica do WEKA. A classe **weka.classifiers.m5** é a classe que implementa o algoritmo $\mathcal{M}5$. O $\mathcal{M}5$ é um algoritmo desenvolvido por Quinlan (Quinlan 1992) para tratar atributos e classes contínuas. Estruturalmente, $\mathcal{M}5$ pertence à família TDIDT (*Top Down Induction Decision Trees*), mas com funções de regressão linear nos nós-folhas, ao invés de um valor categórico predizendo a classe. O algoritmo $\mathcal{M}5$ foi baseado no trabalho inicial de Breiman com o algoritmo CART. $\mathcal{M}5$ possui algumas vantagens frente a outras técnicas que manipulam atributos contínuos, por exemplo, a Regressão Linear clássica que impõe uma relação estritamente linear entre os dados. O algoritmo $\mathcal{M}5$ possui uma fase de particionamento, que divide o conjunto de dados; uma fase de poda, para reduzir o número de nós da árvore obtida; e uma fase adicional denominada *smoothing*, que tem como objetivo reduzir a grande diferença dos valores preditos entre os nós-folhas (Wang & Witten 1997).

```
Standard Regression Rules Conversor      Copyright (c) Daniel Gomes Dosualdo
Inducer: rt                              Input File:  mpg.out
Date: Mon Mar 31 22:31:12 2003           Output File: mpg.rules


[R0001]  IF cylinders IN {8, 6, 3}
                AND displacement <= 284.50
         THEN mpg = 19.47 +/- 0.634

[R0002]  IF cylinders IN {8, 6, 3}
                AND displacement > 284.50
         THEN mpg = 14.58 +/- 0.472

[R0003]  IF cylinders IN {4, 5}
                AND horsepower <= 70.50
         THEN mpg = 33.13 +/- 1.304

[R0004]  IF cylinders IN {4, 5}
                AND horsepower > 70.50
                AND model IN {70, 71, 72, 73, 74, 75, 76, 77, 78}
         THEN mpg = 24.81 +/- 0.705

[R0005]  IF cylinders IN {4, 5}
                AND horsepower > 70.50
                AND model IN {79, 80, 81, 82}
         THEN mpg = 30.44 +/- 1.533
```

Figura 14: Conjunto de Regras da Figura 13 Convertidas para o Formato Padrão de Regras de Regressão

Da biblioteca WEKA foi utilizado um algoritmo modificado do $\mathcal{M}5$, o $\mathcal{M}5$Prime (Witten & Frank 1999). Além de utilizar $\mathcal{M}5$Prime para gerar *Model Trees*, pode-se utilizá-lo para gerar *Regression Trees*, isto é, árvores de decisão que possuem nos nós folhas valores reais preditos a partir das funções lineares da *Model Tree* gerada.

Na Figura 15 é apresentada a *Model Tree* gerada pelo algoritmo $\mathcal{M}5$ do WEKA. Na Figura 16 pode-se visualizar algumas regras que foram convertidas a partir dessa árvore para o formato padrão de regras de regressão. Já na Figura 17 são mostradas as regras com tratamento semântico. Observe que as regras [R0005], [R0006], [R0009] e [R0010] dessa figura são mais compreensíveis que as correspondentes na Figura 16 em função do tratamento semântico realizado.

```
Pruned training model tree:

RM <= 6.68 :
|   DIS <= 1.98 :
|   |   LSTAT <= 14.6 : LM1 (15/81.1%)
|   |   LSTAT >  14.6 : LM2 (57/33.6%)
|   DIS >  1.98 : LM3 (275/34.8%)
RM >  6.68 :
|   RM <= 7.44 :
|   |   INDUS <= 7.82 :
|   |   |   RM <= 7.12 :
|   |   |   |   LSTAT <= 5.2 : LM4 (20/32.6%)
|   |   |   |   LSTAT >  5.2 : LM5 (20/45.9%)
|   |   |   RM >  7.12 : LM6 (22/25.9%)
|   |   INDUS >  7.82 :
|   |   |   NOX <= 0.589 : LM7 (8/18%)
|   |   |   NOX >  0.589 :
|   |   |   |   NOX <= 0.651 : LM8 (4/14.7%)
|   |   |   |   NOX >  0.651 : LM9 (4/34.3%)
|   RM >  7.44 : LM10 (30/49.7%)

Models at the leaves:
  Smoothed (complex):

    LM1:  MedHouseVal = 75.7 - 0.129CRIM + 0.00632ZN + 0.441CHAS - 2.99NOX
                        + 0.184RM - 22.4DIS + 0.0711RAD - 0.00231TAX
                        - 0.118PTRATIO - 0.997LSTAT
    LM2:  MedHouseVal = 36.4 - 0.148CRIM + 0.00632ZN + 0.441CHAS - 2.99NOX
                        + 0.184RM - 3.96DIS + 0.0711RAD - 0.00926TAX
                        - 0.118PTRATIO - 0.354LSTAT
    LM3:  MedHouseVal = 20.2 - 0.0182CRIM + 0.0032ZN + 0.198CHAS - 1.32NOX
                        + 3.81RM - 0.0448AGE - 0.707DIS + 0.225RAD - 0.0124TAX
                        - 0.554PTRATIO - 0.205LSTAT
    LM4:  MedHouseVal = 0.633 - 0.0204CRIM + 0.00546ZN - 0.285INDUS + 0.278CHAS
                        + 5.39NOX + 6.16RM - 0.00522AGE - 0.552DIS + 0.0432RAD
                        - 0.00146TAX - 0.355PTRATIO - 0.61LSTAT
    LM5:  MedHouseVal = 22.7 - 0.0204CRIM + 0.00546ZN - 0.285INDUS + 0.278CHAS
                        + 5.39NOX + 2.89RM - 0.0289AGE - 0.552DIS + 0.0432RAD
                        - 0.00146TAX - 0.355PTRATIO - 0.61LSTAT
    LM6:  MedHouseVal = 46 - 0.0204CRIM + 0.00546ZN - 0.146INDUS + 0.278CHAS
                        + 6.98NOX - 0.322RM - 0.0162AGE - 0.41DIS + 0.0432RAD
                        - 0.00146TAX - 0.355PTRATIO - 0.662LSTAT
    LM7:  MedHouseVal = 37.9 - 0.0204CRIM + 0.00546ZN + 0.278CHAS + 1.79NOX
                        + 1.5RM - 0.271DIS + 0.0432RAD - 0.00146TAX
                        - 0.514PTRATIO - 1.28LSTAT
    LM8:  MedHouseVal = 39 - 0.0204CRIM + 0.00546ZN + 0.278CHAS + 7.87NOX
                        + 1.5RM - 0.711DIS + 0.0432RAD - 0.00146TAX
                        - 0.514PTRATIO - 1.52LSTAT
    LM9:  MedHouseVal = 37.4 - 0.0204CRIM + 0.00546ZN + 0.278CHAS + 7.87NOX
                        + 1.5RM - 0.271DIS + 0.0432RAD - 0.00146TAX
                        - 0.514PTRATIO - 1.52LSTAT
    LM10:  MedHouseVal = 24.8 - 6.19CRIM + 0.00546ZN + 0.649INDUS + 0.278CHAS
                         - 1.76NOX + 5.82RM - 0.379DIS + 0.0432RAD - 0.00146TAX
                         - 1.5PTRATIO - 0.492LSTAT
```

Figura 15: *Model Tree* Gerada pelo $\mathcal{M}5$ Utilizando o Conjunto de Dados Housing

```
Standard Regression Rules Conversor      Copyright (c) Daniel Gomes Dosualdo
Inducer: m5                              Input File:  housing.out
Date: Mon Mar 31 17:13:32 2003           Output File: housing.rules


[R0001] IF RM <= 6.68
                AND DIS <= 1.98
                AND LSTAT <= 14.6
        THEN MedHouseVal = 75.7 - 0.129CRIM + 0.00632ZN + 0.441CHAS - 2.99NOX
             + 0.184RM - 22.4DIS + 0.0711RAD - 0.00231TAX - 0.118PTRATIO - 0.997LSTAT

[R0002] IF RM <= 6.68
                AND DIS <= 1.98
                AND LSTAT >  14.6
        THEN MedHouseVal = 36.4 - 0.148CRIM + 0.00632ZN + 0.441CHAS - 2.99NOX
             + 0.184RM - 3.96DIS + 0.0711RAD - 0.00926TAX - 0.118PTRATIO - 0.354LSTAT
                .
                .
                .

[R0005] IF RM >  6.68
                AND RM <= 7.44
                AND INDUS <= 7.82
                AND RM <= 7.12
                AND LSTAT >  5.2
        THEN MedHouseVal = 22.7 - 0.0204CRIM + 0.00546ZN - 0.285INDUS + 0.278CHAS
                           + 5.39NOX + 2.89RM - 0.0289AGE - 0.552DIS + 0.0432RAD
                           - 0.00146TAX - 0.355PTRATIO - 0.61LSTAT

[R0006] IF RM >  6.68
                AND RM <= 7.44
                AND INDUS <= 7.82
                AND RM >  7.12
        THEN MedHouseVal = 46 - 0.0204CRIM + 0.00546ZN - 0.146INDUS + 0.278CHAS
                           + 6.98NOX - 0.322RM - 0.0162AGE - 0.41DIS + 0.0432RAD
                           - 0.00146TAX - 0.355PTRATIO - 0.662LSTAT
                .
                .
                .

[R0009] IF RM >  6.68
                AND RM <= 7.44
                AND INDUS >  7.82
                AND NOX >  0.589
                AND NOX >  0.651
        THEN MedHouseVal = 37.4 - 0.0204CRIM + 0.00546ZN + 0.278CHAS + 7.87NOX
             + 1.5RM - 0.271DIS + 0.0432RAD - 0.00146TAX - 0.514PTRATIO - 1.52LSTAT

[R0010] IF RM >  6.68
                AND RM >  7.44
        THEN MedHouseVal = 24.8 - 6.19CRIM + 0.00546ZN + 0.649INDUS + 0.278CHAS
                           - 1.76NOX + 5.82RM - 0.379DIS + 0.0432RAD
                           - 0.00146TAX - 1.5PTRATIO - 0.492LSTAT
```

Figura 16: Algumas Regras da Figura 15 Convertidas para o Formato Padrão de Regras de Regressão

```
Semantic Threatment of Regression Rules  Copyright (c) Daniel Gomes Dosualdo
Inducer: m5                              Input File:  housing.rules
Date: Mon Mar 31 17:13:32 2003           Output File: housing.Rules


[R0001] IF DIS <= 1.98
                AND RM <= 6.68
                AND LSTAT <= 14.6
        THEN MedHouseVal = 75.7 - 0.129CRIM + 0.00632ZN + 0.441CHAS - 2.99NOX
             + 0.184RM - 22.4DIS + 0.0711RAD - 0.00231TAX - 0.118PTRATIO - 0.997LSTAT

[R0002]  IF DIS <= 1.98
                AND RM <= 6.68
                AND LSTAT >  14.6
        THEN MedHouseVal = 36.4 - 0.148CRIM + 0.00632ZN + 0.441CHAS - 2.99NOX
             + 0.184RM - 3.96DIS + 0.0711RAD - 0.00926TAX - 0.118PTRATIO - 0.354LSTAT
                .
                .
                .

[R0005]  IF INDUS <= 7.82
                AND RM <= 7.12
                AND RM >  6.68
                AND LSTAT >  5.2
         THEN MedHouseVal = 22.7 - 0.0204CRIM + 0.00546ZN - 0.285INDUS + 0.278CHAS
                            + 5.39NOX + 2.89RM - 0.0289AGE - 0.552DIS + 0.0432RAD
                            - 0.00146TAX - 0.355PTRATIO - 0.61LSTAT

[R0006]  IF INDUS <= 7.82
                AND RM <= 7.44
                AND RM >  7.12
         THEN MedHouseVal = 46 - 0.0204CRIM + 0.00546ZN - 0.146INDUS + 0.278CHAS
                            + 6.98NOX - 0.322RM - 0.0162AGE - 0.41DIS + 0.0432RAD
                            - 0.00146TAX - 0.355PTRATIO - 0.662LSTAT
                .
                .
                .

[R0009]  IF INDUS >  7.82
                AND RM <= 7.44
                AND RM >  6.68
                AND NOX >  0.651
        THEN MedHouseVal = 37.4 - 0.0204CRIM + 0.00546ZN + 0.278CHAS + 7.87NOX
             + 1.5RM - 0.271DIS + 0.0432RAD - 0.00146TAX - 0.514PTRATIO - 1.52LSTAT

[R0010]  IF RM >  7.44
        THEN MedHouseVal = 24.8 - 6.19CRIM + 0.00546ZN + 0.649INDUS + 0.278CHAS
                           - 1.76NOX + 5.82RM - 0.379DIS + 0.0432RAD
                           - 0.00146TAX - 1.5PTRATIO - 0.492LSTAT
```

Figura 17: Regras da Figura 16 Convertidas para o Formato Padrão de Regras de Regressão com Tratamento Semântico

6 Considerações Finais

A regressão é uma tarefa pouco explorada dentro da área de Aprendizado de Máquina e Mineração de Dados, visto que a maioria trata de problemas de classificação. Da mesma maneira, no DISCOVER, o panorama não é muito diferente.

Nesse sentido, este relatório técnico apresenta a definição de uma sintaxe padrão para representar regras de regressão no DISCOVER, e a implementação de uma classe para converter as saídas dos algoritmos de regressão Cubist, RT e WEKA para essa sintaxe. Além disso, relata a implementação de outra classe que realiza um tratamento semântico das regras de regressão.

Essa sintaxe para representar regras de regressão é importante pois permite a implementação de conversores de diferentes algoritmos para um formato único. Essa unificação permite implementar diversas medidas de pós-processamento uma única vez e utilizá-las para os tipos de saída gerada pelos diferentes algoritmos.

Por fim, essa sintaxe objetiva incrementar a potencialidade do ambiente DISCOVER para trabalhar problemas de regressão, uma vez que nesse ambiente a maioria das implementações estão voltadas para problemas de classificação.

A Código Fonte da Classe de Conversão para a Sintaxe Padrão de Regras de Regressão

Neste apêndice é apresentado o código da classe `RegConvert` implementada para converter as hipóteses geradas pelo Cubist, RT e WEKA para a sintaxe padrão de regras de regressão definida para o DISCOVER.

```
package RegConvert;
use Array::Compare;


#-----------------------------------------------------------------------
# Constructor Method
#-----------------------------------------------------------------------
sub new
{
  my $class = shift;
  my $self = {};
  bless $self,$class;
  return $self;
}
#-----------------------------------------------------------------------



#-----------------------------------------------------------------------
# Convert from Cubist to Standard Regression Rule Format
#-----------------------------------------------------------------------
sub cubist_to_stdRegRule
{
  my($self,$dataset,$inducer) = @_;
  my($rr_number,@nominal,@numeric,$rule,$where_is,@stack,$count,$braces,$input_file,
      $output_file);

   $input_file = "$dataset.out";
   $output_file= "$dataset.rules";

  open(IN, $input_file) || die "Could not open the input file\n";
  open(OUT, ">$output_file");

   print OUT "Standard Regression Rules Conversor\t Copyright (c) Daniel Gomes Dosualdo\n";
   print OUT "Inducer: $inducer\t\t\t Input File:  $input_file\n";
   print OUT "Date: ".gmtime(time()).".\t\t Output File: $output_file\n\n\n";

   $names_file = "$dataset.names";
  open(NAMES, $names_file) || die "Could not open the names file\n";

   $count = 0;
   $rr_number = 1;
   $where_is = "begin";
   $braces = "close";

   while(<NAMES>)
   {
     if ((!/^\|/) && (/:/)){
       if (!/continuous/)
       {
         /([^:]+):/;
         push(@nominal,$1);  # Push nominal attributes in @nominal
       }
       elsif (/continuous/)
       {
         /([^:]+):/;
         push(@numeric,$1);  # Push numeric attributes in @numeric
```

```
    }
  }
}

close NAMES;  # Close the names file

while(<IN>)
{
  chop;       # \n at end
  s/^\s+//;  # white spaces at begin
  s/\s+$//;  # white spaces at end

  if (($where_is ne "end") && (/[^\s]/))
  {
    if (/^Model/)
    {
      $where_is = "rules";  # Beginning of Cubist rules
    }
    elsif (/^Evaluation/)  # End of Cubist rules; print the last rule
    {
      $rule = join("",@stack);
      $rule =~ s/- /-/g;
      $rule =~ s/\+ /\+/g;
      foreach (@numeric)
      {
        $tmp = $_;
        $rule =~ s/$tmp\+/$tmp \+/g;
        $rule =~ s/$tmp\-/$tmp \-/g;
      }
      print OUT "$rule\n\n";
      $where_is = "end";
      }
      elsif ($where_is eq "rules")  # Cubist rules
      {
        if (/^Rule/)
        {
          if ($rr_number != 1)  # Except the first rule
          {
            $rule = join("",@stack);
            $rule =~ s/- /-/g;
            $rule =~ s/\+ /\+/g;
            foreach (@numeric)
            {
              $tmp = $_;
              $rule =~ s/$tmp\+/$tmp \+/g;
              $rule =~ s/$tmp\-/$tmp \-/g;
            }
            print OUT "$rule\n\n";  # Print the conclusion of regression rule
            $count = 0;
            @stack = undef();
          }
        }
        elsif (/^if/)
        {
          # Print the first part of regression rule
          printf OUT "[R%04d]\t IF ",$rr_number++;
        }
        elsif (/^then/)
        {
          $rule = join("\n\t\t AND ",@stack);
          $rule =~ s/in {/IN {/g;
          print OUT "$rule\n\t THEN ";  # Print the conditions of regression rule
          $count = 0;
          @stack = undef();
        }
        else
        {
          if (/{/)  # Values of nominal attributes
          {
            $braces = "open";
```

```
            }
            if ($braces eq "close")
            {
              $tmp = $_;
              foreach (@nominal)
              {
                if ($tmp =~ /$_ = /)
                {
                  $tmp =~ s/= /IN {/;
                  $tmp .= "}";
                }
              }
            $stack[$count++] = $tmp;
            }
            else
            {
              $stack[$count] .= $_;
              if (/}/)
              {
                $count++;
                $braces = "close";
              }
            }
          }
        }
  } # End of statement 'if (($where_is ne "end") && (/[^\s]/))'
} # End of statement 'while (<IN>)'

close IN;        # Close input file
close OUT;       # Close output file

}
#---------------------- End of sub cubist_to_stdRegRule ----------------------



#-----------------------------------------------------------------------------
# Convert from RT to Standard Regression Rule Format
#-----------------------------------------------------------------------------
sub rt_to_stdRegRule
{
  my($self,$dataset,$inducer) = @_;
  my($rr_number,%nominal,$tmp,$rule,$attr,$tree,$input_file,$output_file);
  my(@stack1,@stack2,@split_var,@valid_values,@all_values,@result);

  $input_file = "$dataset.out";
  $output_file= "$dataset.rules";

  open(IN, $input_file) || die "Could not open the input file -tchan\n";
  open(OUT, ">$output_file");

  print OUT "Standard Regression Rules Conversor\t Copyright (c) Daniel Gomes Dosualdo\n";
  print OUT "Inducer: $inducer\t\t\t Input File:  $input_file\n";
  print OUT "Date: ".gmtime(time()).".\t\t Output File: $output_file\n\n\n";

  $domain_file = "$dataset.domain";
  open(DOMAIN, $domain_file) || die "Could not open the domain file\n";

  $rr_number = 1;
  $tree = "false";

  while(<DOMAIN>)
  {
    if ((!/^\./) && (/:/) && (!/continuous/))
    {
      s/ //g;
      /([^:]+):([^\.|^\n]+)[\.|\n]/;
      $nominal{"$1"} = $2;  # Set %nominal with nominal attributes and their values
    }
  }
```

```
close DOMAIN;  # Close domain file

while(<IN>)
{
  chop;        # \n at end
  s/^\s+//;  # white spaces at begin

  last if (/^NUMBER OF NODES/);  # End of tree

  if (/^RESULTING TREE/)  # Beginning of tree
  {
      $tree = "true";
  }
  elsif (($tree eq "true") && (/[^\s]/))
  {
    if (/^True/)
    {
      push(@stack2, "True");
    }
    elsif (/^False/)
    {
      $stack2[$#stack2] = "False";
    }
    elsif (/^LEAF/)  # Leaves
    {
      /::[\s]*([^\[]+)\[/;
      $tmp = $1;
      $rule = join("\n\t\tAND ",@stack1);
      $rule =~ s/In/IN/g;
      $rule =~ s/{ /{/g;
      $rule =~ s/ }/}/g;
      if (($tmp =~ /\*/) && ($tmp =~ /\(R.*\)/))  # RETIS Linear Equation
      {
        $tmp =~ s/\*/ /g;
        $tmp =~ s/-[\s]*/-/g;
        $tmp =~ s/\+[\s]*/\+/g;
        $tmp =~ s/\(R.*\)//;
      }
      $tmp =~ s/[\s]+/ /g;
      if ($#stack1 == -1)
      {
        print OUT "$tmp\n\n";
      }
      else
      {
        # Print regression rules
        printf OUT "[R%04d]\t IF $rule\n\t THEN $tmp\n\n",$rr_number++;
      }
      while (($stack2[$#stack2]) eq "False")
      {
        pop(@stack1);
        pop(@stack2);
      }

      if ($#stack1 >= 0) #1
      {
        # Look for the operator and changes it for its opposite
        if ($stack1[$#stack1] !~ s/<=/>/) #2
        {
          if ($stack1[$#stack1] !~ s/>=/</) #3
          {
            if ($stack1[$#stack1] !~ s/</>=/) #4
            {
              if ($stack1[$#stack1] !~ s/>/<=/) #5
        {
            if ($stack1[$#stack1] !~ s/=/<>/) #6
        {
          if ($stack1[$#stack1] !~ s/<>/=/) #7
          {
```

```
            if ($stack1[$#stack1] =~ /In/) #8
            {
              $tmp = $stack1[$#stack1];
              @split_var = split(/ In /,$tmp);
              # $attr variable keeps the name of the variable
              $attr = $split_var[0];
              $tmp =~ /\{([^\}]+)\}/;

              # @valid_values keeps the values assumed for the $attr variable
              # in that rule
              # $1 receives the values of the variable whitout the braces
              @valid_values = split(/, /,$1);

              while (($var, $value) = each(%nominal))
              {
                if ($attr eq $var)
                {
                  @all_values = split(/,/,$value);
                }
              }

                      foreach (@valid_values)
                      {
                        s/ //g;
                      }

              @result = undef();

                      foreach $tmp (@all_values)
              {
                if (!grep($tmp eq $_, @valid_values))
                {
                  # @result keeps the values of the domain file that the
                  # variable didn't assume in that rule
                  push(@result,$tmp);
                }
              }

              $stack1[$#stack1] = "$attr IN {";
              foreach (@result)
              {
                $stack1[$#stack1] .= "$_, ";
              }
              $stack1[$#stack1] .= "}";
                      $stack1[$#stack1] =~ s/IN {, /IN {/;
                      $stack1[$#stack1] =~ s/,[\s]*}/}/;
            } #8
          } #7
        } #6
              } #5
        } #4
      } #3
        } #2
      } #1
    } # End of statement elsif (/LEAF/)

    else  # Conditions of the rule
    {
      /([^\(]+)\(/;
      $tmp = $1;
      if ($tmp =~ /In/)
      {
        $tmp =~ s/  /, /g;
      }
      push(@stack1,$tmp);  # Push the conditions in @stack1
    }
  } # End of statement 'else'
} # End of statement 'while <IN>'

close IN;        # Close input file
```

```
close OUT;        # Close output file

}
#------------------------------- End of sub rt_to_stdRegRule -------------------------------



#-----------------------------------------------------------------------------------------
# Convert from Weka to Standard Regression Rule Format
#-----------------------------------------------------------------------------------------
sub weka_to_stdRegRule
{
  my($self,$dataset,$inducer) = @_;
  my($rr_number,$count,$tree_level,$node_level,$rule,$tmp,$where_is,$key);
  my($number_nominal,$equation,$keep_equation);
  my(%nominal,@attributes,@stack,@node,@result,$keep_equation);
  my($keep_conditions,@valid_values,@all_values,$set_equation,$comp);
  my(@nominal_values1,@nominal_values2,$input_file,$output_file);

   $input_file = "$dataset.out";
   $output_file= "$dataset.rules";

   open(IN, $input_file) || die "Could not open the input file\n";
   open(OUT, ">$output_file");

   print OUT "Standard Regression Rules Conversor\t Copyright (c) Daniel Gomes Dosualdo\n";
   print OUT "Inducer: $inducer\t\t\t\t Input File:   $input_file\n";
   print OUT "Date: ".gmtime(time()).".\t\t Output File: $output_file\n\n\n";

   $arff_file = "$dataset.data.arff";
   open(ARFF, $arff_file) || die "Could not open the arff file\n";

   $comp = Array::Compare->new(Sep => ',');
   $rr_number = 1;
   $tree_level = $count = $number_nominal = $equation = 0;
   $where_is = "begin";
   $key = "false";

   while(<ARFF>)
   {
     if (/\@attribute/)
     {
       chop;
       s/\@attribute //;
       /([^ ]+) /;
       push(@attributes,$1);  # Push all kind of attributes in @attributes
       if (/.*\{.*\}/)
       {
         /([^ ]+) \{([^\}]+)\}/;
         $nominal{"$1"} = $2;  # Set %nominal with nominal attributes and their values

       }
     }
   }

   close ARFF;   # Close the arff file

   while(<IN>)
   {
     chop;       # \n at end
     s/^\s+//;  # white spaces at begin
     s/\s+$//;  # white spaces at end

     if (($where_is ne "end") && (/[^\s]/))
     {
       if (/Pruned/)  # Beginning of tree
       {
         $where_is = "tree";
       }
       elsif (/Unsmoothed/)  # M5 Regression, with constant values at leaves
```

```
{
  $where_is = "constants";
}
elsif (/Smoothed/)
{
  $where_is = "equations";  # M5 Model, with linear equations at leaves
}
elsif (/Number of Rules/)  # End of tree
{
  /([^\d+]+)(\d+)/;
  if ($2 == 1)
  {
    $one_rule = "true";
  }
  else
  {
    $one_rule = "false";
  }
  if ($where_is eq "constants")
  {
    $where_is = "end";  # End of the constant values
  }
  else  # $where_is eq "equations"
  {
    $where_is = "last_equation";  # Last equation of the model
  }
}
elsif ($where_is eq "tree")
{
  $node_level = s/\|//g;  # count number of | (tree level)
  @node = split /:/;      # $node[0] is node, $node[1] is leaf
  if ($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++;
  $node[0] =~ s/^\s+//;

  foreach $attr (keys %nominal)  # Look for nominal attributes
  {
    if (/$attr/)
    {
      $key = "true";
    }
  }

  if ($key eq "true")
  {
    if (/<=/)
    {
      $node[0] =~ /([^=]+)=([^ ]+) /;
      $tmp = $2;
      @valid_values = split(/,/,$tmp);

      while (($var, $value) = each(%nominal))
      {
        if ($1 eq $var)
        {
          $value =~ s/ //g;
          @all_values = split(/,/,$value);
        }
      }

      @other = ();

      foreach $tmp (@all_values)
      {
```

```
      if (!grep($tmp eq $_, @valid_values))
      {
        $tmp .= ",";
            push(@other,$tmp);
      }
    }

        if ($#other >= 0)
        {
          $tmp = $1 . " IN {";
          foreach (@other)
          {
            $_ =~ s/ //g;
            $tmp .= "$_";
          }
          $tmp .= "}";
          $tmp =~ s/,}/}/;
          $tmp =~ s/,/, /g;
        }
        else
        {
          $tmp = "";
        }
      }
      elsif (/>/)
      {
        $node[0] =~ /([^>]+)>/;
        $tmp = $1;
        $tmp =~ s/,/, /g;
        $tmp =~ s/=/ IN {/;
        $tmp .= "}";
      }
    }
    else  # key eq false
    {
      $tmp = $node[0];
    }

    $key = "false";
    $tmp =~ s/ }/}/;
    push(@stack,$tmp);

    if ($node[1] =~ /\S+/)
    {
      $rule = join("\n\t\tAND ",@stack);
      push(@result,$rule);
    }
  }  # End of statement 'elsif ($where_is eq "tree")'
  elsif ($where_is eq "constants")  # Tree with constant values at leaves
  {
    @node = split /:/;
    $node[1] =~ s/^\s+//;
    if ($#result >= 0)
    {
      # Print regression rules with constant values in the conclusion
      printf OUT "[R%04d]\t IF $result[$count]\n\t THEN $node[1]\n\n",$rr_number++;
      $count++;
    }
    else  # Just one rule
    {
      print OUT "$node[1]\n";
    }
  }
  # Tree with linear equations at leaves
  elsif (($where_is eq "equations") || ($where_is eq "last_equation"))
  {
    if ($where_is eq "last_equation")
    {
      $where_is = "end";  # End of tree, print the last equation of the model
    }
```

```
if (($where_is eq "equations") && (/LM1:/))  # First linear equation of the model
{
  $equation = $_;
  $equation =~ s/LM1:\s+//;
}
elsif ((($where_is eq "equations") && (/LM[0-9]+:/)) || ($where_is eq "end")) # 1
{
  $number_nominal = 0;
  @list_attr = ();
  foreach (keys %nominal)
  {
    if ($equation =~ /$_/)  # Look for nominal attributes in linear equations
    {
      $number_nominal++;
      push(@list_attr,$_);
    }
  }
  if ($number_nominal == 0)  # Tree without nominal attributes in linear equations
  {
    if ($one_rule eq "true")
    {
      print OUT "$equation\n";
    }
    else
    {
      # Print regression rules without nominal attributes in conclusion
      printf OUT "[R%04d]\t IF $result[$count]\n\t THEN $equation\n\n",$rr_number++;
      $count++;
    }
  }
  else  # $number_nominal > 0; Tree with nominal attributes in linear equations
  { # 2
    $equation =~ s/- /-/g;
    $equation =~ s/\+ /\+/g;
    foreach (@attributes)
    {
      if ($equation =~ /\d*.?\d+$_/)
      {
        $equation =~ s/$_/ $_/g;
      }
    }

    $keep_equation = $equation;
    $keep_conditions = $result[$count];
    $set_equation = "false";

    for $i (0 .. $#result)
    {
      #print "result eh $result[$i]\n";
    }

    foreach $attr (@list_attr)
    { # 3
      if ($result[$count] =~ /$attr/)
      { #4
        $result[$count] =~ /($attr IN \{[^\}]+)\}/;
        $tmp = $1;
        $tmp =~ s/$attr IN {//;
        $tmp =~ s/ //g;
        @nominal_values1 = split(/,/,$tmp);

        $equation =~ /($attr=\S+)/;
        $tmp = $1;
        $tmp =~ s/$attr=//;
        @nominal_values2 = split(/,/,$tmp);

        # The nominal values in conditions (1) and nominal values in
        # conclusion (2) are the same
        if ($comp->perm(\@nominal_values1, \@nominal_values2))
        {
```

```
        $equation =~ s/ $attr=\S+//g;
        $keep_equation =~ s/ $attr=\S+//g;
      }
      else # Add the nominal values from conclusion in the conditions
      {
        $set_equation = "true";
        @conditions = $keep_equation =~ /$attr=\S+/g;
        $keep_equation =~ s/ $attr=\S+//g;
        foreach (@conditions)
        {
          s/=/ IN {/;
          s/,/, /g;
          $_ .= "}";

          if (length($keep_conditions) > 0)
          {
            $keep_conditions .= "\n\t\tAND $_";
          }
          else
          {
            $keep_conditions = $_;
          }

        }

        $equation =~ s/ [+-]\d*.?\d+ $attr=\S+//g;
     }

    } # 4
    else  # this nominal attribute is not in the conditions of the rule
    {
      $set_equation = "true";
      @conditions = $keep_equation =~ /$attr=\S+/g;
      $keep_equation =~ s/ $attr=\S+//g;
      foreach (@conditions)
      {
        s/=/ IN {/;
        s/,/, /g;
        $_ .= "}";

        if (length($keep_conditions) > 0)
        {
          $keep_conditions .= "\n\t\tAND $_";
        }
        else
        {
          $keep_conditions = $_;
        }

      }

      $equation =~ s/ [+-]\d*.?\d+ $attr=\S+//g;

    }

  }

  if (length($result[$count]) > 0)
  {
    # Print regression rule without extra constants
    printf OUT "[R%04d]\t IF $result[$count]\n\t THEN $equation\n\n",$rr_number++;
  }

  if ($set_equation eq "true")
  {
    # Print regression rule with extra constants
    printf OUT "[R%04d]\t IF $keep_conditions\n\t THEN $keep_equation\n\n",
              $rr_number++;
  }
```

```
            $count++;

          } # 2

          if ($where_is ne "end")
          {
            $equation = $_;
            $equation =~ s/LM[0-9]+:\s+//;
          }
        } # 1
        else  # Other lines of the equations
        {
          $equation .= " $_";
        }
      }  # End of statement 'elsif(($where_is eq "equations")||($where_is eq "last_equation"))'
    }  # End of statement 'if (($where_is ne "end") && (/[^s]/))'
  }  # End of statement 'while (<IN>)'

  close IN;      # Close input file
  close OUT;         # Close output file

}
#------------------------------- End of sub weka_to_stdRegRule -------------------------------


1;  # End of package
```

B Código Fonte da Classe de Tratamento Semântico para Regras de Regressão

Neste apêndice é apresentado o código da classe `RegSemantic` implementada para realizar um tratamento semântico nas regras convertidas para o formato padrão de regras de regressão definido neste relatório técnico.

```
package RegSemantic;
use List::Compare;


#-----------------------------------------------------------------------
# Constructor Method
#-----------------------------------------------------------------------
sub new
{
  my $class = shift;
  my $self = {};
  bless $self,$class;
  return $self;
}
#-----------------------------------------------------------------------



#-----------------------------------------------------------------------
# Semantic Threatment of Regression Rules
#-----------------------------------------------------------------------
sub semantic
{
  my($self,$dataset) = @_;
  my($input_file, $output_file, $names_file, $tmp, $tmp2, $braces);
  my($count, $conclusion, $found, $attribute, $value, $values, $type, $rule);
  my($rr_number, $count_conditions, $unique_rule, $lc);
  my(%attr, %nominal, %occurrence, @numeric, @conditions, @stack, @intersection)
  my(@new_values, $skip);

  $input_file = "$dataset.rules";
  $output_file = "$dataset.Rules";
  open(IN, $input_file) || die "Could not open the rules file\n";
  open(OUT, ">$output_file");

  $rr_number = 1;
  $count_conditions = -1;
  %attr = ("MIN" => "MIN",
           "MAX" => "MAX",
           "MIN_SIGN" => "MIN_SIGN",
           "MAX_SIGN" => "MAX_SIGN");



  #-----------------------------------------------------------------------
  # Search for inducer (cubist, m5, rt, cart, retis)
  #-----------------------------------------------------------------------

  while (<IN>)
  {
    last if (/Inducer/);
  }

  unless (/Inducer/)
  {
    die "Incorret rules file\n";
  }
```

```
/([^:]+):\s*([^\s]+)/;  # $2 take the correct inducer


#-----------------------------------------------------------------------
# Header
#-----------------------------------------------------------------------

print OUT "Semantic Threatment of Regression Rules\t Copyright (c) Daniel Gomes Dosualdo\n";
print OUT "Inducer: $2\t\t\t\t Input File:   $input_file\n";
print OUT "Date: ".gmtime(time()).".\t\t Output File: $output_file\n\n\n";



#-----------------------------------------------------------------------
# Open the names, domain or arff file according the inducer
#-----------------------------------------------------------------------

if ($2 eq "cubist")
{
  $names_file = "$dataset.names";
  open(NAMES, $names_file) || die "Could not open the names file\n";
  $braces = "close";

  while(<NAMES>)
  {
    s/^\s+//;  # white spaces at begin
    if ((!/^\|/) && (!/ignore/) && (!/label/)){
      if ((!/continuous/) && (!/:=/))
      {
        if (/:/)
        {
          /([^:]+):\s+([^\.|^\||^\n]+)[\.|\||\n]/;
          $tmp = $1;
          $nominal{"$tmp"} = $2;  # Push nominal attributes in @nominal
          $nominal{"$tmp"} =~ s/\s+$//;
          if (!/.\$/)
          {
            $braces = "open";
          }
        }
        elsif ($braces eq "open")
        {
          /([^.]+)./;
          $nominal{"$tmp"}.= " $1";
          if (/.\$/)
          {
            $braces = "close";
            $nominal{"$tmp"} =~ s/\s+$//;
          }
        }
      }
      elsif ((/:/) && (/continuous/))
      {
        /([^:]+):/;
        push(@numeric,$1);  # Push numeric attributes in @numeric
      }
      elsif (/:=/)
      {
        /([^\s*:=]+)\s*:=/;
        push(@numeric,$1);
      }
    }
  }

  close NAMES;     # Close names file

}
elsif (($2 eq "rt") || ($2 eq "cart") || ($2 eq "retis"))
```

```
{
  $names_file = "$dataset.domain";
  open(NAMES, $names_file) || die "Could not open the domain file\n";

  while(<NAMES>)
  {
    if ((!/^\./) && (/:/))
    {
      if (!/continuous/)
      {
        /([^\s*:]+)\s*:([^\.|^\n]+)[\.|\n]/;
        $nominal{"$1"} = $2;  # Set %nominal with nominal attributes and their values
      }
      elsif (/continuous/)
      {
        /([^\s*:]+)\s*:/;
        push(@numeric,$1);
      }
    }
  }

  close NAMES;      # Close domain file

}
elsif ($2 eq "m5")
{
  $names_file = "$dataset.data.arff";
  open(NAMES, $names_file) || die "Could not open the arff file\n";

  while(<NAMES>)
  {
    if ((!/^%/) && (/\@attribute/))
    {
      s/\@attribute //;
      s/\s+$//;
      if ((/.*\{.*\}/) && (!/real$/))
      {
        /([^ ]+) \{([^\}]+)\}/;
        $nominal{"$1"} = $2;  # Set %nominal with nominal attributes and their values

      }
      elsif (/real$/)
      {
        /([^ ]+) /;
        push(@numeric,$1);  # Push numeric attributes in @numeric
      }
    }
  }

  close NAMES;      # Close arff file

}
else
{
  die "$2: Unknown Inducer\n";
}


#-----------------------------------------------------------------------------
# Semantic threatment of the rules
#-----------------------------------------------------------------------------

while(<IN>) #1
{
  s/^\s+//;  # white spaces at begin

  if (/[^\s]/) #2
  {
    $unique_rule = $_;
```

```
if ((/\[R\d+\]/) || (/^AND/))
{
  $count_conditions++;
  s/\[R\d+\]\s+IF //;
  s/^AND //;
  /([^ ]+) ([^ ]+) ([^\n]+)\n/;
  @{$conditions[$count_conditions]}[0] = $1;  # Name of variable
  @{$conditions[$count_conditions]}[1] = $2;  # Operator
  @{$conditions[$count_conditions]}[2] = $3;  # Value of that variable
}
elsif (/THEN/) #3
{
  @stack = undef();
  %occurrence = ();
  $count = 0;
  s/THEN //;
  $conclusion = $_;
  $skip = "false";

  for $i (0 .. $count_conditions)
  {
    $found = "false";
    foreach (keys (%occurrence))
    {
      if (@{$conditions[$i]}[0] eq $_)
      {
        $found = "true";
        $occurrence{"$_"}++;
      }
    }
    if ($found ne "true")
    {
      $tmp = @{$conditions[$i]}[0];
      $occurrence{"$tmp"} = 1;
    }
  }

  while (($attribute, $value) = each(%occurrence))
  {
    if ($value == 1)
    {
      for $i (0 .. $count_conditions)
      {
        if (@{$conditions[$i]}[0] eq $attribute)
        {
          $tmp = @{$conditions[$i]}[0]." ".@{$conditions[$i]}[1]
                 ." ".@{$conditions[$i]}[2];
          $stack[$count++] = $tmp;
        }
      }
    }
    else  # value > 1
    {
      $type = "none";
      foreach (@numeric)
      {
        if ($attribute eq $_)
        {
          $type = "numeric";
        }
      }
      if ($type eq "numeric")
      {
        for $i (0 .. $count_conditions)
        {
          if (@{$conditions[$i]}[0] eq $attribute)
          {
            if (((@{$conditions[$i]}[1] eq "<") || (@{$conditions[$i]}[1] eq "<="))
              && (($attr{"MAX"} eq "MAX") || (@{$conditions[$i]}[2] < $attr{"MAX"})))
```

```
        {
          $attr{"MAX"} = @{$conditions[$i]}[2];
          $attr{"MAX_SIGN"} = @{$conditions[$i]}[1];
          $tmp = @{$conditions[$i]}[0]." ".$attr{"MAX_SIGN"}." ".$attr{"MAX"};
        }
        elsif (((@{$conditions[$i]}[1] eq ">") || (@{$conditions[$i]}[1] eq ">="))
             && (($attr{"MIN"} eq "MIN") || (@{$conditions[$i]}[2] > $attr{"MIN"})))
        {
          $attr{"MIN"} = @{$conditions[$i]}[2];
          $attr{"MIN_SIGN"} = @{$conditions[$i]}[1];
          $tmp2 = @{$conditions[$i]}[0]." ".$attr{"MIN_SIGN"}." ".$attr{"MIN"};
        }
      }
    }
    if ($attr{"MAX"} ne "MAX")
    {
      $stack[$count++] = $tmp;
    }
    if ($attr{"MIN"} ne "MIN")
    {
      $stack[$count++] = $tmp2;
    }

    %attr = ("MIN" => "MIN",
             "MAX" => "MAX",
             "MIN_SIGN" => "MIN_SIGN",
             "MAX_SIGN" => "MAX_SIGN");

  }
  else  # type ne "numeric"
  {

    @new_values = @intersection = ();
    $found = "false";

    for $i (0 ..$count_conditions)
    {
      if (@{$conditions[$i]}[0] eq $attribute)
      {
        $tmp = @{$conditions[$i]}[2];
        $tmp =~ s/{//;
        $tmp =~ s/}//;
        $tmp =~ s/ //g;
        @new_values = split(/,/,$tmp);

        if ($found eq "true")
        {
          $lc = List::Compare->new(\@new_values, \@intersection);
          @intersection = $lc->get_intersection;
        }
        else
        {
          foreach (@new_values)
          {
            push(@intersection,$_);
          }
        }

        $found = "true";
      }
    }

    if ($#intersection >= 0)
    {
      $rule = join(", ",@intersection);
      $tmp = $attribute." IN {".$rule."}";
      $stack[$count++] = $tmp;
    }
    else
    {
```

```
                $skip = "true";
              }
            }
          }  # End of statement else (value > 1)
        }  # End of statemet while...

        if ($skip eq "false")
        {
          $rule = join("\n\t\tAND ",@stack);
          printf OUT "[R%04d]\t IF $rule\n\t THEN $conclusion\n\n",$rr_number++;
        }

        $count_conditions = -1;

      } #3
    } #2
  } #1

  if ($rr_number == 1)
  {
    print OUT "$unique_rule\n\n";
  }

close IN;    # Close input file
close OUT;       # Close output file
}
#------------------------------ End of sub semantic ------------------------------------

1;  # End of package
```

Referências

Batista, G. E. A. P. A. (2003). Pré-processamento de Dados em Aprendizado de Máquina Supervisionado. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Breiman, L., J. H. Friedman, R. A. Olshen, & C. J. Stone (1984). *Classification and Regression Trees*. Wadsworth International, Monterey, CA.

Cheng, S. (1998, Feb). Statistical approaches to predictive modeling in large databases. Dissertação de Mestrado, Simon Fraser University, British Columbia, Canada.

Dosualdo, D. G. (2002). Investigação de Regressão para Data Mining. Monografia do Exame de Qualificação de Mestrado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Geromini, M. R. (2002). Projeto e desenvolvimento da interface gráfica do sistema DISCOVER. Monografia de Qualificação de Mestrado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Karalic, A. (1995). Retis - a machine learning system. Disponível em: `http://www-ai.ijs.si/AramKaralic/retis/index.html`.

Martins, C. A. (2001). Interpretação de Clusters em Aprendizado de Máquina. Monografia do Exame de Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Melanda, E. A. (2000). Pós-processamento de Conhecimento de Regras de Associação. Monografia do Exame de Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Prati, R. C. (2003). O framework de integração do sistema DISCOVER. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001a). Extração de Informações Padronizadas para a Avaliação de Regras Induzidas por Algoritmos de Aprendizado de Máquina Simbólico. Relatório Técnico do ICMC - Número 145. Disponível em: `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip`.

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001b). Uma Proposta de Unificação da Linguagem de Representação de Conceitos de Algoritmos de Aprendizado de Máquina Simbólicos. Relatório Técnico do ICMC - Número 137. Disponível em: `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. Monografia do Exame de Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação - USP São Carlos.

Quinlan, J. R. (1992). Learning with continuous classes. In *Proceedings Australian Joint Conference on Artificial Intelligence*, pp. 343–348. World Scientific.

Rezende, S. O., J. B. Pugliesi, E. A. Melanda, & M. F. Paula (2003). *Sistemas Inteligentes: Fundamentos e Aplicações* (1 ed.)., Chapter 12 – Mineração de Dados, pp. 307–335. Barueri, SP: Manole.

Rulequest-Research (2000, 03). Rulequest research – Data mining tools. Disponível em: `http://www.rulequest.com/cubist-info.html`.

Till, D. (1996). *Teach Yourself Perl 5 in 21 Days*. Sams Publishing.

Torgo, L. (2001). Rt 4.1 user's manual. Disponível em: `http://www.ncc.up.pt/~ltorgo/RT/rt_manual.pdf`. Acesso em 25/07/2001.

Wang, Y. & I. H. Witten (1997). Inducing Models Trees for Continuous Classes. In *Proceedings of the Poster Papers of the European Conference on Machine Learning*, Department of Computer Science, University of Waikato, New Zeland.

Weiss, S. M. & N. Indurkhya (1995). Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research 3*, 383–403. Disponível em: `http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume3/weiss95a.ps`.

Witten, I. H. & E. Frank (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. São Francisco, Califórnia: Morgan Kaufmann. Disponível em: `http://www.cs.waikato.ac.nz/~ml/weka/book.html`.