UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2569

Descrição do Ambiente Computacional DiPER

Daniel Gomes Dosualdo

Solange Oliveira Rezende

Nº 204

RELATÓRIOS TÉCNICOS

São Carlos - SP

Julho/2003

Descrição do Ambiente Computacional DiPER*

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

Resumo

Mineração de dados refere-se ao processo responsável por identificar padrões em grandes conjuntos de dados com o objetivo de extrair um novo conhecimento. Depois de extraídos os padrões, a etapa de pós-processamento tem como objetivo avaliar alguns aspectos desses padrões. Um dos problemas da mineração de dados preditiva conhecido como regressão tenta predizer o valor de um atributo-meta contínuo baseado em um conjunto de atributos de entrada.

Por outro lado, o Discover é um projeto que está sendo desenvolvido no Laboratório de Inteligência Computacional. Ele tem como objetivo fornecer um ambiente integrado para apoiar as etapas do processo de descoberta de conhecimento, oferecendo funcionalidades voltadas para aprendizado de máquina e mineração de dados e textos.

Neste relatório técnico é apresentado o ambiente computacional Discover Post-Processing Environment of Regression – DiPER, que tem como objetivo auxiliar na avaliação de um modelo de regressão simbólico. Esse ambiente, integrado ao Discover, é composto por uma biblioteca de classes e oferece uma série de métodos para serem utilizados na etapa de pós-processamento do processo de mineração de dados.

Palavras-Chave: Mineração de Dados, Pós-Processamento, Regras de Regressão, Discover, DiPER.

Julho/2003

*Trabalho realizado com auxílio financeiro da FAPESP.

Sumário

1 Introdução **1**

2 O Projeto Discover **2**

3 Sistemas e Algoritmos de Regressão **4**

3.1 RT 4

3.2 WEKA 4

3.3 Cubist 5

4 O Ambiente DiPER **5**

4.1 Classe RegConvert 6

4.2 Classe RegSemantic 8

4.3 Classe RegMeasures 9

4.4 Classe RegTableMeasures 10

4.4.1 Tabela de Contingência e Medidas Derivadas 10

4.4.2 Algoritmo P-Class 11

4.4.3 Processo de Geração da Tabela de Contingência para Regras de Regressão 13

4.5 Classe RegHypothesis 14

4.6 Classe RegRoutines 16

4.7 Interface com Usuário 16

5 Considerações Finais **18**

Referências **21**

Lista de Figuras

1 Funcionalidades do DISCOVER 3

2 Funcionalidades do ambiente DIPER 6

3 Conjunto de regras padronizadas 7

4 Desempenho de acordo com número de pseudo-classes 12

5 Classes do ambiente DIPER 18

Lista de Tabelas

1 Formas de representação de hipótese dos algoritmos de regressão utilizados neste trabalho 6

2 Descrição da classe RegConvert 8

3 Descrição da classe RegSemantic 9

4 Descrição da classe RegMeasures 10

5 Tabela de contingência para a regra $B \rightarrow H$ 11

6 Descrição da classe RegTableMeasures 14

7 Descrição da classe RegHypothesis 15

8 Descrição da classe RegRoutines 16

9 Opções para executar o *script regression.pl* 17

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

1 Introdução

A grande quantidade de dados que pode ser armazenada nos dias de hoje a um custo relativamente baixo, a necessidade das empresas de extrair conhecimento a partir de suas bases de dados para conhecer melhor seus clientes, a popularização da Internet, o aparecimento de novas soluções tecnológicas, como a tecnologia de *Data Warehouse* (Inmon 1997) e os avanços científicos em diversas áreas, por exemplo nas áreas de aprendizado de máquina e estatística, impulsionaram o processo de mineração de dados, responsável por extrair padrões ou modelos embutidos em grandes conjuntos de dados (Fayyad, Piatetsky-Shapiro, Smyth, & Uthurusamy 1996).

Mineração de dados refere-se ao processo de análise e aplicação de algoritmos que, mediante limitações de eficiência computacional aceitáveis, são capazes de produzir uma relação particular de padrões a partir de grandes massas de dados (Fayyad, Piatetsky-Shapiro, Smyth, & Uthurusamy 1996). Trata-se de um processo iterativo e interativo constituído de várias etapas, no qual são utilizadas técnicas e métodos que auxiliam na identificação de padrões a partir dos dados, com o objetivo de extrair novo conhecimento.

O processo de mineração de dados inclui duas grandes atividades: a predição e a descrição. A atividade de predição, ou mineração de dados preditiva, utiliza-se de inferência indutiva para examinar exemplos que possuam algum rótulo e obter uma generalização que permita prever os rótulos de novos exemplos. A predição apresenta dois tipos de problemas, de acordo com os valores que os rótulos assumem: se os rótulos assumem valores nominais (discretos ou categóricos), o problema é denominado classificação, e se eles assumem valores contínuos, então o problema é chamado de regressão.

Os métodos de regressão já são estudados pela comunidade estatística há bastante tempo. Porém, nas áreas de aprendizado de máquina e mineração de dados, a maioria das pesquisas é voltada para os problemas de classificação, que são mais comumente encontrados na vida real do que problemas de regressão (Weiss & Indurkhya 1995). Recentemente, as pesquisas têm se voltado também para problemas de regressão, já que muitos problemas são desse tipo, como problemas relacionados com habilidades esportivas e controle dinâmico de robôs (Uysal & Güvenir 1999). Portanto, a regressão em mineração de dados preditiva é uma questão pouco explorada dentro do processo de extração de conhecimento de bases de dados, sendo de grande relevância o estudo de métodos para a exploração de tarefas desse tipo.

Os métodos de regressão simbólicos geralmente induzem hipóteses no formato de regras e árvores de regressão. As regras são diferentes das árvores, e mesmo as árvores geradas por diferentes algoritmos são distintas umas das outras. Desse modo, ao submeter um conjunto de dados a vários algoritmos de regressão, diferentes tipos de regressores são obtidos. Com isso, torna-se difícil avaliar o conhecimento extraído por diferentes algoritmos porque a avaliação precisa ser feita de maneira diferente para cada algoritmo, uma vez que cada modelo é representado de uma forma diferenciada. Essa avaliação dos modelos faz parte da etapa de pós-processamento do processo de mineração de dados. O primeiro ponto que deve ser avaliado na etapa de pós-processamento é o desempenho da predição (ou precisão) do modelo frente a novos exemplos. Porém, quando o problema é de regressão, não é fácil medir o desempenho de predição do modelo, pois como o rótulo a ser predito assume valores numéricos, não se pode afirmar se o valor predito está correto ou não. Além da precisão, existem ainda outros fatores que devem ser avaliados, como a compreensibilidade, que caracteriza o quão compreensível esse modelo é ao usuário, e a interessabilidade, que consiste em verificar se o conhecimento obtido por esse modelo é novo ou não. Além disso,

no caso de se tratar de um novo conhecimento, deve-se verificar se ele é interessante e útil para os usuários do processo de mineração de dados.

Por outro lado, em nosso laboratório de pesquisa – LABIC[1] (Laboratório de Inteligência Computacional) sempre foram implementados diversos tipos de programas por parte dos pesquisadores nas áreas de aprendizado de máquina e mineração de dados e textos, de acordo com as necessidades de cada um. Porém, por causa de vários fatores, como a falta de documentação desses programas, essas implementações não eram reaproveitadas, fazendo com que, muitas vezes, um mesmo programa fosse implementado diversas vezes por diferentes pessoas. Isso impulsionou a criação de um projeto que envolvesse os membros do laboratório, projeto esse que recebeu o nome de Discover. O objetivo do Discover é fornecer um ambiente integrado para apoiar as etapas do processo de descoberta do conhecimento, oferecendo funcionalidades voltadas para aprendizado de máquina, mineração de dados e de textos. No Discover, os programas são implementados como bibliotecas de classes e essas classes são empacotadas como componentes, com a composição deles sendo feita por meio de uma interface gráfica.

Assim sendo, este relatório técnico tem como objetivo apresentar o ambiente computacional Discover Post-Processing Environment of Regression – DiPER, que auxilia na avaliação de um modelo de regressão simbólico. O DiPER é composto por uma biblioteca de classes, implementada de acordo com as especificações do Discover, que oferece uma série de métodos para serem utilizados na etapa de pós-processamento do processo de mineração de dados.

Este relatório técnico está organizado da seguinte maneira: na Seção 2 é apresentado o projeto Discover, um projeto que está sendo desenvolvido no LABIC integrando diversos trabalhos nas áreas de aprendizado de máquina e mineração de dados e textos. Na Seção 3 são apresentados os sistemas e algoritmos de regressão utilizados neste trabalho. Na Seção 4 é detalhado o ambiente DiPER, um dos projetos integrado ao Discover, cujo objetivo é auxiliar na avaliação de um modelo de regressão simbólico. Para isso, serão detalhadas as classes do ambiente DiPER, relatando seus métodos, funcionalidades, arquivos de entrada e saída e módulos utilizados na implementação de cada classe. Por fim, na Seção 5 são feitas as considerações finais deste relatório técnico.

2 O Projeto Discover

Em nosso laboratório de pesquisa – LABIC – sempre foram implementados diversos tipos de programas por parte dos pesquisadores nas áreas de aprendizado de máquina e mineração de dados e textos, conforme as necessidades de cada um. Porém, devido a alguns fatores, como a falta de documentação dos programas já implementados, ou porque seus autores se desligavam do laboratório, normalmente essas implementações não eram reaproveitadas. Dessa maneira, muitas vezes, um mesmo programa era implementado diversas vezes por diferentes pessoas. Isso impulsionou a criação de um projeto que envolvesse vários 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 de textos, como apresentado na Figura 1.

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

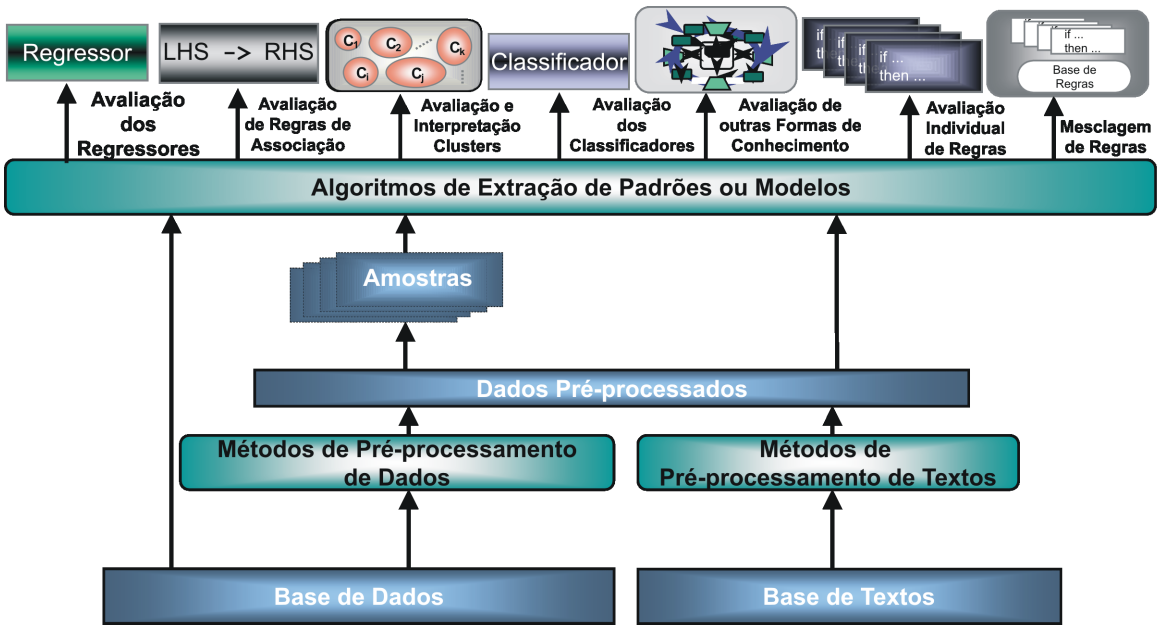

Figura 1: Funcionalidades do DISCOVER

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 modo a facilitar a execução de experimentos. Ou seja, o DISCOVER consistiria de um conjunto de ferramentas independentes que poderiam ser compostas conforme as necessidades dos usuários, e reunidas em um repositório compartilhado entre esses usuários. Assim, os usuários poderiam combinar essas ferramentas para executar experimentos mais complexos.

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

De um modo 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 maneira, 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. A sintaxe padrão para representação de dados é denominada DSX (*Discover Dataset Sintax*) e é apresentada em (Batista 2003). Com relação às sintaxes padrões para representação de conhecimento, destacam-se a sintaxe padrão para regras de decisão (Prati, Baranauskas, & Monard 2001), regras de regressão (Pugliesi, Dosualdo, & Rezende 2003), regras de associação (Melanda 2002) e agrupamentos (Martins 2001).

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

3 Sistemas e Algoritmos de Regressão

Nesta seção é apresentada uma descrição dos sistemas e algoritmos de regressão utilizados neste trabalho. Os sistemas RT e WEKA são *freeware*, enquanto que o LABIC possui uma versão do Cubist. Uma descrição mais detalhada dos sistemas e algoritmos de regressão apresentados nesta seção pode ser encontrada em (Dosualdo & Rezende 2003).

3.1 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, por intermédio do algoritmo *K-Nearest Neighbor*.

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, Stone, & Olshen 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.

3.2 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

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

porque certos algoritmos compartilham muitas funcionalidades, e dessa maneira, várias classes em um pacote podem ser utilizadas por mais de um algoritmo (Witten & Frank 1999).

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 M5, um algoritmo que trabalha com atributos e classes contínuas descrito em (Quinlan 1992). O algoritmo M5 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). O M5 permite obter como modelos de saída tanto uma *Model Tree* quanto uma *Regression Tree*. Neste trabalho será utilizado o termo M5 Model quando o algoritmo gera uma *Model Tree* e M5 Regression quando ele gera uma *Regression Tree*.

3.3 Cubist

O Cubist é uma ferramenta utilizada para a geração de modelos preditivos numéricos baseados em regras a partir de um conjunto de dados fornecido. 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 permite também construir outros tipos de modelos, como os compostos, que permitem melhorar a precisão da predição de um modelo baseado em regras combinando-o com um modelo baseado em exemplos (*Nearest Neighbor*) (Rulequest-Research 2001).

Além dos diferentes tipos de modelos que podem ser gerados, o Cubist disponibiliza ainda várias outras funcionalidades, como a utilização de um parâmetro de extrapolação (que controla quais predições feitas pelos modelos lineares do Cubist podem estar fora do intervalo de valores do conjunto de treinamento) e da técnica de *cross-validation*.

4 O Ambiente DiPER

A etapa de pós-processamento do processo de mineração de dados tem como objetivo avaliar o modelo resultante da extração de padrões. Neste trabalho, os modelos obtidos são regras e árvores de regressão. Dessa maneira, é muito importante avaliar esses modelos com relação a alguns fatores, como precisão, compreensibilidade e interessabilidade. A precisão diz respeito à eficiência com que o modelo prediz o valor do atributo-meta de novos exemplos. A compreensibilidade não é fácil de ser medida, e envolve alguns fatores, como a capacidade de abstração do usuário. Já a interessabilidade é responsável por verificar se o conhecimento obtido por esse modelo é novo ou não, e no caso de se tratar de um novo conhecimento, constatar se ele é interessante e útil para os usuários do processo de mineração de dados. Além disso, existem outras operações que são realizadas depois da extração de um modelo, por exemplo, comparar modelos resultantes da indução de diferentes algoritmos.

O ambiente DiPER (Discover Post-Processing Environment of Regression) é composto por uma biblioteca de classes e tem como objetivo auxiliar na avaliação de um modelo de regressão simbólico. Para isso, ele fornece uma série de métodos a serem

utilizados na etapa de pós-processamento. Esses métodos disponibilizam uma série de funcionalidades: conversão dos regressores gerados pelos algoritmos Cubist, RT, RETIS/RT, CART/RT, M5 Model e M5 Regression para um formato padrão de regras de regressão definido para o DISCOVER; aplicação de um tratamento semântico ao conjunto de regras de regressão no formato padrão; cálculo de medidas de precisão e compreensibilidade para regras de regressão; construção de uma tabela de contingência para regras de regressão e cálculo de uma série de medidas derivadas dessa tabela; e por fim, realização de um teste de hipóteses para comparar os regressores induzidos por diferentes algoritmos de regressão sobre um mesmo conjunto de dados. Na Figura 2 podem ser visualizadas as funcionalidades disponibilizadas pelo ambiente DIPER.

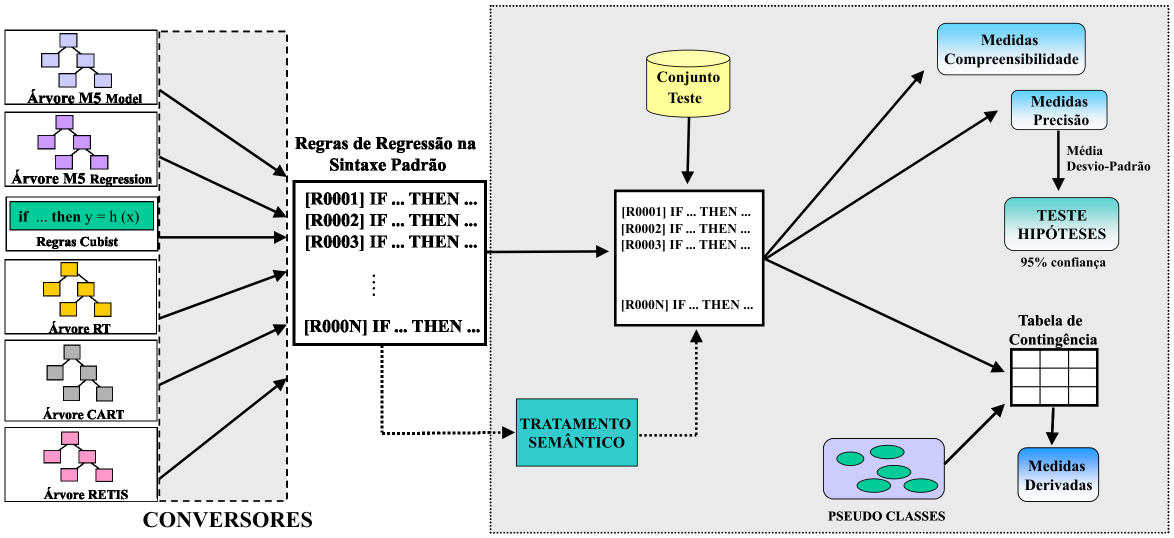

Figura 2: Funcionalidades do ambiente DIPER

4.1 Classe RegConvert – Conversão dos Regressores

Os métodos de regressão simbólicos geram como saída 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, nos mecanismos de poda e no formato em que a árvore pode ser visualizada.

Os algoritmos de regressão simbólicos utilizados neste trabalho, juntamente com a forma como representam suas hipóteses, são apresentados na Tabela 1.

Algoritmo	**Forma de Representação**
Cubist	Conjunto de regras de regressão
M5/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: Formas de representação de hipótese dos algoritmos de regressão utilizados neste trabalho

Devido aos diferentes formatos de representação das hipóteses geradas por esses algoritmos, qualquer medida ou estatística que precisasse ser calculada sobre as hipóteses teria de ser implementada várias vezes, uma para cada formato de hipótese. Dessa maneira, foi definida então uma sintaxe padrão de regras de regressão para o ambiente DISCOVER. A gramática dessa sintaxe é descrita detalhadamente em (Pugliesi, Dosualdo, & Rezende 2003). Baseado nessa sintaxe, padroniza-se os regressores convertendo-os para um formato único. Além de facilitar a implementação de medidas e cálculos a serem efetuados sobre as hipóteses, isso permite comparar os regressores gerados pelos diferentes algoritmos.

O processo de conversão dos regressores funciona da seguinte maneira: 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`. Em seguida, utiliza-se a classe **RegConvert** do ambiente DIPER, 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 definido para o ambiente DISCOVER (Dosualdo 2003). Independentemente do algoritmo de regressão utilizado para induzir a hipótese, o conjunto de regras padronizado obtido a partir dessa hipótese, apresenta o formato mostrado na Figura 3.

```
Standard Regression Rules Conversor     Copyright (c) Daniel Gomes Dosualdo
Inducer: <name of inducer>              Input File:  <dataset>.out
Date: <date>                            Output File: <dataset>.rules


[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 3: Conjunto de regras padronizadas

Na Tabela 2 é apresentada uma descrição da classe **RegConvert**, com seus métodos e funcionalidades, arquivos de entrada, arquivo de saída e os módulos do CPAN utilizados na implementação dessa classe. O CPAN[4] (*Comprehensive Perl Archive Network*) é um excelente repositório de código reusável na linguagem Perl. Para adicionar novos algoritmos de regressão ao ambiente DIPER, e conseqüentemente ao DISCOVER, basta adicionar um novo método à classe **RegConvert**, que receba como entrada o regressor no formato do novo algoritmo, e converta esse regressor para o formato padrão de regras de regressão.

[4]http://www.cpan.org/

Classe	**RegConvert**
Métodos	main, cubistToStdRegRule, rtToStdRegRule, wekaToStdRegRule, printHeader, printRule
Arquivos entrada	`.out`, `.names` (Cubist), `.domain` e `.data.pruned` (RT), `.data.arff` (WEKA)
Arquivo saída	`.rules`
Módulos CPAN	Array::Compare, List::Compare

Método	**Funcionalidade**
main	método principal (gerencia os demais métodos da classe)
cubistToStdRegRule	converte as regras geradas pelo Cubist para o formato padrão de regras de regressão
rtToStdRegRule	converte as árvores geradas pelo RT (algoritmos RT, CART e RETIS) para o formato padrão de regras de regressão
wekaToStdRegRule	converte as árvores geradas pelo algoritmo M5 (Model e Regression) do WEKA para o formato padrão de regras de regressão
printHeader	imprime cabeçalho no arquivo de saída
printRule	imprime regra de regressão (no formato padrão) no arquivo de saída

Tabela 2: Descrição da classe RegConvert

4.2 Classe RegSemantic – Tratamento Semântico

Muitas vezes, depois que os regressores são convertidos para o formato padrão de regras de regressão, as regras que constituem esse regressor podem possuir condições desnecessárias, principalmente quando o conjunto de regras é derivado de uma árvore. Isso acontece porque em muitos casos um mesmo atributo 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.

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 esse atributo numérico. Já 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.

Nesse sentido, aplica-se então um tratamento semântico ao arquivo `.rules`, que contém as regras no formato padrão com o objetivo de excluir as condições desnecessárias de cada regra. O tratamento semântico apresenta duas vantagens: melhora a compreensibilidade da regra, visto que quanto maior o número de condições, mais difícil o entendimento da regra; e aumenta o desempenho dos programas que utilizam essas regras como entrada, já que muitos testes desnecessários são evitados. Isso é bastante importante quando o arquivo de regras é extenso e o número de exemplos de teste é grande (Dosualdo 2003).

Para realizar o tratamento semântico, utiliza-se a classe **RegSemantic** do ambiente DIPER para gerar um arquivo com a extensão `.Srules`, contendo as regras semanticamente tratadas. Na Tabela 3 é apresentada uma descrição da classe **RegSemantic**.

Classe	**RegSemantic**
Métodos	main, semantic, searchInducer, openNamesFile, printHeader, printRule
Arquivos entrada	`.rules`, `.names` (Cubist), `.domain` (RT), `.data.arff` (WEKA)
Arquivo saída	`.Srules`
Módulo CPAN	List::Compare

Método	**Funcionalidade**
main	método principal (gerencia os demais métodos da classe)
semantic	aplica tratamento semântico ao conjunto de regras de regressão no formato padrão
searchInducer	procura e retorna o nome do algoritmo
openNamesFile	abre arquivo `.names`, `.domain` ou `.data.arff` (de acordo com algoritmo)
printHeader	imprime cabeçalho no arquivo de saída
printRule	imprime regra de regressão semanticamente tratada no arquivo de saída

Tabela 3: Descrição da classe RegSemantic

4.3 Classe RegMeasures – Cálculo de Medidas de Regressão

As regras de regressão, apesar de serem semelhantes às regras de classificação, predizem um valor numérico ao invés de um valor nominal, envolvendo um grau de incerteza quanto à classe predita (Cheng 1998). As medidas mais utilizadas para avaliar a precisão das hipóteses induzidas em problemas de regressão são: MAD (*Mean Absolute Deviation*) e MSE (*Mean Squared Error*) (Merz 1998; Torgo 1995). Essas duas medidas são baseadas na diferença entre o valor real do atributo-meta e o valor predito pelo algoritmo.

A medida MAD consiste na média da diferença (em módulo) entre os valores reais e preditos para um atributo-meta. Seja h_i a hipótese construída pelo algoritmo na i-ésima partição. O valor da MAD calculado em cada uma das i partições é obtido por meio da Equação 1.

$$MAD(h_i) = \frac{1}{n_{teste}} \sum_{j=1}^{n_{teste}} |y'_j - y_j| \tag{1}$$

na qual n_{teste} corresponde ao número de exemplos do arquivo de teste; y'_j corresponde ao valor predito pelo algoritmo no j-ésimo exemplo de teste; e y_j é o valor real do atributo-meta desse mesmo exemplo.

A medida MSE consiste na média da diferença ao quadrado entre os valores reais e preditos para um atributo-meta. Novamente, considerando h_i como sendo a hipótese gerada na i-ésima partição, o valor da MSE calculado em cada partição é obtido por meio da Equação 2.

$$MSE(h_i) = \frac{1}{n_{teste}} \sum_{j=1}^{n_{teste}} (y'_j - y_j)^2 \tag{2}$$

Outro fator, além da precisão, que deve ser avaliado, é a compreensibilidade de uma hipótese induzida. No entanto, tanto para um problema de regressão quanto de classificação, não é fácil medir a compreensibilidade, pois ela é dependente de vários fatores, como a capacidade de abstração do usuário; a forma de representação da hipótese induzida e o tamanho e a variação dos valores do conjunto de dados. Por exemplo, quanto maior o número de exemplos e maior a variação dos valores de seus atributos, maior tende a ser a

árvore induzida, e portanto, mais difícil tende a ser sua compreensão.

Para regras de regressão, um fator de compreensibilidade pode ser o número de condições da regra, visto que quanto maior o número de condições, mais difícil seu entendimento por parte do usuário. Além disso, o número de termos existentes na equação linear que prediz o valor do atributo-meta (para as regras que utilizam uma equação para fazer a predição) também está relacionado com a compreensibilidade da regra, uma vez que um número muito grande de termos compromete a compreensão da conclusão da regra. Já para as regras que predizem o valor do atributo-meta por meio de um valor constante, esse fator não pode ser aplicado.

Dessa maneira, a classe do ambiente DiPER denominada **RegMeasures** permite calcular as medidas de precisão MAD e MSE e possibilita que o usuário saiba qual o número de condições de cada regra de regressão e o número de termos da equação linear (Dosualdo 2003). Na Tabela 4 é apresentada uma descrição da classe **RegMeasures**, com seus métodos e funcionalidades, arquivos de entrada, arquivo de saída e os módulos do CPAN utilizados na implementação dessa classe.

Classe	**RegMeasures**
Métodos	main, calculateMeasures, searchInducer, printHeader, printRule, printEvaluation
Arquivos entrada	`.Srules` (ou `.rules`)
Arquivo saída	`.measures`
Módulos CPAN	Math::FixedPrecision, Math::Libm ':all'

Método	**Funcionalidade**
main	método principal (gerencia os demais métodos da classe)
calculateMeasures	calcula medidas de precisão e compreensibilidade
searchInducer	procura e retorna o nome do algoritmo
printHeader	imprime cabeçalho no arquivo de saída
printRule	imprime cada regra de regressão com suas respectivas medidas
printEvaluation	imprime avaliação global do modelo

Tabela 4: Descrição da classe RegMeasures

4.4 Classe RegTableMeasures – Tabela de Contingência

Diferentemente de outros problemas, como os de classificação, existem poucas medidas que podem ser utilizadas em problemas de regressão. Em (Dosualdo 2003) foi definindo um procedimento, descrito a seguir, que permite construir uma tabela de contingência para uma regra de regressão. Após construída essa tabela, torna-se possível calcular várias medidas, que normalmente não poderiam ser aplicadas aos problemas de regressão.

Para isso, primeiramente será definida uma tabela de contingência, como ela é calculada, e quais são as medidas derivadas a partir dela. Em seguida, é apresentado o algoritmo P-Class, utilizado na definição dessa tabela. Por fim, é detalhado o processo de geração da tabela de contingência para regras de regressão.

4.4.1 Tabela de Contingência e Medidas Derivadas

Considere uma regra de decisão $B \rightarrow H$, em que H é a classe associada. Diz-se que um exemplo é coberto por uma regra $B \rightarrow H$ quando todas as condições da regra são satisfeitas.

Dessa maneira, calcula-se a tabela de contingência para uma regra conforme a Tabela 5. Nessa tabela, $n(B)$ denota o número de exemplos para os quais todas as condições da regra são verdadeiras e $n(\overline{B})$ denota seu complemento (número de exemplos para os quais pelo menos uma das condições é falsa). Analogamente, $n(H)$ denota o número de exemplos para os quais a conclusão da regra é verdadeira (ou seja, a classe é predita corretamente) e $n(\overline{H})$ seu complemento. Conseqüentemente, $n(HB)$ denota o número de exemplos em que B e H são verdadeiros, $n(\overline{H}B)$ denota o número de exemplos em que B é verdadeiro e H é falso, e assim por diante. Finalmente, N denota o número total de exemplos (Lavrac, Flach, & Zupan 1999).

	B	$\overline{B}$	
H	$n(HB)$	$n(H\overline{B})$	$n(H)$
$\overline{H}$	$n(\overline{H}B)$	$n(\overline{H}\overline{B})$	$n(\overline{H})$
	$n(B)$	$n(\overline{B})$	N

Tabela 5: Tabela de contingência para a regra $B \rightarrow H$

A partir da tabela de contingência apresentada são definidas então uma série de medidas sobre regras calculadas em termos de estimativas de probabilidade (Lavrac, Flach, & Zupan 1999): Precisão, Erro, Confiabilidade Negativa, Sensitividade, Especificidade, Cobertura, Suporte, Novidade e Satisfação.

No entanto, não é trivial construir uma tabela de contingência para uma regra de regressão, pois como o atributo-meta a ser predito é um valor numérico, diferentemente da classificação, não se pode dizer se a conclusão da regra está correta ou não. Dessa maneira, também não é possível calcular as medidas descritas.

Como existem poucas medidas para avaliar o conhecimento extraído de problemas de regressão, surgiu então a idéia de se definir uma tabela de contingência para regras de regressão. Uma vez que a tabela fosse construída, tornar-se-ia possível calcular as medidas dela derivadas. A seguir é apresentado o algoritmo P-Class, utilizado para definir uma estratégia que permite calcular tabela de contingência para regras de regressão.

4.4.2 Algoritmo P-Class

O algoritmo P-Class (Algoritmo 1) é utilizado para gerar um conjunto de "pseudo-classes" em um processo de discretização descrito em (Weiss & Indurkhya 1995). O objetivo desse algoritmo é associar os valores do atributo-meta y às pseudo-classes de modo a minimizar a distância entre cada y_i e a média de sua pseudo-classe.

O P-Class descreve como associar um conjunto de valores $\{y_i\}$ a k classes da seguinte maneira: ordena os valores de $\{y\}$; associa aproximadamente um número igual de valores de $\{y_i\}$ vizinhos para cada pseudo-classe; move um valor y_i para uma pseudo-classe vizinha quando isso reduz a distância de y_i para a média de sua pseudo-classe; e por fim, concatena pseudo-classes com médias iguais.

Fornecido o valor de k, o algoritmo P-Class associa de maneira relativamente rápida os y valores de modo que as distâncias totais sejam minimizadas. No entanto, uma questão-chave é como determinar k, ou seja, o número de pseudo-classes a serem geradas. Em (Weiss & Indurkhya 1995) são realizados alguns experimentos para determinar o melhor número de pseudo-classes. Na Figura 4 é apresentado um gráfico do erro relativo *versus*

o número de pseudo-classes. Como pode ser visualizado nesse gráfico, à medida que o número de pseudo-classes aumenta, os resultados melhoram, até alcançar o valor 6.

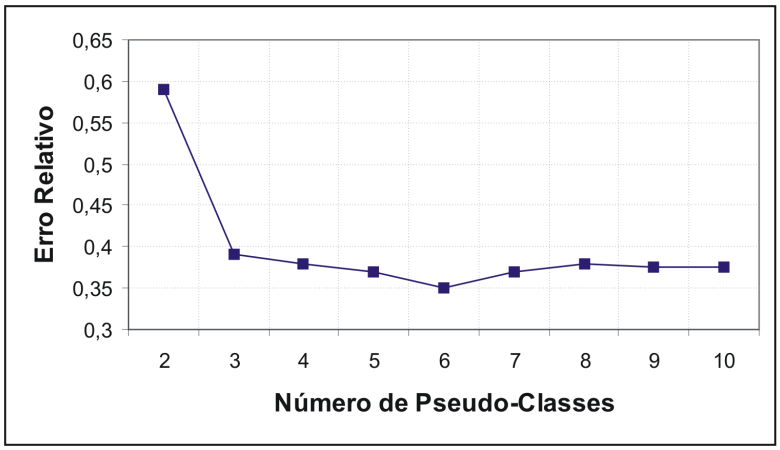

Figura 4: Desempenho de acordo com número de pseudo-classes (Weiss & Indurkhya 1995)

Algoritmo 1 Algoritmo P-Class (Weiss & Indurkhya 1995)

Require: $\{y_i\}$ a set of output values
Initialize n:=number of cases, k:=number of classes
for $i := 1$ to k **do**
  $Class_i$:=next n/k cases from list of sorted y values
end for
Compute Err_{new}
repeat
  $Err_{old} := Err_{new}$
  for each $Case_j$ **do**
    When it is in $Class_i$
    if Dist[$Case_j$, Mean($Class_{i-1}$)] $<$ Dist[$Case_j$, Mean($Class_i$)] **then**
      Move $Case_j$ to $Class_{i-1}$
    end if
    if Dist[$Case_j$, Mean($Class_{i+1}$)] $<$ Dist[$Case_j$, Mean($Class_i$)] **then**
      Move $Case_j$ to $Class_{i+1}$
    end if
  end for
  Compute Err_{new}
until Err_{new} is not less than Err_{old}
for each $Class_i$ **do**
  if Mean($Class_i$) = Mean($Class_j$) **then**
    Join $Class_i$ and $Class_j$
  end if
end for

O algoritmo P-Class foi desenvolvido com o intuito de transformar um problema de regressão em um problema de classificação. Para isso, os valores numéricos do atributo-meta passam por um processo de discretização, em que são geradas as pseudo-classes. Realizada a discretização dos valores (geradas as pseudo-classes), um algoritmo de indução de regras de decisão como o C4.5 (Quinlan 1993) ou o CN2 (Clark & Niblett 1989), pode ser

utilizado para gerar as regras. Com as regras induzidas, no processo proposto em (Weiss & Indurkhya 1995) são aplicados então procedimentos de poda e otimização para produzir um conjunto otimizado de regras.

4.4.3 Processo de Geração da Tabela de Contingência para Regras de Regressão

Como citado, o algoritmo P-Class apresentado foi desenvolvido para transformar um problema de regressão em um problema de classificação, gerando um conjunto de pseudo-classes por meio da discretização dos valores do atributo-meta. No entanto, em (Dosualdo 2003) o P-Class foi utilizado com um propósito diferente, para definir uma tabela de contingência para regras de regressão. Os valores do atributo-meta do conjunto de treinamento são, primeiramente, ordenados, e em seguida, fornecidos ao algoritmo P-Class. Depois de geradas as pseudo-classes, são então armazenados o menor e o maior valor do atributo-meta de cada uma dessas pseudo-classes. Esses valores constituem então os "pseudo-intervalos", que são utilizados posteriormente para construir a tabela de contingência de cada regra. O número de pseudo-intervalos a serem gerados é definido pelo usuário, sendo o padrão 6 adotado se nenhum valor for especificado. Conforme descrito anteriormente, o valor 6 obteve empiricamente a menor taxa de erro.

Gerados os pseudo-intervalos, começa então o processo de construção da tabela de contingência para cada regra de regressão. Primeiramente, verifica se a regra realiza suas predições por intermédio de um valor constante ou por meio de uma equação linear: se for por intermédio de uma constante, calcula-se o pseudo-intervalo a que pertence aquela regra, pois se o valor é constante, ele será o mesmo para todos os exemplos de teste. Em seguida, para cada exemplo de teste, aplica-se o seguinte procedimento (Dosualdo 2003):

1. se a regra prediz valores por meio de uma equação linear, substitui os atributos presentes na equação linear pelos seus respectivos valores do exemplo de teste em questão, e então, calcula-se o pseudo-intervalo a que pertence a regra para aquele exemplo de teste;

2. se todas as condições da regra forem verdadeiras, o valor de B na tabela de contingência é incrementado, e caso contrário, incrementa-se o valor de seu complemento $\overline{B}$;

3. calcula a qual pseudo-intervalo pertence o exemplo de teste (verificando a qual pseudo-intervalo pertence o valor do atributo-meta daquele exemplo de teste);

4. se o valor calculado para a regra (valor predito) e o valor real do atributo-meta pertencem ao mesmo pseudo-intervalo, então incrementa-se o valor de H na tabela de contingência, senão, o valor de $\overline{H}$ é incrementado;

5. baseado nos valores de B, $\overline{B}$, H e $\overline{H}$, calcula-se então os demais valores da tabela de contingência para aquela regra ($HB, \overline{H}B, H\overline{B}$ e $\overline{HB}$).

Construídas as tabelas de contingência para cada regra de regressão, torna-se então possível calcular as medidas derivadas. Para construir as tabelas de contingência e efetuar os cálculos das medidas derivadas utiliza-se a classe **RegTableMeasures** do ambiente DiPER, que gera um arquivo contendo o menor e o maior valor de cada pseudo-intervalo,

além da tabela de contingência e das medidas derivadas para cada regra de regressão (Dosualdo 2003).

Na Tabela 6 é apresentada uma descrição da classe **RegTableMeasures**, com seus métodos e funcionalidades, arquivos de entrada, arquivo de saída e os módulos do CPAN utilizados na implementação dessa classe.

Classe	**RegTableMeasures**
Métodos	main, contingencyTable, searchInducer, printHeader, printRule, calculateValues, table, createIntervals, join, calculateInterval
Arquivos entrada	`.Srules` (ou `.rules`)
Arquivo saída	`.table`
Módulos CPAN	Array::Reform, Math::FixedPrecision, Math::Libm ':all'

Método	**Funcionalidade**
main	método principal (gerencia os demais métodos da classe)
contingencyTable	calcula as tabelas de contingência e as medidas derivadas
searchInducer	procura e retorna o nome do algoritmo
printHeader	imprime cabeçalho no arquivo de saída
printRule	imprime cada regra de regressão com sua tabela de contingência e suas medidas
calculateValues	calcula os valores da tabela de contingência
table	gera os pseudo-intervalos a partir do arquivo de dados
createIntervals	agrupa os valores do atributo-meta e os retorna com suas respectivas médias
join	concatena pseudo-classes com médias iguais
calculateInterval	retorna o intervalo correto de uma regra ou de um exemplo de teste

Tabela 6: Descrição da classe RegTableMeasures

4.5 Classe RegHypothesis – Teste de Hipóteses

Baseando-se apenas em medidas de erro, como as medidas MAD e MSE, no caso da regressão, não é fácil perceber se um algoritmo é melhor que outro. Dessa maneira, a classe do ambiente DIPER denominada **RegHypothesis** permite realizar um teste de hipóteses que compara os algoritmos dois a dois, baseando-se em *cross-validation*.

Considere A um algoritmo e $erro(h_i)$ equivalente à medida MAD calculada sobre a hipótese induzida h_i na i-ésima partição. A média da medida MAD calculada sobre os regressores obtidos com o algoritmo A é dada pela Equação 3.

$$media(A) = \frac{1}{k}\sum_{i=1}^{k} erro(h_i) \tag{3}$$

em que k corresponde ao número de partições do conjunto de dados, nesse caso $k = 10$.

Em seguida, baseado no valor da média, calcula-se a variância da medida MAD de cada regressor gerado pelo algoritmo por meio da Equação 4.

$$var(A) = [\frac{1}{k-1}\sum_{i=1}^{k}(erro(h_i) - media(A))^2] \tag{4}$$

Por fim, calcula-se o desvio-padrão da medida MAD obtido sobre cada algoritmo baseado

na variância, como mostra a Equação 5.

$$dp(A) = \sqrt{var(A)} \tag{5}$$

Para decidir qual algoritmo é melhor que outro com um grau de confiança de 95% é assumido o caso geral para determinar se a diferença entre dois algoritmos – A_S e A_P – é significante ou não, assumindo uma distribuição normal. Em geral, a comparação é feita de modo que A_P é o algoritmo proposto e A_S é o algoritmo padrão. Para isso, a média e desvio padrão combinados são calculados de acordo com as Equações 6 e 7, respectivamente. Então, a diferença absoluta, em desvios padrões, é calculada por meio da Equação 8.

$$media(A_S - A_P) = media(A_S) - media(A_P) \tag{6}$$

$$dp(A_S - A_P) = \sqrt{\frac{dp(A_S)^2 + dp(A_P)^2}{2}} \tag{7}$$

$$ad(A_S - A_P) = \frac{media(A_S - A_P)}{dp(A_S - A_P)} \tag{8}$$

Dessa maneira, se $ad(A_S - A_P) > 0$ então A_P supera A_S. Porém, se $ad(A_S - A_P) \geq 2$ desvios padrões, então A_P supera A_S com grau de confiança de 95%. Por outro lado, se $ad(A_S - A_P) \leq 0$, então A_S supera A_P e se $ad(A_S - A_P) \leq -2$ desvios padrões, então A_S supera A_P com grau de confiança de 95% (Monard & Baranauskas 2003).

A realização desse teste por meio da classe **RegHypothesis** fornece um arquivo de saída contendo os resultados da comparação efetuada entre os algoritmos de regressão escolhidos pelo usuário, relatando quais algoritmos superam os outros e se existe algum algoritmo que supera todos os demais (Dosualdo 2003). Na Tabela 7 é apresentada uma descrição da classe **RegHypothesis**.

Classe	**RegHypothesis**
Métodos	main, convertFromDSX, runInducer, convertToStdRegRule, calculateMean, hypothesisTest, printHeader, printResults
Arquivos entrada	`.Srules` (ou `.rules`)
Arquivo saída	`.hypothesis`
Módulos CPAN	Math::FixedPrecision, Math::Libm ':all', Math::NumberCruncher

Método	**Funcionalidade**
main	método principal (gerencia os demais métodos da classe)
convertFromDSX	converte conjunto de dados do formato padrão DSX para o formato dos dados de entrada dos sistemas RT, WEKA e Cubist
runInducer	executa algoritmo com conjunto de dados especificado
convertToStdRegRule	converte regressores para o formato padrão de regras de regressão
calculateMean	calcula média, variância e desvio-padrão
hypothesisTest	compara precisão dos algoritmos de regressão de acordo com as medidas MAD e MSE
printHeader	imprime cabeçalho no arquivo de saída
printResults	imprime resultados do teste de hipóteses no arquivo de saída

Tabela 7: Descrição da classe RegHypothesis

4.6 Classe RegRoutines – Rotinas Auxiliares

Nesta seção é apresentada a classe do ambiente DiPER **RegRoutines**, que fornece vários métodos utilizados pelas demais classes do ambiente. O método *pushAttributes*, por exemplo, é utilizado por métodos das classes **RegMeasures** e **RegTableMeasures**. Na Tabela 8 é apresentada uma descrição da classe **RegRoutines**.

Classe	**RegRoutines**
Métodos	pushAttributes, equationOrConstant, openTestFile, checkConditions
Módulo CPAN	Math::Libm ':all'

Método	**Funcionalidade**
pushAttributes	empilha os atributos do conjunto de dados
equationOrConstant	verifica se a conclusão da regra é uma constante ou uma equação linear
openTestFile	abre arquivo de teste
checkConditions	verifica se todas as condições de uma regra são satisfeitas

Tabela 8: Descrição da classe RegRoutines

4.7 Interface com Usuário

Para interagir com o ambiente DiPER, e conseqüentemente utilizar as funcionalidades disponíveis no ambiente, o usuário precisa invocar os métodos implementados nas classes que compõem o ambiente. Os métodos podem ser invocados de duas maneiras. A primeira maneira é o usuário invocá-los dentro de seu próprio programa. Nesse caso, o usuário precisa invocar primeiramente o construtor da classe (por meio do método *new*) e em seguida invocar os métodos que desejar. A segunda maneira de utilizar o ambiente DiPER ocorre por meio de um *script* (denominado *regression.pl*) que atua como uma interface entre o ambiente e o usuário. Esse *script*, executado por linha de comando do sistema operacional, oculta do usuário detalhes da implementação do ambiente DiPER. Essa linha de comando é executada da seguinte maneira:

>regression.pl <conjunto de dados> -<opção> <parâmetros adicionais>

O primeiro parâmetro utilizado na execução é sempre o nome do conjunto de dados (sem qualquer extensão de arquivo) e o segundo parâmetro é a opção do usuário (conversão de um regressor para o formato padrão, tratamento semântico, cálculo de medidas de precisão e compreensibilidade, construção de uma tabela de contingência e cálculo das medidas derivadas dessa tabela e teste de hipóteses). Dependendo da opção, podem ser utilizados alguns parâmetros adicionais, sendo que alguns são obrigatórios e outros opcionais. As possíveis opções utilizadas para executar o *script regression.pl*, juntamente com suas respectivas linhas de comando são descritas a seguir:

-convert: converte os regressores gerados pelos algoritmos Cubist, RT, RETIS/RT, CART/RT, M5 Model e M5 Regression para o formato padrão de regras de regressão definido para o Discover. A linha de comando dessa opção recebe como parâmetro adicional o nome do algoritmo a partir do qual o regressor foi gerado.

>regression.pl <conjunto de dados> -convert <algoritmo>

-semantic: aplica um tratamento semântico ao conjunto de regras de regressão no formato padrão. Essa opção não utiliza nenhum parâmetro adicional.

>regression.pl <conjunto de dados> -semantic

-measures: calcula o valor das medidas de precisão MAD e MSE e de algumas medidas de compreensibilidade para cada regra de regressão do arquivo de regras. Essa opção aceita como parâmetro opcional o nome do conjunto de teste: se nenhum parâmetro for especificado, o *script* procura pelo arquivo `<conjunto de dados>.test`.

>regression.pl <conjunto de dados> -measures [**<arquivo de teste>**]

-table: constrói uma tabela de contingência para cada regra de regressão do arquivo de regras e calcula as medidas derivadas dessa tabela para cada regra. Essa opção aceita como parâmetro adicional dois parâmetros opcionais: o primeiro, k, especifica o número de pseudo-intervalos a serem gerados; se nenhum valor for especificado é adotado o valor padrão 6. O segundo parâmetro opcional é o nome do conjunto de teste, sendo utilizado o arquivo `<conjunto de dados>.test` se nenhum nome de arquivo for especificado.

>regression.pl <conjunto de dados> -table [**<k>**] [**<arquivo de teste>**]

-evaluation: avalia a precisão global de um regressor, ou seja, a média das medidas MAD e MSE calculadas por meio de *cross-validation*. Essa opção aceita dois parâmetros adicionais, sendo um obrigatório e um opcional. O parâmetro obrigatório é o nome do algoritmo de regressão, e o parâmetro opcional, n, corresponde ao número de partições utilizadas no *cross-validation*. Se o número de partições não for especificado, é adotado o valor padrão 10.

>regression.pl <conjunto de dados> -evaluation <algoritmo> [**<n>**]

-hypothesis: realiza um teste de hipóteses para comparar a precisão dos regressores induzidos pelos diferentes algoritmos de regressão sobre um mesmo conjunto de dados. Essa opção aceita dois parâmetros adicionais, sendo um obrigatório e um opcional. O parâmetro obrigatório é uma lista de valores contendo os nomes dos algoritmos de regressão a serem comparados, que devem estar separados por vírgulas. Já o parâmetro opcional, n, corresponde ao número de partições utilizadas no *cross-validation*. Novamente, se o número de partições não for especificado, é adotado o valor padrão 10.

>regression.pl <conjunto de dados> -hypothesis <algoritmo1,algoritmo2,...> [**<n>**]

Na Tabela 9 são apresentadas as opções que podem ser utilizadas na execução do *script regression.pl* e seus respectivos parâmetros adicionais.

Opção	**Parâmetros Obrigatórios**	**Parâmetros Opcionais**
-convert	<algoritmo>	—
-semantic	—	—
-measures	—	<arquivo de teste>
-table	—	<número de pseudo-intervalos> <arquivo de teste>
-evaluation	<algoritmo>	<número de partições>
-hypothesis	<algoritmo1,algoritmo2,...>	<número de partições>

Tabela 9: Opções para executar o *script regression.pl*

5 Considerações Finais

A extração dos padrões em mineração de dados não é a etapa final desse processo. A etapa de pós-processamento visa avaliar esses padrões segundo alguns aspectos, como precisão, compreensibilidade e interessabilidade. O ambiente computacional DiPER apresentado neste relatório técnico tem como objetivo auxiliar na avaliação de um modelo de regressão simbólico, oferecendo, para isso, uma série de métodos a serem utilizados na etapa de pós-processamento do processo de mineração de dados. O DiPER é composto por uma biblioteca de classes e foi implementado de acordo com as especificações do Discover, um projeto que está sendo desenvolvido em nosso laboratório de pesquisa (LABIC) com o objetivo de fornecer um ambiente integrado para apoiar as etapas do processo de descoberta do conhecimento. As classes do ambiente DiPER são apresentadas na Figura 5.

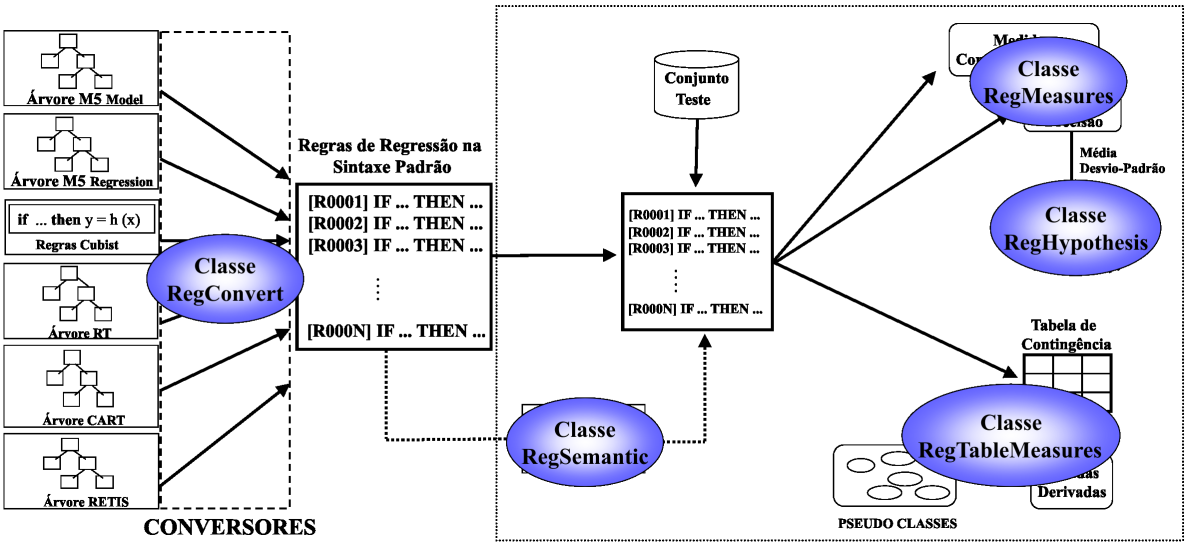

Figura 5: Classes do ambiente DiPER

Os modelos de regressão simbólicos, sejam eles regras ou árvores, gerados por diferentes algoritmos de regressão, são representados de diferentes formas. Assim, qualquer avaliação que precise ser efetuada sobre uma hipótese (cálculo de medidas, estatísticas, entre outras) deve ser feita diversas vezes, uma para cada algoritmo. Nesse sentido, baseada na sintaxe padrão de regras de regressão do Discover (Pugliesi, Dosualdo, & Rezende 2003), a classe **RegConvert** converte a saída dos algoritmos de regressão RT, CART, RETIS, Cubist e M5 para esse formato padrão de regras de regressão.

Mesmo com as regras padronizadas em um único formato, podem ocorrer ainda sobreposições de condições em uma determinada regra, diminuindo a compreensibilidade dessa regra e acarretando em uma queda no desempenho nos cálculos efetuados sobre o arquivo de regras. A classe **RegSemantic** realiza um tratamento semântico sobre o conjunto de regras com o objetivo de diminuir o número de condições das regras, sempre que possível.

Depois de gerado um modelo simbólico de regras, ele precisa ser avaliado para verificar alguns fatores, por exemplo, sua precisão na predição do valor do atributo-meta de novos exemplos e sua compreensibilidade. A classe **RegMeasures** calcula algumas medidas de precisão (baseadas na diferença entre o valor real do atributo e o valor predito pelo algoritmo) e algumas medidas de compreensibilidade para cada regra do modelo, além de calcular a precisão do modelo como um todo.

Porém, existem poucas medidas que podem ser calculadas para os problemas de regressão.

Nesse sentido, foi apresentado neste relatório técnico o procedimento definido em (Dosualdo 2003), que possibilita construir uma tabela de contingência para regras de regressão. A tabela de contingência é tipicamente calculada para regras de decisão, e a partir dela, são definidas uma série de medidas. Assim, por intermédio da classe **RegTableMeasures** é possível construir uma tabela de contingência para uma regra de regressão e calcular as medidas dela derivadas.

No entanto, não se pode afirmar que um algoritmo é melhor que outro levando-se em consideração apenas medidas de erro, como as medidas MAD e MSE. Nesse sentido, a classe **RegHypothesis** apresentada neste relatório implementa um teste de hipóteses para comparar algoritmos dois a dois baseando-se em *cross-validation*, o que permite afirmar se um algoritmo é melhor que outro com um grau de confiança de 95%.

A última classe do ambiente DiPER descrita neste relatório técnico foi a classe **RegRoutines**, que fornece vários métodos utilizados pelas outras classes do ambiente.

Para interagir com o ambiente DiPER, o usuário pode invocar os métodos implementados nas classes do ambiente a partir de seus próprios programas, ou utilizar o *script regression.pl*, que atua como uma interface entre o ambiente DiPER e o usuário.

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.

Breiman, L., J. H. Friedman, C. J. Stone, & R. A. Olshen (1984). *Classification and Regression Trees*. Chapman & Hall / CRC.

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

Clark, P. & T. Niblett (1989). The CN2 Induction Algorithm. *Machine Learning 3*(4), 261–283. Disponível em `http://citeseer.nj.nec.com/clark89cn.html`. Acesso em 20/03/2003.

Dosualdo, D. G. (2003). Investigação de regressão no processo de mineração de dados. Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação - USP.

Dosualdo, D. G. & S. O. Rezende (2003). Análise da Precisão de Métodos de Regressão. Relatório Técnico do ICMC - Número 197. Disponível em: `ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_197.zip`.

Fayyad, U. M., G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI Press.

Inmon, W. H. (1997). *Como Construir o Data Warehouse* (2 ed.). Rio de Janeiro: Editora Campus.

Karalic, A. (1995). RETIS - a Machine Learning System. Disponível em `http://www-ai.ijs.si/AramKaralic/retis/index.html`. Acesso em 07/10/2001.

Lavrac, N., P. Flach, & B. Zupan (1999). Rule Evaluation Measures: A Unifying View. In S. Džeroski & P. Flach (Eds.), *Ninth International Workshop on Inductive Logic Programming (ILP'99)*, Volume 1634 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag. Disponível em `http://link.springer.de/link/service/series/0558/papers/1634/16340174.pdf`.

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.

Melanda, E. A. (2002). Pós-processamento de Conhecimento de Regras de Associação. Monografia do Exame de Qualificação de Doutorado, Instituto de Ciências Matemáticas e de Computação - USP.

Merz, C. J. (1998). *Classification and Regression by Combining Models*. Tese de Doutorado, University of California, Irvine. Disponível em: `http://www.ics.uci.edu:80/~cmerz/thesis.ps`.

Monard, M. C. & J. A. Baranauskas (2003). *Sistemas Inteligentes: Fundamentos e Aplicações* (1 ed.)., Chapter Conceitos sobre Aprendizado de Máquina, pp. 89–114. Barueri, SP: Manole.

Prati, R. C., J. A. Baranauskas, & M. C. Monard (2001). 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.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip`.

Pugliesi, J. B., D. G. Dosualdo, & S. O. Rezende (2003). Sintaxe Padrão para Representar Regras de Regressão no Discover. Relatório Técnico do ICMC - Número 193. Disponível em: `ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_193.zip`.

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

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rulequest-Research (2001). An Overview of Cubist. Disponível em: `http://www.rulequest.com/cubist-unix.html`. Acesso em 20/03/2003.

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

Torgo, L. (1995). Data Fitting with Rule-based Regression. In J. Z. . P. Brazdil (Ed.), *Proceedings of the Workshop on Artificial Intelligence Techniques (AIT'95)*, Brno, Czech Republic. Disponível em `http://www.ncc.up.pt/~ltorgo/Papers/DFRBR/DFRBR.html`.

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.

Uysal, I. & H. A. Güvenir (1999). An Overview of Regression Techniques for Knowledge Discovery. *The Knowledge Engineering Review 14*(4), 319–340.

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`. Acesso em 20/03/2003.

Witten, I. H. & E. Frank (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. San Francisco, California: Morgan Kaufmann.