UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Mineração de Regras de Classificação com Sistemas de Banco de Dados Objeto-Relacional

Estudo de Caso:
Regras de Classificação de Litofácies de Poços de Petróleo

Benitz de Souza Vasconcelos

Campina Grande – Paraíba – Brasil
Dezembro – 2002

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Mineração de Regras de Classificação com Sistemas de Banco de Dados Objeto-Relacional

Estudo de Caso:
Regras de Classificação de Litofácies de Poços de Petróleo

Benitz de Souza Vasconcelos

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-graduação em Informática da Universidade Federal de Campina Grande, como parte integrante dos requisitos necessários para a obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Campina Grande – Paraíba – Brasil
Dezembro – 2002

VASCONCELOS, Benitz de Souza

V331M

Mineração de Regras de Classificação com Sistemas de Banco de Dados Objeto-Relacional**.** Estudo de Caso: Regras de Classificação de Litofácies de Poços de Petróleo.

Dissertação (Mestrado), Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande – Pb, Dezembro de 2002.
127 p. Il.

Orientador: Dr. Marcus Costa Sampaio

Palavras Chave:

1. Banco de Dados
2. Mineração de Dados
3. Regras de Classificação
4. Petróleo

CDU - 681.3.07B

"Talvez não tenhamos conseguido fazer o melhor, mas lutamos para que o melhor fosse feito... Não somos o que deveríamos ser, não somos o que iremos ser. Mas, graças a Deus, não somos o que éramos ..."

Martin Luther King

Agradecimentos

Ao apoio financeiro recebido da Agência Nacional do Petróleo – ANP – e da Financiadora de Estudos e Projetos – FINEP – por meio do Programa de Recursos Humanos da ANP para o Setor Petróleo e Gás – PRH-ANP/MME/MCT.

Ao professor Marcus Costa Sampaio, pela paciência e excelente orientação dedicadas à construção deste trabalho.

Aos professores e funcionários do Departamento de Sistemas e Computação da Universidade Federal de Campina Grande.

À coordenação do Programa de Recursos Humanos da Agência Nacional de Petróleo na Universidade Federal de Campina Grande (PRH25).

Sumário

Lista de Abreviaturas e Siglas

ANP	— Agência Nacional do Petróleo;
BD	— Banco de Dados;
BLOB	— ‘Binary Large Object’;
DM	— ‘Data Mining’;
FINEP	— Financiadora de Estudos e Projetos;
ID3	— ‘Induction of Decision Trees’;
MCT	— Ministério de Ciência e Tecnologia;
MD	— Mineração de Dados;
MME	— Ministério das Minas e Energia;
ODBC	— ‘Open Database Connectivity Bridge Driver’;
ODL	— ‘Object Definition Language’;
ODMG	— ‘Object Data Management Group’;
OO	— Orientado a objeto;
OQL	— ‘Object Query Language’;
OR	— Objeto-Relacional;
PL/SQL	— ‘Programming Language/Structured Query Language’;
PRH	— Programa de Recursos Humanos;
PRISM	— Algoritmo PRISM na sua forma original;
PrismJ	— Algoritmo PRISM na sua forma de acesso em arquivos, implementado em linguagem Java;
PrismOO	— Algoritmo PRISM na sua forma orientada a objeto;
PrismOR	— Procedimento que implementa o algoritmo PRISM na sua forma objeto-relacional;
PrismR	— Procedimento que implementa o algoritmo PRISM na sua forma relacional;
R	— Relacional;
SGBD	— Sistema de Gerência de Banco de Dados;

SGBDOO — Sistema de Gerência de Banco de Dados Orientado a Objeto;
SGBDOR — Sistema de Gerência de Banco de Dados Obejto-Relacional;
SGBDR — Sistema de Gerência de Banco de Dados Relacional;
SQL — ‘Structured Query Language’;
TDIDT — ‘Top-Down Induction of Decision Trees’;
UDF — ‘User Defined Function’;
WEKA — ‘Waikato Environment for Knowledge Analysis’;

Lista de Figuras

Lista de Tabelas

Lista de Gráficos

Resumo

Em aplicações de mineração de dados, a qualidade do conhecimento inferido é proporcional ao volume de dados a minerar. Levando isso em conta, a integração de algoritmos de mineração de dados com SGBDs vem sendo intensivamente pesquisada, haja vista a capacidade dos SGBDs de gerenciar grandes volumes de dados. Infelizmente, os esforços de integração têm se concentrado principalmente em conhecimento sob a forma de regras de associação, em detrimento de outros modelos de conhecimento. Esta dissertação, descreve como integrar com SGBDs o algoritmo PRISM. PRISM é um algoritmo de inferência de regras de classificação, muitas vezes mais simples e confiáveis que as indiretamente inferidas por algoritmos de árvores de decisão. A integração do PRISM se dá sob dois enfoques: relacional (PrismR) e objeto-relacional (PrismOR). São mostrados resultados de testes comparativos de desempenho do PrismR, do PrismOR, e do PrismJ, versão “stand-alone” em Java do PRISM.

Como em muitos outros domínios, a mineração de dados está se tornando crucial na exploração e produção de petróleo. Na indústria petrolífera, os custos típicos de perfuração de um novo poço para exploração de novos campos são da ordem de milhões de dólares. Porém, a chance de que se obtenha um sucesso econômico é, às vezes, de 1 em 10 poços perfurados. Avanços recentes na tecnologia de perfuração e nos métodos de coleta de dados têm levado as indústrias petrolíferas a coletarem grandes quantidades de dados geológicos/geofísicos de locais onde se encontram poços em produção e exploração. Essas informações encontram-se armazenadas nos grandes bancos de dados dessas companhias. Os dados históricos de campos explorados podem ser utilizados para derivar relações entre os parâmetros observados que diretamente contribuam para o aumento no sucesso na descoberta das reservas de óleo e gás natural.

Abstract

In Data Mining applications, the quality of the induced knowledge is proportional to the volume of data available for mining. Taking this into account, the integration of Data Mining algorithms with DBMSs has been intensively researched, considering the capability of DBMSs of managing large volumes of data. Unfortunately, integration efforts have concentrated mainly on the shape of association rules, to the detriment of other knowledge models. This work describes how to tightly integrate the PRISM algorithm with DBMS. PRISM is an induction algorithm for classification rules that are often more simple and more reliable than those indirectly induced from decision tree algorithms. The integration is made using two approaches: relational (RPrism) and object-relational (ORPrism). Comparative performance tests are shown between RPrism, ORPrism and with stand-alone Java version of Prism, JPrism.

Like a number of other domains, database mining is becoming crucial in oil exploration and production. It is common knowledge in the oil industry that the typical cost of drilling a new offshore well is in the range of millions, but the chance of that site being an economic success is 1 in 10. Recent advances in drilling technology and data collection methods have led to oil companies and their ancillaries collecting large amounts of geophysical/geological data from production wells and exploration sites, and then organizing them into large databases. The historical data of explored fields can be used to derive relationships among the parameters observed that directly contribute to the increase in the success in the discovery of the oil and natural gas basin.

Capítulo 1

Introdução

"Computers have promised us a fountain of wisdom,
but delivered us a flood of data."
W. J. Frawley

1.1 O Programa de Recursos Humanos da Agência Nacional de Petróleo

Este trabalho de dissertação de mestrado foi desenvolvido no contexto do Programa Interdepartamental de Tecnologia em Petróleo e Gás, PRH (25). Esse programa tem como objetivo a formação de profissionais especializados em petróleo e gás e faz parte do Programa de Recursos Humanos da Agência Nacional do Petróleo (ANP) para o setor Petróleo e Gás, (PRH-ANP/ MCT).

O PRH-ANP/MCT, voltado para o nível superior, é uma parceria entre o Ministério da Ciência e Tecnologia e universidades brasileiras, cujo objetivo é viabilizar e incentivar a criação de programas específicos para formação de pessoal na área de petróleo e gás natural nos níveis de graduação, mestrado e doutorado. Para tanto, o PRH-ANP/MCT concede apoio financeiro às instituições de ensino e bolsas de estudos aos alunos previamente selecionados.

A Universidade Federal de Campina Grande, através do PRH (25), recebeu nos últimos 2 anos cerca de 56 bolsas distribuídas pelos mais variados cursos, dentre os quais destacamos o curso de Pós-graduação em Informática, no qual este trabalho foi desenvolvido. De acordo com cada curso existem diferentes

especializações. Este projeto se encontra associado à especialização intitulada Engenharia do Conhecimento, cujo principal objetivo é o desenvolvimento de ferramentas avançadas de informática para inferir conhecimento “escondido” nos mais diversos bancos de dados.

Para a realização deste trabalho, temos à disposição o CD-ROM intitulado Campo Escola de Namorado, disponibilizado pela Agência Nacional do Petróleo, do qual retiramos dados para a realização de um estudo de caso. O estudo de caso com dados reais visa gerar conhecimento para a identificação automática de litofácies de poços de petróleo, utilizando técnicas avançadas de Mineração de Dados.

1.2 Contexto

1.2.1 Descoberta de Conhecimento

Os bancos de dados constituem a memória dos sistemas de informação atuais. Mas, a memória tem pouco uso sem a inteligência. A nossa inteligência permite-nos a análise de nossa memória e a partir daí o estabelecimento de modelos, relações e formulação de novas idéias para fazer previsões sobre o futuro. Capacitar os bancos de dados, com algum mecanismo de inteligência, é fundamental para podermos inferir algum conhecimento dos dados por eles armazenados.

As formas naturais de transformar dados em conhecimento — dados organizados e processados para inerir compreensão, experiência e aprendizado acumulado — sempre estiveram associadas ao processo de análise e interpretação manual. As diversas ferramentas de análise dispõem de um método baseado na verificação, isto é, o usuário constrói hipóteses específicas e então as verifica ou as refuta, através do sistema. Esse modelo torna-se dependente da intuição e habilidade do analista em propor hipóteses interessantes, em manipular a complexidade do espaço de atributos, e em refinar a análise baseado-se nos resultados de consultas potencialmente complexas do banco de dados (Fayyad, 1997). Esta estratégia, ao longo do tempo, tornou-se proibitiva devido ao problema da “explosão dos dados”

(Fürnkranz, 1995). Como o número de possíveis relacionamentos entre dados de um banco de dados é muito grande, sua busca manual torna-se proibitiva. A necessidade de um mecanismo automatizado ou semi-automatizado de análise faz-se necessária. Atualmente, deseja-se que programas inteligentes devam ser capazes de sugerir hipóteses por conta própria, e apresentar ao usuário os resultados descobertos e inferidos. A validação final do conhecimento ainda é da responsabilidade do especialista do domínio da aplicação. Podemos então concluir que, no estágio atual, o processo de inferência de conhecimento é semi-automatizado.

A inferência de conhecimento tornou-se possível graças aos algoritmos de aprendizagem de máquina (‘machine learning’). Alguns desses algoritmos constituem o núcleo do processo chamado de Descoberta de Conhecimento em Banco de Dados (‘Knowledge Discovery in Databases’ — KDD). Essencialmente, o conhecimento inferido por esses algoritmos se dá sob a forma de *padrões* ou *modelos* de dados. Um padrão é uma expressão genérica formal descrevendo um subconjunto de dados, ou seja, induz-se um modelo a partir de um conjunto de dados de treinamento. Para ser útil, um padrão deve ser simples e compreensível. Os casos (ou exemplos) coletados são chamados de instâncias e o conjunto destas instâncias constitui a base de dados. Cada instância contém um conjunto de valores que caracterizam certas propriedades da instância. Estas propriedades são denominadas atributos. Cada instância é descrita pelo mesmo conjunto de atributos.

Historicamente, a noção de encontrar padrões úteis nos dados tem sido dada uma variedade de nomes, incluindo mineração de dados, extração de conhecimento, descoberta de informação, arqueologia de dados, colheita de informação... O termo KDD foi utilizado pela primeira vez em 1989 para enfatizar que o conhecimento é o produto final de uma descoberta baseada nos dados. KDD refere-se a um processo geral de descoberta de conhecimento útil sendo a Mineração de Dados (‘Data Mining’) referente a um passo particular neste processo (aplicação de algoritmos específicos para extração de padrões dos dados). Os passos adicionais no processo de KDD mostrados na figura 1, tais como, preparação, seleção, limpeza, transformação dos dados e interpretação dos resultados da mineração é a garantia essencial de que o conhecimento é derivado dos dados. KDD tem envolvido e continua a envolver diferentes campos de pesquisas, tais como, aprendizagem de

máquina, reconhecimento de padrões, banco de dados, estatística, inteligência artificial, aquisição de conhecimento por sistemas especialistas, visualização de dados e computação de alta performance.

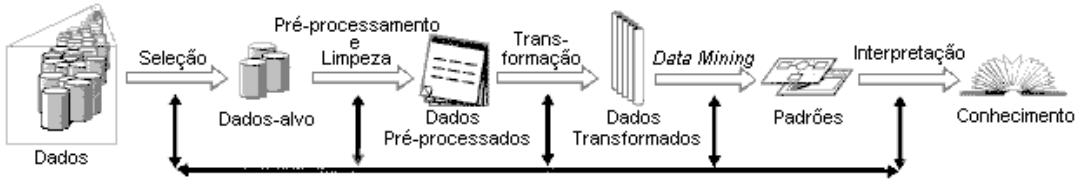


FIGURA 1. Os passos do Processo de KDD[1]

O processo de KDD começa com o entendimento do domínio da aplicação e dos objetivos finais a serem atingidos. A etapa da limpeza dos dados vem a seguir, através de um pré-processamento dos dados, visando adequá-los aos algoritmos. Isso se faz através da integração de dados heterogêneos, eliminação de incoerências dos dados, repetição de instâncias, etc. Os dados pré-processados devem ainda passar por uma transformação que os armazena adequadamente. Prosseguindo no processo, chega-se à fase de 'Data Mining' especificamente, que começa com a escolha dos algoritmos a serem aplicados. Ao final do processo, o relatório das descobertas é gerado e passa então a ser interpretado pelos analistas. Somente após a interpretação das informações obtidas encontramos *conhecimento*.

1.2.2 Mineração de Dados

Mineração de Dados (MD) é o coração do processo de KDD, situado entre a preparação de dados e a interpretação dos resultados. Este passo do processo caracteriza-se pela escolha e aplicação do algoritmo ou algoritmos de MD, de acordo com o método adequado ao problema que se está resolvendo. Existe uma grande variedade de algoritmos de mineração, inferindo conhecimento como *regras de classificação* ou *árvores de decisão*[2], ou então, como *regras de associação*, além de outros padrões de conhecimento (Fayyad et al., 1996b; Ganti et al., 1999). Classificação é o processo que encontra propriedades comuns entre um conjunto de

[1] www.sims.berkeley.edu/~hearst

instâncias num banco de dados e classifica-os em diferentes classes, de acordo com um modelo de classificação. Para construir tal modelo, uma base de dados exemplo, ‘E’, é tratada como o conjunto de treinamento, na qual cada instância consiste do mesmo conjunto de múltiplos atributos que as instâncias num grande banco de dados, ‘D’, e adicionalmente cada instância tem um identificador de classe a ele associado. O objetivo da classificação é primeiro analisar os dados de treinamento e desenvolver uma descrição acurada ou um modelo para cada classe usando características disponíveis nos dados. Tais descrições de classes são então utilizadas para classificar dados de teste futuros no banco de dados ‘D’ ou desenvolver uma descrição melhor para cada classe no banco de dados. A descrição construída é utilizada para predizer classes a partir de novos casos que não apresentem o valor do atributo de classificação expresso. Há vários métodos para tratar-se do problema de classificação, as alternativas mais comumente usadas são classificação via indução de árvores de decisão ou através de regras de classificação *strictu sensu*. Uma regra de classificação obedece ao formato — SE *antecedente* ENTÃO *conseqüente* — em que o conseqüente é composto de apenas um par “atributo = valor”. Um classificador é um conjunto de regras de classificação em que o atributo do conseqüente é sempre o mesmo.

Regras de associação diferenciam-se das regras de classificação por não eleger um de seus atributos como alvo de classificação (fixar um atributo para o conseqüente da regra). Ou seja, todos os atributos em regras de associação podem aparecer tanto no antecedente quanto no conseqüente da regra. Outra característica diferenciadora é a criação de um classificador que enquadrará um exemplo dentro de uma das classes existentes, em regras de associação não é realizado nenhum tipo de classificação, mas sim a extração de associações existentes entre os itens de uma base de dados. Dada a sua utilidade em aplicações de comércio varejista, o conhecimento sob a forma de regras de associação tem despertado um esforço concentrado de pesquisa e desenvolvimento. Entretanto, outras categorias de aplicação, como ‘Customer Relations Management’ (CRM), de grande interesse em comércio eletrônico ‘business-to-consumer’, estão a demandar a utilização intensiva

[2] - Árvores de decisão são formas indiretas de regras de classificação.

de algoritmos de regras de classificação. Além disso, mineração de dados é uma necessidade premente na exploração e produção de petróleo: avanços recentes na tecnologia de perfuração e nos métodos de coleta de dados têm levado as indústrias petrolíferas a armazenar grandes volumes de dados geológicos; a inferência de regras de classificação de dados geológicos pode contribuir eficazmente para aumentar as chances de sucesso na procura de reservas de óleo e gás natural, reduzindo os custos de prospecção e perfuração de novos poços.

Dentre alguns algoritmos que induzem conhecimento diretamente no formato de regras de classificação proposicionais, baseados em pares atributo-valor (SE-ENTÃO), destacamos: AQ (Michalski, 1969), PRISM (Cendrowska, 1987), RIPPER (Cohen, 1995) dentre outros. Utilizou-se neste trabalho o algoritmo PRISM (Cendrowska, 1987), que é um dos algoritmos que implementa de maneira simples e direta a estratégia separar-e-conquistar para indução de regras de classificação. Também, pelo fato do conhecimento ser gerado num formato inteligível para o especialista do domínio da aplicação que irá analisar as regras induzidas. Em (Bramer, 2000), é demonstrado que PRISM pode inferir regras mais simples e confiáveis do que as inferidas por algoritmos da família TDIDT (‘Top-Down Induction of Decision Trees’), de árvores de decisão (Quinlan, 1993).

1.2.3 Sistemas de Banco de Dados Objeto-Relacional

A viabilização de algoritmos de mineração de dados praticamente impõe que eles sejam implementados estreitamente integrados com Sistemas de Gerência de Banco de Dados (SGBDs), dada a extensa gama de recursos que esses dispõem para gerenciar grandes volumes de dados.

A maior parte dos SGBDs utilizados nos últimos anos fundamenta-se no modelo relacional (SGBDR). No entanto, SGBDs baseados em outros modelos têm surgido devido à demanda de novas aplicações. São exemplos destes novos sistemas, os SGBDs orientados a objeto (SGBDOO), e os SGBDs objeto-relacional (SGBDOR).

O modelo relacional, que foi introduzido por E. F. Codd em 1970, serviu como alicerce para a indústria de bancos de dados atual. Baseia-se em uma estrutura de dados simples e tabular, representando um banco de dados na forma de um conjunto de tabelas, podendo ser implicitamente relacionadas através de atributos com os mesmos domínio e semântica. O modelo é amplamente difundido, com uma base firmemente estabelecida no mundo das aplicações de bancos de dados e contando com numerosos produtos — SGBDs — de pequeno, médio e grande porte.

Nos SGBDs relacionais, os bancos de dados geralmente são estritamente normalizados, o que resulta em tabelas freqüentemente muito fragmentadas. Entretanto, fragmentação não é desejável em mineração de dados que, em última análise, busca padrões em dados que se repetem com freqüência. Com o advento dos novos SGBDs, pelo menos a primeira forma normal não é mais um requisito de projeto: sendo assim, podemos afirmar que os SGBDS, OO e OR, são potencialmente muito mais adequados à mineração de dados que os SGBDs relacionais.

SGBDOOs compartilham as potencialidades das linguagens orientadas a objeto incluindo: encapsulamento, reutilização de código, mecanismos de herança e polimorfismo. Esses sistemas podem armazenar objetos complexos definidos pelos usuários e podem capturar o comportamento, bem como o estado desses objetos. SGBDOOs estão se tornando populares — ainda que sejam muitas vezes considerados revolucionários em relação ao relacional —, devido à riqueza da semântica e das estruturas de dados que permitem modelar, como objetos de dados complexos, hierarquias de classes/subclasses ou métodos.

Ambos, sistemas relacional e orientado a objeto, têm vantagens importantes. Os desenvolvedores de aplicações tendem a escolher um sistema de banco de dados dependendo de qual conjunto de vantagens é mais importante para a aplicação específica. Recentemente, as aplicações buscam tanto os benefícios conseguidos ao longo de décadas pela tecnologia relacional (e.g. linguagem de consulta padronizada e conhecida), como as potencialidades fornecidas pela tecnologia orientada a objeto (e.g. semântica definida pelo usuário). Parece não haver razões, em princípio, para que esses conjuntos de vantagens não possam ser

combinados. Tentativas para permitir essa combinação vêm sendo chamadas de Sistemas de Bancos de Dados Objeto-Relacional.

SGBDORs dão suporte a um modelo relacional estendido com certos aspectos de orientação a objeto. A tecnologia objeto-relacional é evolucionária, herda características de gerência de performance da tecnologia relacional e adiciona a flexibilidade da tecnologia orientada a objeto. Pode-se trabalhar com estruturas tabulares familiares, enquanto adicionam-se novas possibilidades de gerência de objetos. Objetiva-se com isso a possibilidade de fornecer um sistema de tipos mais rico — através da inclusão de características de orientação a objetos — e adicionar construções às linguagens de consultas relacionais, tal como SQL, para manipular os novos tipos de dados adicionados. Tais extensões, tentam preservar os fundamentos relacionais, enquanto estendem o poder de modelagem dos dados.

A semântica definida pelo usuário torna-se possível através da definição de novos tipos de dados. Estes podem incluir tipos simples, como também, tipos estruturados tais como, tipos abstratos de dados e coleções (e.g. conjuntos, listas, arranjos etc.). Alguns dos sistemas OR suportam mecanismos de herança de tipos, permitindo a definição de um novo tipo de dado como especialização de um ou mais tipos existentes. Cada novo tipo de dados requer um construtor, que provê um modo para criar instâncias do tipo.

Desde que a essência de um tipo de dados reside em seu comportamento, um sistema OR deve prover algum meio pelo qual o comportamento de um tipo possa ser especificado. Isso é geralmente feito criando-se métodos que operam sobre instâncias de tipos e fazem computações baseadas nos valores dos seus atributos. Os métodos correspondem ao comportamento dos objetos, implementando as operações associadas ao tipo. Sistemas que suportam mecanismos de herança geralmente também dão suporte a polimorfismo, que envolve a seleção em tempo de execução de funções baseadas nos argumentos dos tipos dinâmicos. Por exemplo, a função denominada função(x) pode invocar diferentes funções dependendo do tipo do argumento *x*. Entretanto, seu potencial não tem sido ainda convenientemente explorado. A riqueza de semântica dos SGBDOR traz poder e maleabilidade ao sistema.

Agregando-se às tecnologias previamente descritas, surge a tecnologia de banco de dados multidimensionais, emergindo como fator chave na análise interativa de grandes quantidades de dados para propósitos de tomadas de decisão. Os bancos de dados construídos de acordo com este paradigma visualizam os dados como "cubos" multidimensionais (generalização de tabelas para um número maior de perspectivas/dimensões) que são particularmente mais convenientes para análise de dados. Modelos de dados multidimensionais possuem três importantes áreas de aplicação na análise de dados: (i) ‘Data Warehouses’ (DW) que são grandes repositórios centralizados que integram dados de várias fontes para análise de consultas de grande complexidade a dados integrados; (ii) ‘On-line Analytical Processing’ (OLAP) que são ferramentas que fazem uso da infra-estrutura do DW para prover suporte necessário às consultas dos usuários e (iii) ‘Data Mining’. Funções de mineração sobre banco de dados multidimensioanis geralmente custam mais que operações simples de OLAP. Implementações eficientes e respostas rápidas são o maior desafio na realização de análise de mineração on-line (On-line Analytical Mining — OLAM) em grandes bancos de dados ou DWs. Assim, um mecanismo de OLAM é mais sofisticado que um mecanismo de OLAP uma vez que ele geralmente consiste de múltiplos módulos de mineração os quais podem interagir uns com os outros para uma mineração efetiva.

1.3 Objetivos da Dissertação

O objetivo geral deste trabalho foi integrar o algoritmo de classificação PRISM com SGBDs, sob dois enfoques: relacional (versão PrismR do PRISM) e objeto-relacional (versão PrismOR do PRISM). Para atingir este objetivo geral, foram definidos os seguintes objetivos específicos:

- Versão PrismOO do PRISM.

Como primeira etapa para a integração do PRISM a um SGBD, uma especificação orientada a objeto e genérica (isto é, independentemente da semântica

e da sintaxe de qualquer SGBD particular) do PRISM — PrismOO, foi construída. A especificação foi definida de acordo com as linguagens ODL (‘Object Definition Language’) e OQL (‘Object Query Language’) do padrão ODMG (‘Object Data Management Group’) (Cattell et al., 2000). Estas linguagens permite-nos uma descrição analítica do PrismOO, enquanto que a linguagem de modelagem UML (Larman, 1998) permite uma descrição gráfica.

Tendo-se o PrismOO, pode-se então implementá-lo em qualquer SGBDOR, como o Oracle9i, o Informix Universal Server, ou o Universal IBM DB/2.

- Implementação do PrismOO integrado com o SGBDOR Oracle9i, e utilizando estruturas de dados objeto-relacional — PrismOR.

Através da utilização de uma estrutura de dados sob a forma de colunas não atômicas, integrar o algoritmo PRISM com o SGBDOR Oracle9i. A implementação foi feita utilizando PL/SQL como linguagem de programação. PL/SQL é a extensão ao SQL da Oracle adicionando construções procedurais, variáveis e outras funcionalidades ao SQL. O PL/SQL é projetado para ter uma íntima integração com o sistema de banco de dados Oracle e com a SQL.

- Implementação em PL/SQL do PrismOO integrado com o SGBDOR Oracle9i.

Através da utilização de uma estrutura de dados sob a forma de colunas atômicas, puramente relacionais, integrar o algoritmo PRISM com o SGBDOR Oracle9i.

- Implementação ‘stand-alone’ do PrismOO em Java — PrismJ.

Através da utilização de arquivos de dados no formato ASCII para armazenamento do conjunto de dados, implementar o algoritmo PRISM em linguagem Java seguindo uma forma de arquitetura não integrada a nenhum SGBD. Objetiva-se com essa implementação lidar com o problema de bases de dados de considerável tamanho que por ventura não possam ser trazidas para memória principal num único momento.

- Análise de desempenho das versões PrismOR, PrismR e PrismJ do PRISM.

Analisar a viabilidade da integração do algoritmo PRISM com o SGBDOR Oracle9i, do ponto de vista do custo de processamento do algoritmo. Para isso, as versões relacional e objeto-relacional do PRISM, ambas estreitamente acopladas ao Oracle9i, foram comparadas com o desempenho da versão "stand-alone" PrismJ, do PRISM.

Investigar, também, em que medida uma estrutura sob a forma de tabela com colunas não-atômicas pode ser melhor para o desempenho do PRISM do que uma estrutura sob a forma de tabela puramente relacional ou com colunas atômicas, para isso as versões R e OR do PRISM foram comparadas entre si.

- Testes do PRISM, utilizando um conjunto de treinamento real.

Processamos o algoritmo PRISM com uma base de dados geológicos do Campo Escola de Namorado, fornecida pela Petrobrás, para a inferência de regras gerais de classificação de litofácies de poços de petróleo.

1.4 Relevância da Dissertação

Este trabalho contribui com uma solução para a integração de um algoritmo de regras de classificação, PRISM, com SGBDs. Trabalhos com o mesmo fim (integração) têm se concentrado em algoritmos de regras de associação (Agrawal et al., 1996; Sarawagi et al., 1998; Thomas et al., 1998; Rajamani et al., 1999; Yoshizawa et al., 2000; Hipp et al., 2001; Tok et al., 2002). Outros (Agrawal et al.,1992; Mehta et al., 1996; Shaffer et al., 1996; Wang et al., 1998; Zaki et al., 1999; Onoda et al., 2001) tratam da integração com SGBDs de algoritmos de regras de classificação, na forma intermediária de árvores de decisão. Problemas com árvores de decisão foram convincentemente mostrados em (Cendrowska, 1987; Bramer,

2000), entre eles, destacamos: regras muitas vezes complexas, e a restrição de que o atributo raiz de uma árvore tenha que aparecer em todas as regras inferidas.

Até onde vai o nosso conhecimento, nossa abordagem de integração, utilizando um algoritmo de regras de classificação e a tecnologia objeto-relacional, é pioneira — ver também (Vasconcelos et al., 2002).

1.5 Estrutura da Dissertação

Esta dissertação encontra-se organizada em sete capítulos, incluindo esta introdução. O capítulo II apresenta uma revisão da literatura existente sobre trabalhos relacionados ao tema de integração de algoritmos de mineração de dados com SGBDs.

O capítulo III aborda o tema de mineração de dados com regras de classificação. O algoritmo de indução de regra de classificação, PRISM, é descrito. Comparações entre este algoritmo e algoritmos da família TDIDT são feitas.

O capítulo IV mostra as modelagens conceitual e lógica referentes à integração do algoritmo PRISM com SGBDs, contemplando também a sintonização do PRISM com o SGBD Oracle9i. Apresenta também a versão ‘stand-alone’ em Java do PRISM, PrismJ.

Para a validação deste trabalho, o capítulo V é dedicado à análise dos resultados dos testes comparativos do PrismR, do PrismOR e do PrismJ com diversas bases de dados.

O capítulo VI é dedicado ao estudo de caso realizado sobre a base de dados geológicos do Campo Escola de Namorado, da Petrobrás, com a finalidade de indução de regras de classificação de litofácies de poços de petróleo.

Finalmente, o capítulo VII apresenta as conclusões e perspectivas do trabalho.

Capítulo

2

Trabalhos Relacionados

"Study the past, if you would
devine the future"
Confúcio

Este capítulo mostra a evolução das arquiteturas de integração de algoritmos de mineração de dados com SGBDs.

2.1 Integração Mineração de Dados — SGBDs

Correntemente, os sistemas de mineração são baseados principalmente em sistemas de arquivos ‘stand-alone’, estruturas de dados especializadas, e estratégias locais de gerência de ‘buffers’. No máximo, os dados para mineração são importados de um SGBD e armazenados localmente (‘cache-mining’). Dessa forma, elimina-se a necessidade de recuperar dados várias vezes do SGBD, melhorando o desempenho da aplicação. Os mecanismos para a importação de dados baseiam-se em protocolos ODBC (‘Open Database Connectivity Bridge Driver’), e na utilização de cursores da linguagem SQL. O processo é adequadamente alcunhado de “Mineração de Arquivos” (Imielinski et al., 1996), apresentando limitações no manuseio de arquivos maiores que a memória principal (quase todos os algoritmos de mineração de dados assumem que os dados residem em memória principal).

Entretanto, a realidade é que se faz mineração tipicamente com grandes volumes de dados, praticamente sob a gerência de SGBDs. Dessa forma, é natural que uma aplicação de mineração de dados possa ser pensada também como uma aplicação de banco de dados.

Há vantagens potenciais significativas de integrar aplicações de mineração de dados nos SGBDs: (1) funções de mineração de dados fatoradas no servidor de dados/aplicações — servidor de BD — podem ser compartilhadas por todas as aplicações, em benefício da produtividade dos programadores; (2) a execução de código no servidor de BD pode reduzir o tráfego de dados na rede, uma vez que a lógica do negócio é processada no servidor, e apenas os resultados são enviados para a aplicação-cliente; (3) com a maturidade alcançada pelos SGBDs para armazenar, manter e recuperar informações de grandes bancos de dados, de forma eficiente, essas técnicas de "otimização" poderiam ser incorporadas aos algoritmos de mineração de dados.

Diversos trabalhos de pesquisa têm explorado o tema da integração mineração de dados – SGBD (MD-SGBD), surgindo assim diferentes alternativas de integração. Em (Bezerra et al., 2000) uma taxionomia de alternativas arquiteturais ('frameworks' de integração) é estabelecida. As arquiteturas dividem-se em fracamente acopladas, fortemente acopladas e "caixa-preta". Com integração fracamente acoplada, o SGBD serve apenas como repositório original dos dados a ser minerados. Entretanto, os dados devem passar por uma transformação, antes que eles possam ser acessados por algoritmos de mineração. Com integração fortemente acoplada, o SGBD atua não somente como fonte dos dados, mas também, como processador de algumas funcionalidades da aplicação. Por fim, o SGBD integra completamente toda a aplicação mineradora —integração "caixa-preta".

2.2 Mineração Fracamente Acoplada a SGBDs

Esta alternativa caracteriza-se pelo desenvolvimento de aplicações de banco de dados que usam a linguagem SQL fracamente acoplada a uma linguagem

de programação de propósito geral — linguagem hospedeira. A aplicação de mineração é implementada na linguagem de programação hospedeira na qual são inseridos comandos SQL. A aplicação utiliza comandos 'SELECT' do SQL para recuperar o conjunto de registros de interesse no banco de dados. Um laço no programa de aplicação recupera o conjunto de registros resultante, um por um, do espaço de endereçamento do banco de dados para o espaço de endereçamento da aplicação, no qual o processamento é realizado. A base de dados nunca é copiada para um sistema de arquivo no disco local.

A camada de interface reside no lado cliente e a aplicação de mineração pode residir no lado cliente (arquitetura em duas camadas) ou em um servidor de aplicação (arquitetura em três camadas). O SGBD é responsável apenas pelo processamento do código SQL. Todo o acesso é feito através de uma rede, resultando em alguns casos em problemas de desempenho. O modelo é relativamente simples e isto torna o projeto e a implementação do programa aplicativo relativamente fácil. No entanto, uma vez que há uma chamada ao servidor associada a cada comando SQL, pode haver um congestionamento no tráfego na rede.

Este método apresenta, dessa forma, dois problemas de performance: (i) cópia de registros do espaço de endereçamento do banco de dados para o espaço de endereçamento da aplicação através do uso de parâmetros, e (ii) troca de contexto de processos para cada registro recuperado, o que representa um alto custo para um sistema.

A figura 2 mostra o exemplo de uma aplicação simples fracamente acoplada (Agrawal et al., 1996). Esta aplicação recupera registros da tabela de vendas, e contabiliza quantos clientes compraram cada um dos itens vendidos. O esquema para a tabela é *vendas (tid, itemNo)*. Se dois itens *i1* e *i2* foram comprados por um cliente na transação *t1*, a tabela de vendas terá dois registros, *<t1,i1>* e *<t1,i2>*.

Neste exemplo, o comando SQL é precedido por *exec sql*. Referências a variáveis hospedeiras são prefixadas por dois pontos (:), para distingui-las dos nomes das colunas das tabelas (variáveis *tid* e *itemid* da linha 5 na figura 2). O status de execução de um comando SQL é retornado ao programa aplicação através do indicador *sqlcode*. O comando na linha 1 declara e inicializa o 'array' que mantém a

contagem do número de ocorrências de cada item na tabela de vendas. O comando SQL na linha 2 conecta a aplicação ao banco de dados no qual a tabela vendas está armazenada. Para acessar o conjunto de registros retornado por um comando 'SELECT', um mecanismo de iteração (cursor) é provido. O comando na linha 3 define o cursor para a consulta especificada no comando 'SELECT'.

```
Procedure AlgoritmoLC()
begin
1 declare e inicialize o array count[MAXSIZE];
2. exec sql connect to database;
3. exec sql declare cur cursor for
            select *
            from sales;
4. exec sql open cur;
5. exec sql fetch cur into :tid, :itemid;
6. while (sqlcode ≠ endOfRec) do {
7.       count[itemid] := count[itemid] + 1;
8.      exec sql fetch cur into :tid, :itemid;
9. }
10. exec sql close cur;
11. print count array;
end
```

FIGURA 2: Uma aplicação Loosely Coupled (Agrawal et al., 1996)

A consulta não é executada até que o cursor seja aberto pelo comando SQL na linha 4 do algoritmo. O comando de busca na linha 5 é usado para obter o próximo registro no conjunto de resultados da consulta. Campos recuperados do registro são copiados para variáveis hospedeiras especificadas na cláusula 'INTO'. A busca é feita dentro de um laço até que não haja mais registros no conjunto de resultados. Fato a ser observado é que a cada busca aos registros do banco de dados, resulta na cópia do registro do espaço de endereçamento do banco de dados para o espaço de endereçamento da aplicação e, consequentemente, na troca de contexto de processos, causando a degradação de performance. Terminado o laço de iterações o cursor é fechado.

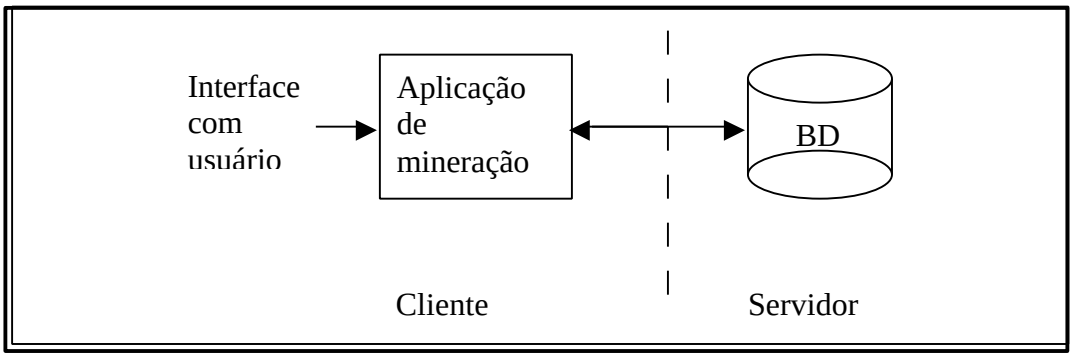


FIGURA 3. Acoplamento Fraco

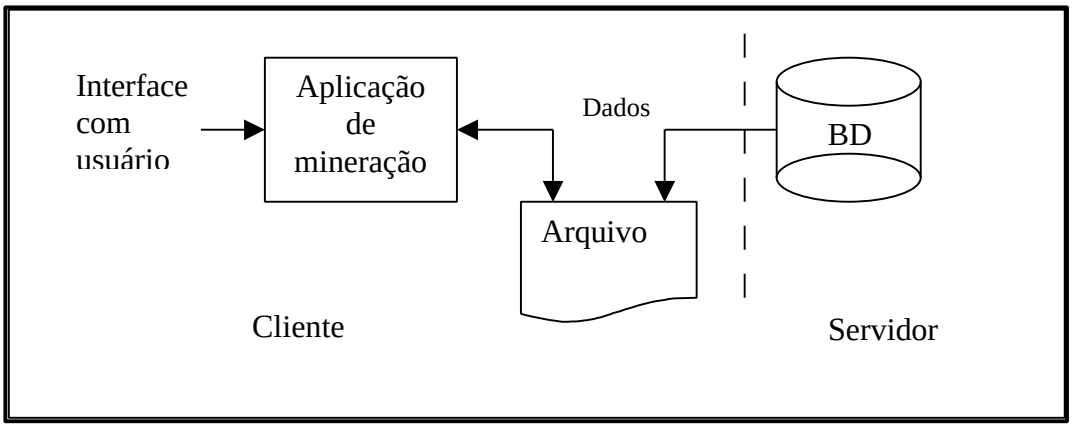


FIGURA 4. Acoplamento Fraco: alternativa 'cache-mining'

Uma variação dessa alternativa é o armazenamento temporário dos dados do SGBD em um 'buffer' no disco local, 'cache-mining' (Sarawagi et al., 1998). A base de dados é lida do banco de dados apenas uma vez e copiada em disco. Os dados armazenados podem ser transformados para um formato que possibilite acessos futuros eficientes e são descartados quando a execução é completada. A desvantagem é que ele requer espaço em disco adicional para armazenamento. Os dados permanentes continuam sendo gerenciados pelo SGBD. Tem como vantagem aumentar a eficiência de acesso aos dados depois de armazenados no 'buffer' local, porém há a necessidade de espaço extra em disco e consumo de tempo na preparação e na busca completa dos dados para o disco local.

2.3 Mineração Fortemente Acoplada a SGBDs

A tecnologia de SGDB oferece várias funcionalidades que a torna valiosa ao implementar aplicações de mineração de dados. Por exemplo, é possível trabalhar com conjuntos de dados consideravelmente maiores que a memória principal, uma vez que o SGBD por si só é o responsável pelo manuseio das informações, paginação e *swapping* quando necessário.

A idéia de executar computação definida pelo usuário dentro de SGBDs, evitando tráfego de rede desnecessário, tem sido uma grande preocupação no projeto de vários SGBDs. Uma conseqüência dessa preocupação é a possibilidade de utilizar procedimentos armazenados em bancos de dados comerciais. Por exemplo, o servidor Oracle oferece a possibilidade de implementar código procedural em PL/SQL (ou em Java) e armazená-lo dentro do próprio dicionário de dados, em uma forma pré-compilada.

Para uma aplicação fortemente acoplada, o objetivo não é retornar do servidor banco de dados cada vez que um registro é buscado. Deseja-se retornar apenas depois que todos os registros tenham sido processados. Alguns autores desenvolveram uma metodologia para uma integração fortemente acoplada de aplicações de mineração de dados e bancos de dados. Ao invés de trazer os registros do banco de dados para o programa de aplicação, insere-se, seletivamente, partes do programa aplicação que realizam a computação sobre registros recuperados no SGBD, evitando assim a degradação de performance ocorrida na categoria fracamente acoplada. Alguns experimentos práticos com um sistema de banco de dados relacional resultaram na proposta de que, sempre quando possível, a computação deve ser incluída dentro de comandos SQL (Agrawal et al., 1996). Assim, não apenas os dados, mas também a aplicação reside no servidor.

Esta categoria pode ser subdividida em duas abordagens: SQL *aware* e *extensions aware*. Na abordagem SQL *aware* (utilizada por Freitas, 1997), estas operações são mapeadas em SQL padrão (SQL-99), enquanto que na *extensions aware* as aplicações de mineração de dados utilizam extensões do banco de dados e extensões da SQL. A abordagem *extensions aware* agrega as seguintes opções:

- Funções internas: parte do algoritmo de mineração é expresso como uma coleção de procedimentos/funções, as quais são apropriadamente utilizadas em expressões SQL. No IBM DB2 recebem o nome de funções definidas pelo usuário (‘User Defined Function’, UDF), utilizado em (Agrawal et al., 1996), estendem e adicionam o suporte provido por funções disponíveis da SQL. No Oracle ocorrem sob a forma de trechos de códigos escritos em PL/SQL, ‘Stored Procedure’, utilizado em (Sousa et al., 1998). Em ambos os casos a execução é feita no mesmo espaço de endereçamento do SGBD.

- Tipos de dados: as aplicações de mineração utilizam alguns tipos de dados encontrados nos SGBDs Objeto-Relacional e Orientados a Objeto para melhorar o tempo de execução do algoritmo. Sarawagi (1998) faz uso de BLOB (‘Binary Large Objects’) na implementação de um algoritmo de regras de associação.

- Extensões do SQL: o SQL é estendido para permitir execuções de operações intensivas de dados no SGBD na forma de primitivas. Assim, as vantagens das extensões da SQL são utilizadas pelas aplicações para realizar suas tarefas de mineração. Uma vantagem da utilização de extensões da SQL sobre o SQL padrão (SQL *aware*) é que as mesmas informações, que poderiam ser obtidas através de uma seqüência de comandos SQL, podem ser obtidas através de um número menor de requisições ao SGBD. O SGBD encapsula todas as detalhadas consultas SQL para cada operação de mineração fornecendo um novo comando SQL. Outra vantagem é que como as operações intensivas sobre os dados são encapsuladas em uma primitiva no banco de dados, sua execução pode fazer uso de todas as potencialidades do SGBD, melhorando a eficiência. Usada em (Silva et al., 1999).

As alternativas *SQL aware* e a *extension aware SQL* têm como vantagem a disponibilidade dos mecanismos de indexação e processamento de consultas dos SGBDs que podem ser utilizados em sua plenitude.

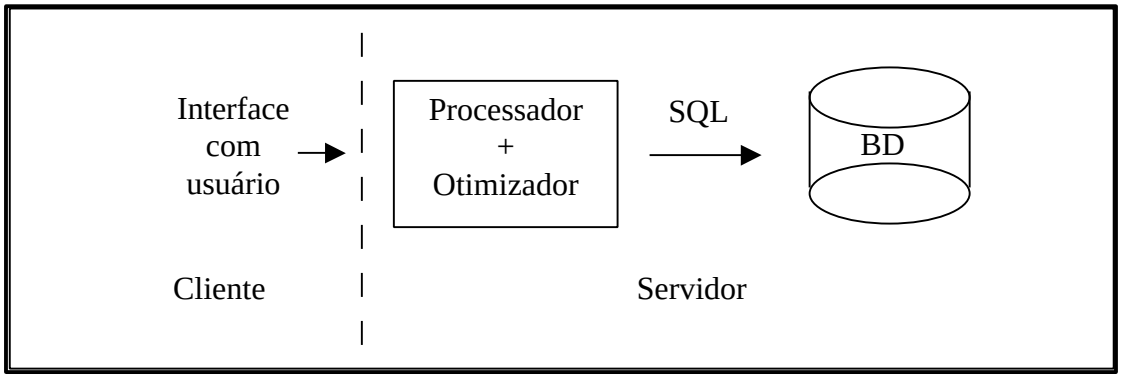


FIGURA 5. Arquitetura Fortemente Acoplada — SQL *Aware*

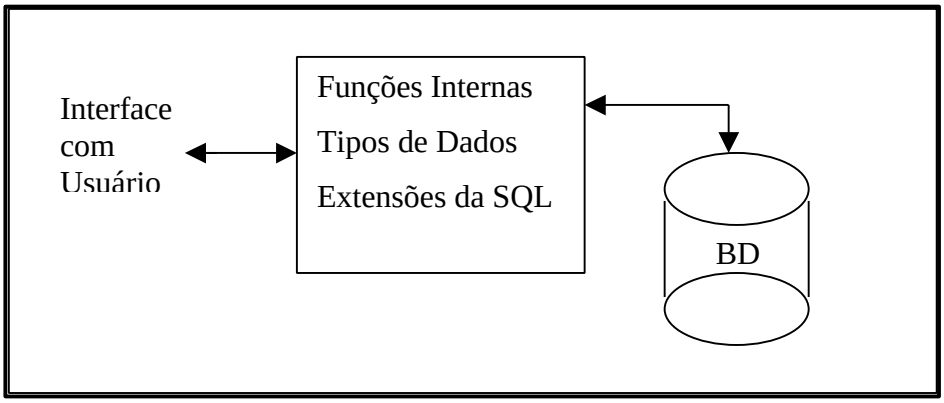


FIGURA 6. Arquitetura Fortemente Acoplada — *extension aware*

As figuras 5 e 6 mostram graficamente as abordagens SQL *aware* e *extension aware*, respectivamente. Na primeira, o algoritmo é mapeado em código SQL e a forma como a consulta é processada será determinada pelo otimizador de consultas, que escolhe o plano ótimo dentre os vários planos de execução que possam ser usados para processar a consulta. Na figura 6, requisições feitas pelos usuários através da passagem de parâmetros serão processadas no SGBD.

Na abordagem *extension aware*, figura 6, parte do algoritmo pode ser escrito como ‘Stored Procedure’ ou ‘User Defined Function’ (UDF), a aplicação utiliza-se das extensões disponibilizadas pelo SGBD e alguns tipos de dados encontrados nos SGBDs Objeto-Relacional e Orientados a Objeto.

2.4 Mineração com Acoplamento Forte: Alternativa “caixa-preta”

Na alternativa “caixa-preta”, que trata-se de um caso mais específico de acoplamento forte, uma linguagem de consulta de mineração de dados é definida. As operações de mineração são integradas ao SGBD e tornam-se parte das primitivas de consulta do banco de dados.

A aplicação envia uma simples requisição para o SGBD, que normalmente é uma expressão de consulta, solicitando a extração de algum conhecimento. A consulta é realizada especificando-se parâmetros, tais como, a fonte de dados na qual a operação de mineração especificada será aplicada e o tipo de conhecimento a ser gerado. O usuário pode chamar a operação de mineração executando o comando SQL correspondente com os devidos parâmetros. Neste caso, não há limites claros entre simples consultas e operações de mineração para os usuários.

No trecho de código da figura 7, um exemplo de uma consulta de mineração aninhada, na qual a consulta mais interna seleciona um conjunto de dados da tabela *trans* que satisfaz o critério de seleção. São mineradas as regras de associação deste subconjunto com um grau de suporte de 0.05 e de confiança de 0.7. Este é um exemplo da linguagem de consulta DMQL – ‘Data Mining Query Language’ (Han, 1996) que estende SQL com uma coleção de operadores para mineração de regras.

Esta alternativa apresenta a desvantagem de não permitir ao usuário especificar qual algoritmo de mineração é o mais indicado a ser aplicado na sua base de dados. Nenhum algoritmo de mineração é o mais adequado para todos os conjuntos de dados.

A figura 8 ilustra este modelo de arquitetura.

```
Find association rules
related to item
with key transaction_id
from (
        select *
        from trans
        where price > 15
              and price < 100) trans_new
with support    threshold = 0.05
with confidence threshold = 0.7
```

FIGURA 7. Exemplo de consulta

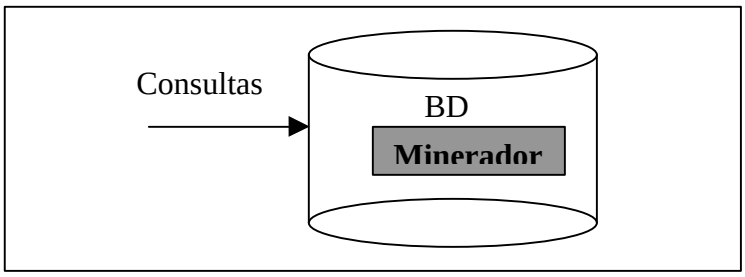


FIGURA 8. Arquitetura "Caixa Preta"

2.5 Conclusões

Nesta seção, apresentamos nossas conclusões sobre as diferentes alternativas de integração de algoritmos de mineração de dados com SGBDs.

Na alternativa fracamente acoplada, uma aplicação de mineração de dados não se beneficia das potencialidades disponibilizadas nos SGBDs, visto que ela não acessa diretamente o SGBD, e não é processada no núcleo do mesmo.

A melhor forma de integração é a fortemente acoplada, com os algoritmos sendo processados no lado-servidor ou no núcleo dos SGBDs. Desta forma, os algoritmos se beneficiam de todos os recursos de gerência de dados dos SGBDs, principalmente otimização de consulta.

Embora a alternativa "caixa-preta" seja uma forma particular de acoplamento estreito, o desenvolvedor de aplicações fica restrito a utilizar o algoritmo implementado no servidor.

2.6 Produtos Existentes

Uma vez que as tarefas e aplicações de mineração de dados são amplas e diversas, é esperado que vários tipos de interfaces flexíveis e interativas fossem surgindo. O SGBDOR Oracle9i (Oracle, 2001) integra dois algoritmos de mineração, sendo um de modelagem estatística, 'Naive Bayes' (Han et al., 2001), e o outro de regras de associação. Seguindo a mesma linha, o Microsoft SQL Server 2000 integra duas classes de algoritmos (classificação e 'clustering') (Netz et al., 2000). Sendo em todos os casos acoplamento do tipo "caixa preta", padecem com a desvantagem de inflexibilidade a mudanças nos códigos dos algoritmos, e portanto, a não garantia de que as versões implementadas sejam as melhores ou mais atuais.

Outros sistemas desenvolvidos em meio acadêmico estão sendo utilizados para o ensino de mineração de dados em universidades, são eles o ambiente WEKA e o DBMiner.

O ambiente de Waikato para análise de conhecimento (Witten, 1999), WEKA ('Waikato Environment for Knowledge Analysis'), é uma implementação de algoritmos de mineração de dados (associação, classificação, 'clustering'). Foi desenvolvida por pesquisadores do Departamento de Computação da Universidade de Waikato da Nova Zelândia (Witten, 2000). WEKA é implementado em Java, uma linguagem de programação que segue o paradigma orientado a objeto e traz benefícios como a reutilização de código, estruturação modular e independência de plataforma. WEKA agrega uma variedade de algoritmos de indução de conhecimento. Dentre os algoritmos que ele implementa estão o ID3, PRISM, OneR e Naïve Bayes. Uma limitação desta ferramenta refere-se ao fato de não poder tratar bases de dados com tamanhos muito superiores a memória física disponível. A mineração é feita através da leitura dos dados, obtidos de um arquivo previamente formatado: um arquivo no formato ARFF (Attribute-Relation File Format) é um arquivo texto (ASCII) que descreve uma lista de instâncias compartilhando um conjunto de atributos. A base de dados é transferida para memória na qual é processado o algoritmo de mineração. Possui duas seções distintas: cabeçalho, que descreve os atributos e seus possíveis valores, e a seção de dados, a relação de instâncias. À medida que são submetidos arquivos consideravelmente maiores que a

memória principal não se consegue a carga do arquivo .arff para a memória. Constitui assim uma arquitetura fracamente acoplada.

As ferramentas e sistemas emergentes de mineração de dados levaram naturalmente a demanda de linguagens de consulta de mineração de dados.

O DBMiner (Han et al., 1996) integra mineração de dados com sistemas de bancos de dados relacionais, oferecendo uma linguagem de mineração de dados, DMQL, parcialmente implementada. A linguagem adota uma sintaxe semelhante a SQL e provê primitivas para especificação de diferentes tarefas de mineração de dados. A interface para a descoberta de conhecimento baseada em SQL e em operações relacionais, facilita sua integração com SGBDs relacionais. O DBMiner é um sistema de mineração analítica on-line, desenvolvido para mineração interativa multinível de conhecimento em banco de dados relacionais e DW.

O sistema de mineração de dados Quest (Agrawal et al., 1996b) tem trabalhado com a implementação paralela de mineração de regras de associação (APRIORI – Agrawal et al., 1993) e com o algoritmo de classificação SPRINT (Shafer et al., 1996), onde vários processadores são capazes de trabalhar juntos para construir um modelo de classificação, particularmente uma árvore de decisão. Estes algoritmos utilizam tanto arquivos isolados quanto a família de produtos DB2. O sistema Quest é comercialmente disponível através do Intelligent Miner da IBM[3], um produto de mineração de dados da IBM que prover a implementação de um conjunto de algoritmos de mineração, algoritmos de redes neurais, métodos estatísticos e ferramentas de preparação e visualização de dados. As características distintivas do Intelligent Miner incluem a escalabilidade de seus algoritmos e a sua forte integração com o sistema de banco de dados relacional da IBM, DB2.

[3] http://www-3.ibm.com/software/data/iminer/

Capítulo

3

Mineração de Dados com Regras de Classificação

"Knowledge is that area of ignorance that
we arrange and classify."
Ambrose Bierce, 1842-1914?

O capítulo se inicia com uma discussão sucinta do processo de mineração de dados com técnicas de classificação. São introduzidas as técnicas de classificação mais comumente empregadas: regras de classificação e árvores de decisão. Na seqüência, regras de classificação são privilegiadas, pois repercutem diretamente no nosso trabalho. Mais precisamente, é apresentado, em detalhes, o algoritmo de indução de regras de classificação, PRISM, bem como são feitas considerações sobre sua viabilidade como técnica de mineração de dados.

3.1 O Processo de Mineração Dados com Técnicas de Classificação

Qualquer que seja a técnica de mineração de dados por classificação, ela se utiliza de dados sobre o passado (*conjunto de treinamento*) para classificar dados futuros (conjunto de execução de um modelo de classificação, ou simplesmente

conjunto de execução), ou seja, dados que não pertencem ao conjunto de treinamento — em geral coletados cronologicamente após os dados de treinamento. Independentemente também de uma técnica específica, uma classificação consiste em predizer o valor que um determinado atributo do conjunto de execução assumirá, dado um conjunto de valores dos demais atributos do conjunto de execução. Esse primeiro atributo, chamado *atributo de classificação*, indica a classe a que cada instância do conjunto de execução pertence. O algoritmo indutor de classificação baseado em um modelo de classificação é denominado de *classificador* a partir do conjunto de treinamento.

Os dois modelos mais conhecidos de regras de classificação são:

1. Regras de classificação *stricto sensu*, na forma SE <condição> ENTÃO <classificação>, cuja interpretação é "se os valores assumidos pelos atributos de um registro do conjunto de treinamento satisfazem as condições do antecedente da regra, então o registro recebe a classe indicada pelo valor do atributo de classificação";

2. Regras de classificação indiretas, sob a forma de árvores de decisão, ou seja, uma seqüência hierárquica de testes construídos ao longo de uma estrutura em árvore (condições), com os nós folhas da árvore representando as diferentes classes. Note então que uma árvore pode exprimir diferentes regras de classificação (cada regra é um caminho na árvore, da raiz até uma das folhas).

Uma definição formal de classificação é dada na seção 3.2.

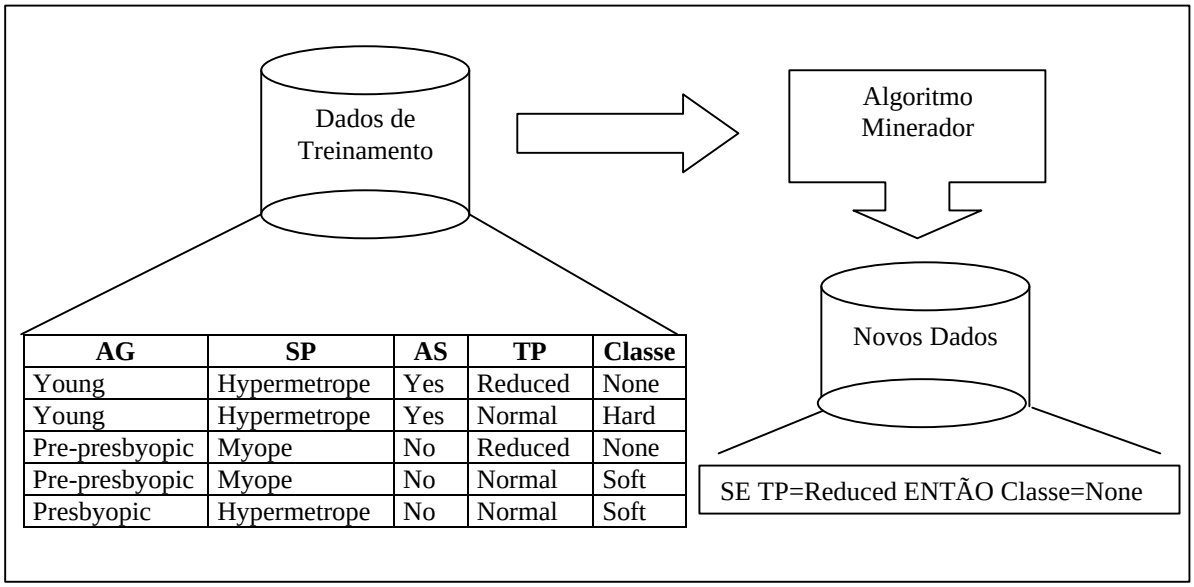
Independentemente do modelo de classificação, o desempenho de um algoritmo de classificação é fortemente dependente da forma como o conjunto de treinamento é preparado e organizado. Dados do mundo real estão longe da perfeição. Erros, valores ausentes e a possibilidade de valores não informados ('null') são fatos. Alguns erros são de medição (calibração incorreta de equipamentos), podendo ser corrigidos. Mas outros tipos de erro são inerentes aos

dados, e não podem ser contornados. Resumidamente, há um inevitável *ruído* (instâncias errôneas ou incompletas) no conjunto de treinamento; por mais que se possa reduzir o seu impacto, sempre sobrará algum ruído (Sampaio, 2001).

A *preparação dos dados para mineração* — essencialmente a redução da margem de ruído — constitui-se assim numa etapa importante do processo de mineração, e é inerentemente dependente do domínio da aplicação (Pyle, 1999).

Outra etapa importante e dependente da preparação de dados é a seleção dos atributos relevantes para mineração. Alguns atributos podem ser bem ruidosos, podendo confundir o algoritmo de mineração (Freitas, 1997). Há ainda a etapa de *transformação*, na qual dados podem ser generalizados para níveis conceituais mais altos. Por exemplo, os valores contínuos de um atributo podem ser generalizados para alguns poucos valores. Desde que a seleção de atributos e a possível discretização de alguns atributos permitem diminuir o volume do conjunto de treinamento, isto é bom para o desempenho do algoritmo de mineração.

O conjunto de treinamento após as suas preparação, seleção e transformação — ou conjunto *pré-processado* — constitui a entrada para o algoritmo indutor (figura 9). Uma vez que o atributo que indicada a qual classe pertence cada exemplo do conjunto de treinamento está explicito, este passo é conhecido como fase de aprendizagem do algoritmo de mineração.



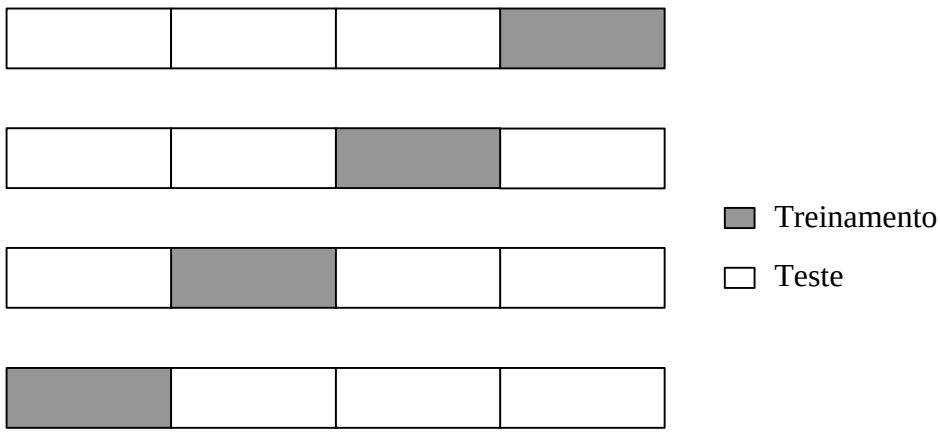
AG	SP	AS	TP	Classe
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	No	Reduced	None
Pre-presbyopic	Myope	No	Normal	Soft
Presbyopic	Hypermetrope	No	Normal	Soft

FIGURA 9. Fase de aprendizagem do algoritmo de mineração

Resta responder à pergunta: qual a confiabilidade — acurácia — das regras induzidas do conjunto de treinamento pelo classificador? A acurácia é medida pela porcentagem de instâncias do *conjunto de teste* (conjunto diferente dos conjuntos de treinamento e execução) que foram corretamente classificadas pelo classificador, de acordo com a classificação inferida. Além do termo “acurácia” , “desempenho” e “taxa de erro” são também empregados como sinônimos.

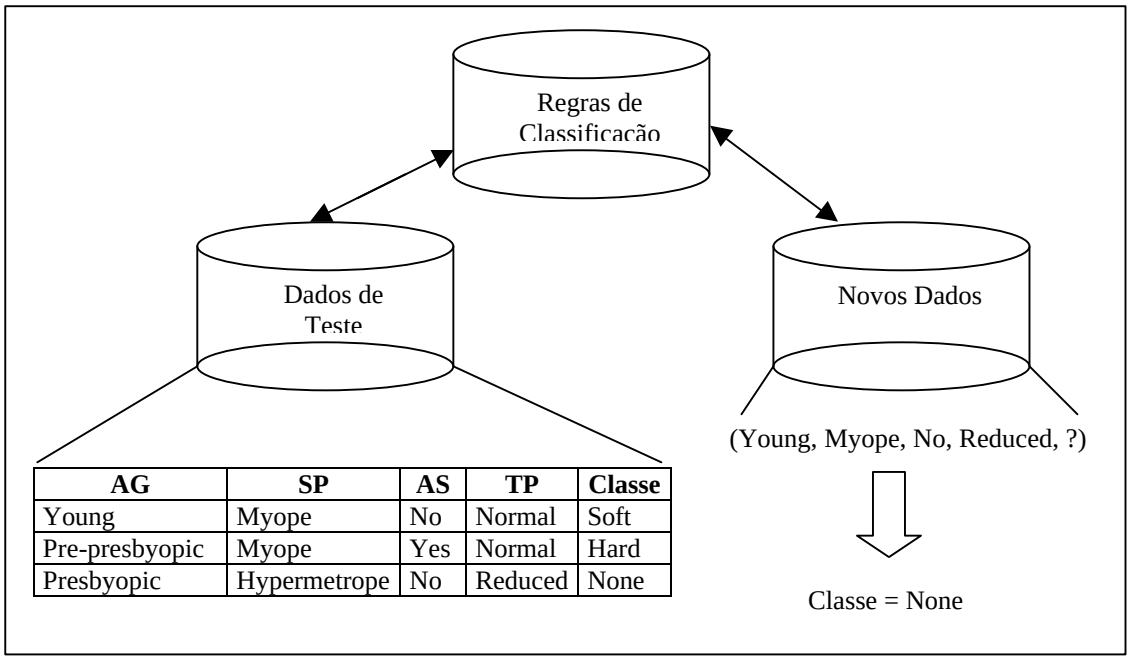
Na verdade, o valor da acurácia é somente estimado. As técnicas para estimar acurácia mais comumente utilizadas são ‘holdout’ e ‘cross-validation’. Com o tempo, surgiram diversas variações. A técnica ‘holdout’ (Witten et al., 2000) consiste na seleção aleatória de exemplos independentes do conjunto de exemplos de treinamento. Tipicamente, 2/3 (dois terços) dos dados são reservados para treinamento e o restante, reservado para teste.

Na técnica ‘cross-validation’ (Han et al., 2001), figura 10, os dados iniciais são fragmentados aleatoriamente em *k* subconjuntos mutuamente exclusivos, também chamados de ‘folds’, cada um de tamanho aproximadamente igual. Treinamentos e testes são realizados *kvezes*. Em cada iteração, um dos *k* subconjuntos é reservado como conjunto de teste e os *k-1* subconjuntos restantes são coletivamente usados para treinar o classificador. A estimativa da acurácia é o número total de classificações corretas das *k* iterações, dividido pelo tamanho do arquivo fragmentado.



FIRURA 10. A técnica ‘cross-validation’ para estimar acurácia

Se a acurácia da classificação induzida for considerada aceitável, ela deverá então ser empregada para classificar instâncias do conjunto de execução (figura 11).



AG	SP	AS	TP	Classe
Young	Myope	No	Normal	Soft
Pre-presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Reduced	None

FIGURA 11. Classificação do conjunto de execução

3.2 Definição Formal de Classificação

A definição formal do problema de classificação pode ser expressa da seguinte forma. O objetivo de um classificador é gerar *conhecimento* (*m*) a partir de um conjunto de dados passado como entrada. O conhecimento *m* pode ser então usado para classificar novos dados.

Mais precisamente, seja um conjunto de dados com n atributos independentes, $X_1 , \ldots , X_n$, tal que cada X_k assume um dos valores do domínio D_{xk}. Adicionalmente, seja um outro atributo X_c, chamado de *atributo de classificação*, cujo domínio é o conjunto de classes $D_c=\{C_1,.., C_m\}$, sendo *m* o número de classes. A tarefa de classificação é:

(i) Dado um conjunto de treinamento *E* consistindo de um conjunto *t* de tuplas: $<t_1 , . . ., t_n , c>$, onde $t_k \hat{I} D_{Xk}$ (K = 1,.., n) e $c \hat{I} D_c$;

(ii) Construir um mapeamento *m*: $(D_{X1} , . . ., D_{Xn}) \text{ Þ } D_C$.

O conhecimento *m* pode agora ser usado para classificar novas tuplas de um conjunto $t' = <t'_1 , t'_2 ,. . ., t'_n>$ tal que $t'_i \hat{I} D_{XK}$ *(K = 1,..,n).* A classe predita *c'* é: *c'= m(t').*

Considere agora um termo $X = v$ onde *X* é um atributo e *v*, um valor de seu domínio, $v \in D_X$. O termo *cobre* uma tupla *t* de um conjunto se o atributo *X* em *t* tem o valor *v*. Uma conjunção de termos *r* cobre uma tupla *t* quando todos os termos em *r* cobrem *t*. Uma regra *r* é uma declaração da forma: "SE *r* ENTÃO *q*", onde *q* é uma classe. A regra *r* cobre uma tupla *t* quando *r* cobre *t* (Giffrida et al., 2000).

3.3 Regras de Classificação *Stricto Sensu*

Uma regra de classificação *stricto sensu* é uma declaração da forma X $\Rightarrow$ Y, onde X e Y são o *antecedente* e o *conseqüente* da regra, respectivamente. O antecedente é composto por uma conjunção de condições envolvendo testes do tipo *atributo = valor* e o conseqüente é formado por um único termo, *atributo_de_classificação = valor.*

A maneira utilizada para produção de regras, diretamente a partir de um conjunto de treinamento, baseia-se na estratégia *separar-e-conquistar*, que teve suas origens na família de algoritmos *AQ* (Michalski, 1980; Michalski et al., 1986), com o nome 'covering strategy' (Michalski, 1969). O termo separar-e-conquistar foi usado por Pagallo (1990) na formalização da teoria que caracteriza esta estratégia de aprendizagem:

- induzir uma regra que cubra uma parte das instâncias de treinamento, remover as instâncias cobertas (a parte separar) e

- recursivamente aprender outras regras que cubram algumas das instâncias restantes (a parte conquistar) até que não reste nenhuma instância.

Isso garante que toda instância do conjunto de treinamento será coberta por pelo menos uma regra.

Esses algoritmos (também chamados ‘covering algorithms’) têm sido desenvolvidos para uma variedade de tarefas de indução (Fürnkranz, 1999). Por exemplo, alguns inferem listas de decisão, que são regras ordenadas, cuja interpretação deve ser feita na seqüência em que as regras foram induzidas, ou seja, uma regra é avaliada se ela cobre a instância corrente, não classificada. Em caso afirmativo, o processo de classificação pára e a classe da regra é atribuída à instância. Caso contrário, a próxima regra na lista é avaliada. A última regra na lista é chamada regra ‘default’.

Algoritmos separar-e-conquistar constróem regras pela adição sucessiva de condições ao lado direito da regra que está sendo construída. Baseiam-se na seleção de cada classe e pela procura das regras que cobrem as instâncias classificadas como tal. O enfoque é ‘bottom-up’ (das classes para os atributos).

O método para construção das regras toma uma classe por vez e procura uma maneira de cobrir todas as suas instâncias, ao mesmo tempo que exclui instâncias não contidas na classe. Começando com uma regra vazia, sucessivamente adiciona novas regras até que todos os exemplos sejam cobertos. Enquanto ela cobrir instâncias negativas (instâncias que apresentem o valor do atributo de classificação diferente do valor buscado na iteração atual), a regra corrente é especializada pela adição de condições ao seu antecedente. Condições possíveis são testadas sobre a presença de certos valores de vários atributos, com o objetivo de encontrar uma regra que não cubra exemplos negativos — um teste que obtenha o melhor valor para a porcentagem de exemplos positivos (instâncias que possuem o valor de classificação procurado) dentre todos os exemplos cobertos. Quando uma regra que cobre apenas exemplos positivos é encontrada, todos os exemplos cobertos serão removidos e outra regra será gerada a partir dos exemplos remanescentes. Isso é repetido até que não reste nenhum exemplo. Assim, é garantido que todas as regras induzidas cobrem

todos os exemplos positivos (completeza), mas nenhum dos exemplos negativos (consistência) (Fürnkranz, 1999). A figura 12 exemplifica graficamente esta característica dos algoritmos de cobertura.

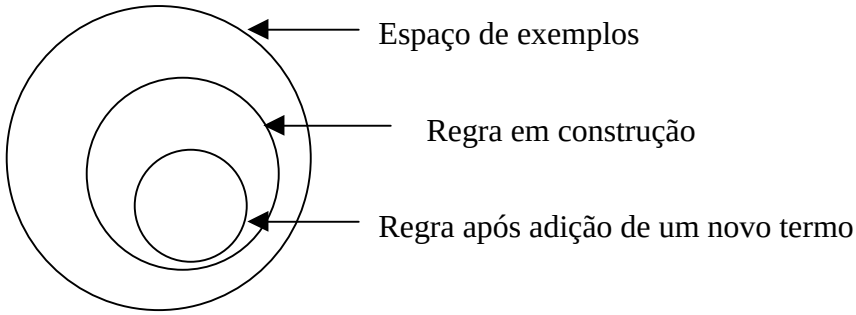


FIGURA 12. *Modus operandi* de um algoritmo de cobertura (Witten et al., 2000)

Em contraste, algoritmos de indução de árvores de decisão utilizam a abordagem dividir-e-conquistar — um problema complexo é decomposto em subproblemas mais simples e recursivamente a mesma estratégia é aplicada a cada subproblema — operam adicionando testes à árvore que está sendo construída, sempre se esforçando em encontrar um atributo no qual o conjunto possa ser dividido da melhor maneira (processo 'top-down', ou dos atributos para as classes).

Pelo menos um importante aspecto torna a indução via estratégia separar-e-conquistar mais atrativa que a abordagem dividir-e-conquistar: regras sob a forma de árvores de decisão são freqüentemente mais complexas e de difícil compreensão que regras de classificação *stricto sensu*. Um outro problema com árvores de decisão é o problema da "subárvore replicada" (Fürnkranz, 1999): freqüentemente acontece que subárvores idênticas têm que ser induzidas em vários lugares na árvore de decisão. Uma boa ilustração deste problema ocorre quando as regras têm a mesma estrutura, mas diferentes atributos como as regras:

SE a E b ENTÃO x
SE c E d ENTÃO x

Estas regras apresentam uma simetria que precisa ser quebrada para que se torne possível a escolha de um teste que constitua o nó raiz da árvore. Se, por

exemplo, *a* é escolhido como raiz, a segunda regra deve ser repetida duas vezes na árvore, como mostrado na figura 13 (Os valores *y* e *n* nos ramos significam o sucesso ou o fracasso no teste, respectivamente) (Witten, 2000).

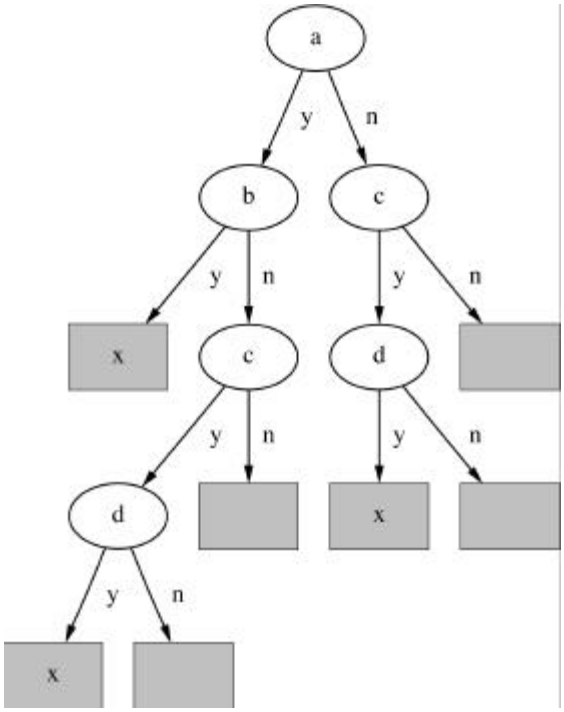


FIGURA 13. Árvore de decisão com subárvores replicadas (Witten, 2000)

3.4 O Algoritmo PRISM, de Regras de Classificação *Stricto Sensu*

PRISM foi desenvolvido por Jadzia Cendrowska (Cendrowska, 1987) baseado no algoritmo de indução de regras ID3. A motivação na sua criação foi de superar as limitações na representação de árvores de decisão. PRISM produz regras modulares (independentes) que são de mais fácil compreensão. A geração de regras de classificação é realizada, diretamente, a partir do conjunto de treinamento. Não há a necessidade da etapa intermediária de construção de uma árvore de decisão, que

apresenta o *problema sintático* relacionado ao fato de que todas as regras geradas a partir de uma árvore terão que conter o atributo raiz em seu antecedente.

Diferentemente do método dividir-e-conquistar, nos quais se baseiam os algoritmos de árvore de decisão, a abordagem adotada pelo PRISM é ‘covering’ e ‘bottom-up’.

O PRISM consiste em: dada uma massa de dados como entrada ao algoritmo este gerará somente regras ‘corretas’ (a regra gerada cobrirá apenas as instâncias que satisfazem as condições que aparecem no antecedente da regra). A cada atributo candidato é atribuída uma pontuação baseada na probabilidade:

$$Pontuação = \frac{p}{p+n} \qquad \textit{Equação 1}$$

Onde, p é o número de exemplos positivos em que este atributo tem o valor “presente” e n é o número de exemplos negativos para este valor do atributo. Esta probabilidade avalia as regras com relação a sua pureza, i.e. a porcentagem de exemplos positivos entre os exemplos cobertos pela regra. Esta medida atingirá seu valor ótimo quando nenhum exemplo negativo for coberto pela regra.

O algoritmo é iterativo, continuando enquanto a regra for “incorreta”, probabilidade menor que a unidade. O conjunto-treinamento é integralmente recuperado a cada nova iteração a fim de identificar os subconjuntos de uma classe específica. Acarreta desta forma, a desvantagem de aumentar um pouco o esforço computacional, mas apresenta a vantagem de gerar uma saída na forma de regras modulares. As regras para cada classe são criadas removendo-se objetos até que não haja mais nenhum registro da classe em questão. Inicialmente, as condições de uma regra são vazias. À medida que, novas proposições lhe são adicionadas, os exemplos vão sendo restringidos, até que só restam exemplos da classe desejada.

Como as classificações são consideradas separadamente sua ordem de apresentação é irrelevante. Se todas as instâncias são da mesma classificação então aquela classificação é retornada como regra e o algoritmo termina.

O algoritmo, tal como aparece em (Bramer, 2000), é mostrado na figura 14, assumindo que existam $n > 1$ classes.

Para cada classe *c* de 1 a *n* inclusive:

(1) Calcule a probabilidade de ocorrência da classe *c* para cada par atributo-valor.

(2) Selecione o par atributo-valor com a probabilidade máxima de ocorrência e crie um subconjunto a partir do conjunto de treinamento tomado como entrada compreendendo todas as instâncias que contenha o par selecionado (para todas as classes).

(3) Repita os passos 1 e 2 para este subconjunto até o momento em que ele apresente apenas instâncias da classe *c*. A regra induzida é então a conjunção de todos os pares atributo-valor selecionados na criação deste subconjunto homogêneo.

(4) Remova todas as instâncias, que satisfaçam a regra formada, do conjunto de treinamento.

Repita a seqüência de passos de 1 a 4 até que todas as instâncias da classe *c* tenham sido removidas.

FIGURA 14. Algoritmo PRISM

Este é o algoritmo básico da técnica separar-e-conquistar, uma regra que cobre instâncias na classe (e exclui as não contidas na classe), separando-as e continuando o processo sobre aquelas restantes. Tais algoritmos têm sido utilizados como base para muitos sistemas de geração de regras. O critério utilizado para escolha de qual teste adicionar a regra a cada estágio tem um efeito significante sobre as regras produzidas.

O real problema com estes esquemas de geração de regras é que eles tendem a especializar (overfitting) os dados de treinamento e não generalizar bem o bastante para o conjunto independente de teste. Alternativas para evitar-se essa característica indesejável é a mudança no critério de escolha dos testes a serem adicionados ao antecedente da regra em formação na tentativa de encontrar a melhor regra.

Em seguida, acompanhamos o algoritmo através do exemplo apresentado em (Cendrowska, 1987).

3.5 Algoritmo PRISM Através de Exemplo

Seja o conjunto de treinamento lens24 (Bramer, 2000), contendo um histórico de prescrições de lentes de contato, em uma clínica oftalmológica. São 24 instâncias, com 5 atributos — age (AG), spectacle prescription (SP), astigmatism (AS), tear production rate (TP) e o atributo de classificação, Classe. Os valores das classes são, 'hard', 'soft' e 'none', correspondendo às lentes de contato recomendadas (Tabela 1).

A primeira regra inferida (incompleta) é SE ? ENTÃO classe = hard, com a classe 'hard' sendo escolhida ao acaso. Para substituir "?" temos as 09 possibilidades mostradas na tabela 2 — passo (1) do algoritmo, figura 11.

TABELA 1. Conjunto de Treinamento lens24

AG	SP	AS	TP	Class
Young	Myope	Yes	Normal	Hard
Young	Myope	Yes	Reduced	None
Young	Myope	No	Reduced	None
Young	Myope	No	Normal	Soft
Young	Hypermetrope	No	Reduced	None
Young	Hypermetrope	No	Normal	Soft
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	No	Reduced	None
Pre-presbyopic	Myope	No	Normal	Soft
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	No	Reduced	None
Pre-presbyopic	Hypermetrope	No	Normal	Soft
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	No	Reduced	None
Presbyopic	Myope	No	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Reduced	None
Presbyopic	Hypermetrope	No	Normal	Soft
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

Da tabela 2, podemos observar que dois pares atributo-valor, AStigmatism = yes e Tear Production rate = normal, têm a mesma maior probabilidade. Suponha que o algoritmo escolha AS = yes (passo (2) do algoritmo). A regra induzida (ainda incompleta) é então SE AS = yes ENTÃO classe = hard. Esta regra apresenta baixa acurácia, isto é, apenas quatro instâncias são classificadas como 'hard', das doze cobertas pelo par astigmatism = yes. A idéia então é continuar refinando a regra, SE AS = yes AND ? ENTÃO classe = hard.

TABELA 2. Probabilidades de ocorrência dos pares

Atributo-valor	Freqüência (P) para classe = hard	Freqüência Total(T) (24 instâncias)	Probabilidade (P/T)
AG = young	2	8	0.25
AG = Pre-presbyopic	1	8	0.125
AG = presbyopic	1	8	0.125
SP = Myope	3	12	0.25
SP = Hypermetrope	1	12	0.083
AS = no	0	12	0
AS = yes	4	12	0.33
TP = reduced	0	12	0
TP = normal	4	12	0.33

O subconjunto obtido a partir do conjunto de treinamento para a regra em formação é mostrado na tabela 3.

TABELA 3. Subconjunto gerado

AG	SP	AS	TP	Classe
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

As probabilidades de ocorrência de cada par (desta vez não envolvendo o atributo AStigmatism, que já é uma condição da regra) são mostradas na tabela 4.

TABELA 4. Probabilidades de ocorrências dos pares

Atributo-valor	Freqüência (P) para classe = hard	Freqüência Total(T) (24 instâncias)	Probabilidade (P/T)
AG = young	2	4	0.5
AG = pre-presbyopic	1	4	0.25
AG = presbyopic	1	4	0.25
SP = myope	3	6	0.5
SP = hypermetrope	1	6	0.17
TP = reduced	0	6	0
TP = normal	4	6	0.67

A probabilidade máxima ocorre quando Tear Production = normal. A regra induzida é então: SE astigmatism = yes E TP = normal ENTÃO Classe = hard. O algoritmo continua buscando por uma regra exata (a relação P/T = 1 ou até que não haja mais atributos a serem adicionados à regra) não importando quão complexa ela possa tornar-se. O novo subconjunto obtido por esta regra é mostrado na tabela 5.

TABELA 5. Subconjunto gerado

AG	SP	AS	TP	Classe
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Normal	Hard
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Pesbyopic	Myope	Yes	Normal	Hard
Pesbyopic	Hypermetrope	Yes	Normal	None

As probabilidades de ocorrência de cada par atributo-valor são mostradas na tabela 6.

TABELA 6. Probabilidade de ocorrência dos pares

Atributo-valor	Freqüência (P) para classe = hard	Freqüência Total(T) (24 instâncias)	Probabilidade (P/T)
AG = Young	2	2	1.0
AG = Pre-presbyopic	1	2	0.5
AG = presbyopic	1	2	0.5
SP = Myope	3	3	1.0
SP = Hypermetrope	1	3	0.33

A probabilidade máxima ocorre quando AGe = Young e Spectacle Prescription = Myope. Neste caso, obtivemos duas frações apresentando o maior valor (AG = Young — 2/2 e SP = Myope — 3/3). Em casos como este suponhamos que a escolha baseie-se na fração que seja mais abrangente (3/3), maior P. Assim a regra final induzida é: SE AS = yes E TP = normal E SP = Myope ENTÃO Classe = hard. Removendo-se todas as instâncias cobertas por esta regra do conjunto de treinamento resultará em um novo conjunto de treinamento (com 21 instâncias). O mesmo processo de geração de regras será aplicado sobre este novo conjunto de instâncias até que não reste mais exemplos da classe 'hard'. O conjunto de treinamento será restaurado (24 instâncias originais) para indução das classes restantes.

3.6 Família de Algoritmos TDIDT

A abordagem dividir-e-conquistar para a construção de regras de decisão foi desenvolvida e refinada durante muitos anos por Ross Quinlan. Sua versão do algoritmo é chamada de ID3 ('Induction of Decision Tree'). Uma série de melhoramentos do ID3 culminou com o algoritmo chamado C4.5. Os melhoramentos incluem métodos para lidar com atributos numéricos, valores não informados, dados com ruído, poda de árvores ('prunning') e geração de regras SE—ENTÃO a partir das árvores.

A técnica básica usada pelos algoritmos da família de indução de regras de classificação (ID3 (Quinlan, 1983), C4.5 (Quinlan, 1993), C5.0 (Quinlan, 1996), CART (Breitman et al., 1984), CN2 (Clark et al. 1987), ASSISTANT (Cestnik et al., 1987) etc.) é conhecida como 'Top Down Induction of Decision Trees' (TDIDT). O princípio básico dos algoritmos da família TDIDT pode ser informalmente descrito como mostrado na figura 15.

SE todos os casos no conjunto de treinamento pertencem à mesma classe

ENTÃO retorne o valor da classe.

SENÃO

(a) selecione um atributo de particionamento.

(b) classifique as instâncias no conjunto de treinamento em subconjuntos não vazios, um para cada valor do atributo.

(c) retorne uma árvore com um ramo para cada subconjunto, cada ramo tendo uma subárvore descendente ou uma classe produzida pela aplicação recursiva do algoritmo para cada subconjunto por vez.

FIGURA 15. Algoritmo TDIDT

É importante que na seleção do atributo no passo (a) um mesmo atributo não seja selecionado mais de uma vez em cada ramo.

Todos os algoritmos de árvore de decisão da família TDIDT utilizam a estratégia dividir-e-conquistar. Após selecionar um teste apropriado para o nó raiz da árvore de decisão, o algoritmo divide o conjunto de treinamento em dois subconjuntos disjuntos, cada um representando um resultado possível do teste escolhido. O algoritmo é então recursivamente aplicado para cada um destes conjuntos independentemente.

O algoritmo ID3 é uma extensão do algoritmo TDIDT básico. Um ponto importante no projeto do ID3 foi a necessidade de construir árvores de decisão para grandes conjuntos de treinamento, onde o algoritmo básico de indução não pudesse ser usado diretamente. O método para este problema adotado pelo ID3 foi uma forma iterativa baseada no uso de subconjuntos relativamente pequenos do conjunto completo.

O ID3 ‘varre’ todos os dados (exemplos) de treinamento e começa a construir a árvore a partir de sua raiz. Inicialmente, é escolhido um atributo ‘a’, com a melhor função de avaliação (função esta que deve gerar um subconjunto de dados com a maioria dos exemplos pertencentes à mesma classe), para particionar esses dados. Para cada valor ‘i’ do atributo ‘a’, um ramo ‘r’ é criado junto com o correspondente subconjunto de dados que possuem o valor de ‘a’ igual a ‘i’. Assim,

para cada ramo ‘r’, é criado um nó ‘n’ na árvore que poderá ser uma classe (caso todos os exemplos analisados tenham ‘a’ igual a ‘i’ e representem a mesma classe) ou um outro atributo ‘a’ (se algum exemplo de ‘a’ igual a ‘i’ representar classes diferentes) para se fazer nova partição. O algoritmo é executado recursivamente em todos aqueles nós que não representarem uma classe (Mongiovi, 1995).

3.6.1 PRISM versus TDIDT

O princípio de obtenção de regras de classificação a partir da geração de árvores de decisão, quando comparada à obtenção direta a partir do conjunto de treinamento, foi fortemente criticado por Cendrowska[4], citada por Bramer (2000), seus comentários foram os seguintes.

> “A representação de regras através de árvores de decisão apresenta inúmeras desvantagens. Primeiramente, árvores de decisão são extremamente difíceis de se manipular — para extrair informação sobre alguma classificação simples é necessário examinar a árvore completa, um problema que é apenas parcialmente resolvido convertendo-se a árvore num conjunto de regras individuais [...]. O mais importante, há regras que não podem ser facilmente representadas por árvores.
>
> Considere, por exemplo, o seguinte conjunto de regras:
>
> Regra 1: IF a = 1 AND b = 1 THEN Classe = 1
> Regra 2: IF c = 1 AND d = 1 THEN Classe =1
>
> Suponha que as Regras 1 e 2 cobrem todas as instâncias da Classe 1 e todas as outras instâncias são da Classe 2. Estas duas regras não podem ser representadas em uma árvore de decisão simples como a que apresenta o nó raiz da árvore dividida a partir de um atributo simples; não há atributo que seja comum a ambas as regras. A representação da árvore

[4] . Cendrowska, J. Knowledge Acquisition for Expert Systems: Inducing Modular Rules from Examples. PhD Thesis, The Open University, 1990.

de decisão mais simples do conjunto de instâncias coberta por estas regras deveria necessariamente adicionar um termo extra para uma das regras, que por sua vez deve requerer pelo menos uma regra extra para cobrir instâncias excluídas pela adição do termo extra. A complexidade da árvore pode depender do número de valores possíveis dos atributos selecionados para particionamento. Por exemplo, tomemos os quatro atributos a, b, c e d cada um apresentando três possíveis valores 1, 2 e 3 e tomemos o atributo a ser selecionado para particionamento constituindo o nó raiz da árvore. A representação da árvore de decisão mais simples das Regras 1 e 2 é mostrada na figura 16.

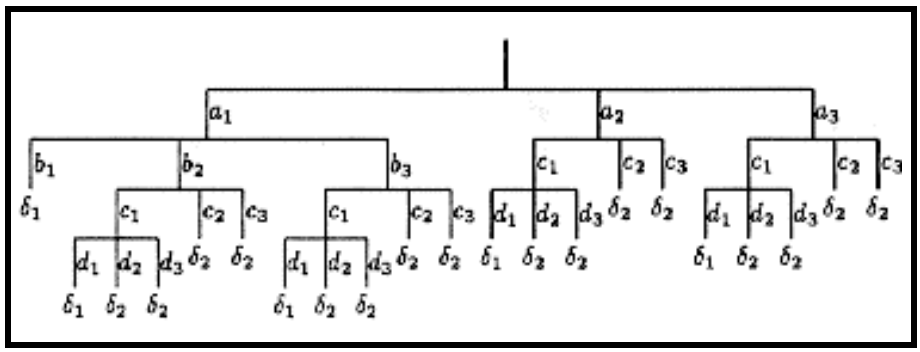


FIGURA 16. Árvore de decisão

Observe na figura as notações, tais como *a1*, são usadas para denotar que o valor do atributo *a* é 1 e δ*1* para denotar que a classe possui valor 1.

Os caminhos relativos a Classe 1 podem ser listados como:

IF a = 1 AND b = 1 THEN Classe = 1

IF a = 1 AND b = 2 AND c = 1 AND d = 1 THEN Classe = 1

IF a = 1 AND b = 3 AND c = 1 AND d = 1 THEN Classe = 1

IF a = 2 AND c = 1 AND d = 1 THEN Classe = 1

IF a = 3 AND c = 1 AND d = 1 THEN Classe = 1

Claramente, a conseqüência de forçar a representação de um conjunto de regras simples em uma árvore de decisão é que as regras

> individuais, quando extraídas da árvore, são freqüentemente muito específicas (i.e. elas referenciam atributos que são irrelevantes). Isto as torna altamente inconvenientes para uso em muitos domínios."

O fenômeno de árvores de decisão desnecessariamente grandes e confusas descrito por Cendrowska está longe de ser meramente uma possibilidade hipotética rara. Este fato ocorrerá sempre que houver duas regras com nenhum atributo em comum, uma situação que é provável de ocorrer freqüentemente na prática.

Sistemas que geram regras de classificação, e.g. C4.5, geralmente o fazem pelo método de poda postergada, i.e. primeiramente geram uma árvore de classificação, convertendo-a posteriormente num conjunto de regras equivalentes (uma regra para cada ramo da árvore) e então generaliza estas regras removendo termos redundantes. Embora este método tenha sucesso, parece ser um modo indireto desnecessário de obtenção de um conjunto de regras (Bramer, 2000).

Comparando ainda as regras obtidas por algoritmos TDIDT, Bramer analisou a complexidade de um conjunto de regras baseado em três parâmetros:

(i) o número de regras;

(ii) o número médio de termos por regra;

(iii) o número total de termos em todas as regras combinadas. Sendo este último parâmetro simplesmente o produto resultante dos dois primeiros.

Ele verificou que o PRISM em geral infere um número menor de regras que as obtidas com um algoritmo TDIDT, para o mesmo conjunto de treinamento. Além disso, na maioria dos casos experimentados, as regras inferidas com o PRISM têm menos termos que as inferidas com o algoritmo TDIDT. As tabelas 7 e 8 sintetizam os experimentos.

Como pode ser visto na tabela 7, no único caso em que PRISM inferiu mais regras que TDIDT, o tamanho médio de suas regras foi inferior ao do algoritmo TDIDT. Seja agora a linha "genetics", da tabela 8: a acurácia das regras com TDIDT é um pouco maior que a das regras induzidas por PRISM (839/950 > 825/950), porém PRISM errou bem menos (58/950 << 90/950); em proveito de inferir regras corretas, PRISM prefere não classificar casos duvidosos (coluna Testes não Classificados) , provocando uma redução no número de instâncias classificadas

incorretamente, o que não parece ser o caso de TDIDT. De uma maneira geral, as mesmas conclusões valem para quase todas as outras linhas da tabela 8.

TABELA 7. Comparação TDIDT x PRISM

	No. de Instâncias	No. de Regras		No. Médio de Termos		No. Total de Termos	
		TDIDT	PRISM	TDIDT	PRISM	TDIDT	PRISM
agaricus_lepiota	5000	17	22	2.18	1.23	37	27
Chess	647	20	15	5.25	3.20	105	48
contact_lenses	108	16	15	3.88	3.27	62	49
Genetics	3190	389	244	5.71	3.95	2221	963
Monk1	124	46	25	4.04	3.00	186	75
Monk2	169	87	73	4.74	4.00	412	292
Monk3	122	28	26	3.36	2.81	94	73
Soybean	683	109	107	5.45	3.57	594	382

FONTE: Bramer, 2000

TABELA 8. Comparação TDIDT x PRISM (II)

	No. de Instâncias	Testes Corretos		Testes Incorretos		Testes Não Classificados	
		TDIDT	PRISM	TDIDT	PRISM	TDIDT	TDIDT
agaricus_lepiota	1478	1474	1474	1	1	3	3
chess	182	181	178	1	2	0	2
contact_lenses	33	30	28	3	3	0	2
genetics	950	839	825	90	58	21	67
monk1	36	26	25	9	8	1	3
monk2	52	22	27	28	24	2	1
monk3	36	33	28	3	5	0	3
Soybean	204	174	169	22	20	8	15

FONTE: Bramer, 2000

O resultado destes experimentos sugerem que o algoritmo PRISM produz regras de classificação que são pelo menos tão boas quanto àquelas obtidas a partir

dos algoritmos TDIDT largamente utilizados. O esforço computacional envolvido na geração de regras usando-se o PRISM é maior que o envolvido pelos algoritmos TDIDT. Entretanto, Prism parece ter potencial considerável para melhorar sua eficiência integrando-se a um SGBD que possibilita mecanismos de otimização e paralelismo.

Capítulo

4

Integração do PRISM com SGBDs

" I never waste memory on things
that can easily be stored and retrieved
from elsewhere."
Albert Einstein, 1879-1955

Este capítulo descreve a implementação do PRISM, estreitamente integrada com SGBDs. Inicialmente, o PRISM é apresentado conceitualmente, utilizando os formalismos UML e ODL/ODMG — seção 4.2. Sendo conceitual, a especificação é genérica, ou independente de um particular SGBD. Em seguida, na seção 4.3, a especificação conceitual UML – ODL/ODMG é mapeada para duas especificações lógicas — objeto-relacional e relacional —, segundo a sintaxe PL/SQL do SGBDOR Oracle9i.

Precedendo as seções 4.2 e 4.3, o capítulo começa efetivamente com uma discussão geral do problema da integração de algoritmos de regras de classificação com SGBDs — seção 4.1.

4.1 Integração de Algoritmos de Regras de Classificação com SGBDs

Uma das técnicas chave da mineração de dados é a indução de regras a partir de exemplos, particularmente a indução de regras de classificação. A maioria

dos trabalhos nesse campo tem se concentrado na geração de tais regras na forma intermediária de árvores de decisão. Um método alternativo é gerar regras de classificação modulares (independentes) diretamente a partir de exemplos.

Como foi discutido no capítulo anterior, PRISM apresenta a grande vantagem de inferir conhecimento comparativamente menos complexo — menor número de regras, e regras com menor número de termos —, e portanto de mais fácil assimilação.

Na prática, os dados normalmente residem em um SGBD, no entanto eles não estão diretamente acessíveis a um algoritmo de mineração. Exportar grande parte dos dados de um SGBD torna-se uma tarefa que consome bastante tempo, requer grande quantidade de espaço extra em disco e inclui freqüentemente a complicada tarefa de transformar dados em um formato apropriado aceitável pelo algoritmo. Como conseqüência, ao invés de trazer os dados para o algoritmo, leva-se o algoritmo aos dados e dessa maneira integrado ao SGBD.

Muita pesquisa tem sido feita em integração de algoritmos de mineração de dados com SGBDs. O foco principal dessas pesquisas tem sido em algoritmos baseados em regras de associação e em SGBDRs. Fica evidente então a necessidade da integração de outras famílias de algoritmos de mineração (árvores de decisão, regras de classificação, Tc) não somente com SGBDRs, mas também com os modernos SGBDOOs e SGBDORs.

4.2 PRISM como Objeto

Nesta seção, tratamos da modelagem conceitual orientada a objeto do PRISM.

Uma característica fundamental da abordagem de banco de dados é tratar de diversos níveis de abstração dos dados, possibilitando a encapsulação de detalhes que são desnecessários aos diferentes tipos de usuários. Para cada nível de abstração, existe um modelo de representação de dados nesse nível. Os modelos conceituais são empregados para representar os dados para os usuários que não estão interessados em

detalhes de implementação. No outro extremo, temos os modelos físicos, descrevendo detalhes de implementação como as arquiteturas das memórias volátil e não-volátil, a indexação para efeitos de desempenho, etc. Entre os modelos conceitual e físico situa-se o modelo lógico, ou modelo implementável (e. g., o modelo lógico relacional implementado pelos SGBDs relacionais), segundo um modelo físico.

Para a definição do algoritmo PRISM como objeto, os atributos e os métodos foram encapsulados numa classe abstrata denominada Prism. Duas outras classes concretas, PrismR e PrismOR, estendem a classe Prism, para especificar as implementações relacional e objeto-relacional do Prism. O diagrama de classes UML (Larman, 1998) é mostrado na figura 17, enquanto que as respectivas descrições analíticas ODL/ODMG das classes se encontram na figura 18.

Foram definidos sete atributos para a classe abstrata Prism:

- ConjTreinamento: armazena o nome do conjunto de treinamento no qual as instâncias estão armazenadas.

- Atributos_ConjTreinamento: contém os nomes dos atributos que irão constituir as condições das regras a serem induzidas.

- Dominios_Atrib_ConjTrein: relaciona todos os valores possíveis que um atributo poderá assumir.

- Cardinalidade_Dominios: quantifica quantos são os valores possíveis para cada atributo, possibilitando o acesso no arranjo de Dominios_Atrib_ConjTrein.

- Classes: relaciona quais são os valores possíveis para o atributo de classificação.

- Regras: armazena as regras induzidas a cada iteração do algoritmo.

- Acuracia_Atrib-Valor: contem as estatísticas de ocorrência de cada par atributo-valor a cada iteração do algoritmo.

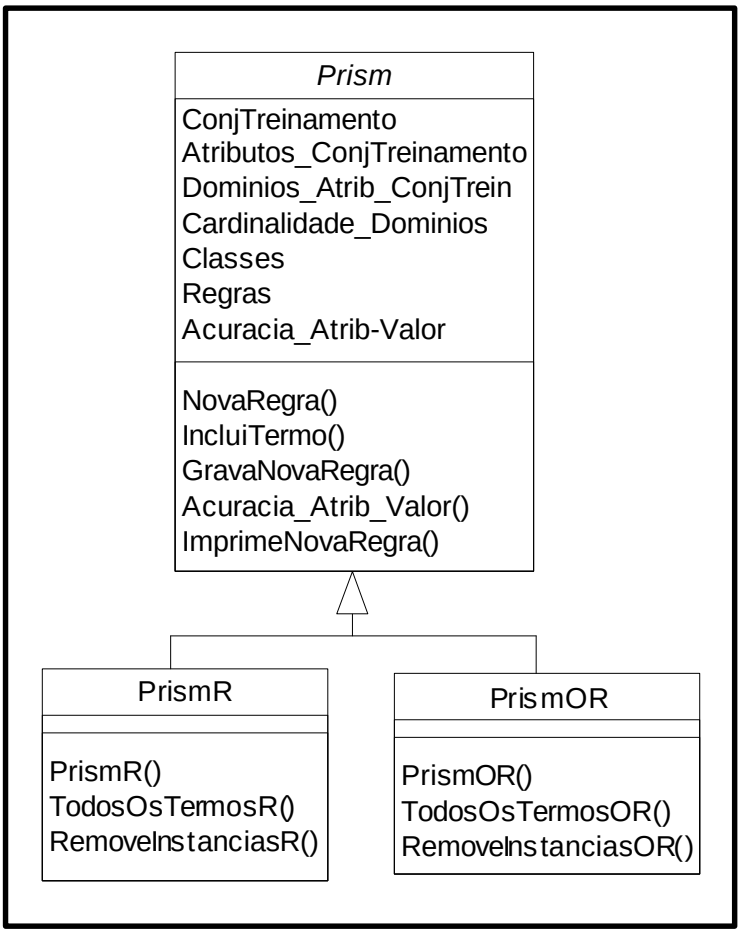


FIGURA 17. Diagrama UML de Classes

Os métodos definidos nas subclasses PrismR e PrismOR são polimorfos, isto é, têm a mesma assinatura mas as implementações diferentes. A figura 18 exibe um comentário sucinto para cada atributo/método das classes Prism, PrismR e PrismOR, respectivamente.

```
Class Prism {
   attribute Atributos_ConjTreinamento; /* nome dos atributos */
   attribute Dominios_Atrib_ConjTrein;   /*domínios dos atributos, menos o de classificação*/
   attribute Cardinalidade_Dominios;     /* cardinalidade dos domínios dos atributos */
   attribute Classes;                    /* domínio do atributo de classificação */
   attribute Regras;                     /* coleção de regras formadas em cada iteração */
   attribute Acuracia_Atrib_Valor;       /* relação P/T, ver tabela 2 da seção 3.5 */

   NovaRegra(inout array<char>, in char);
      /* adiciona uma nova regra a Regras (inout) com lado esquerdo vazio para uma classe (in)*/
   IncluiTermo(inout array<char>, in array<char>);
      /* acessa Acuracia_Atrib_Valor via função do mesmo nome (in) e adiciona a Regras
        (inout) o par atributo-valor com melhor acurácia para a regra em formação */
   GravaNovaRegra(in array<char>);   /* grava uma nova regra em Regras */
   Integer Acuracia_Atrib_Valor(in array<char>);
      /* função que retorna a posição do par atributo-valor com a melhor acurácia para
        a regra sendo formada */
    String ImprimeRegras();   /* exibe o conjunto de regras induzido por PRISM */
}

Class PrismR extends Prism
    (extent PrismsR) {
   attribute ConjTreinamento; /* nome do conjunto de treinamento*/
   Prism();                   /* implementação relacional de PRISM */
   TodosOsTermos();           /* gera valores P/T relativos a pares atributo-valor,
                                 armazenando-os em Acuracia_Atrib_Valor */
   RemoveInstancias();        /* removem as instâncias que casam com uma regra gerada*/
}

Class PrismOR extends Prism
    (extent PrismsOR) {
  attribute ConjTreinamento; /* nome (apontador) do conjunto de treinamento*/
  Prism();                   /* implementação objeto-relacional de PRISM */
  TodosOsTermos();           /* gera valores P/T relativos a pares atributo-valor,
                                armazenando-os em Acuracia_Atrib_Valor */
   RemoveInstancias();       /* removem as instâncias que casam com uma regra gerada */
}
```

FIGURA 18. Descrição ODMG das Classes da Figura 17

O estado de um objeto PrismR é então grosso modo definido por um conjunto de treinamento e por um conjunto de regras de classificação, inferido do conjunto de treinamento pela versão relacional do método Prism.

O estado de um objeto PrismOR é então grosso modo definido por um conjunto de treinamento e por um conjunto de regras de classificação, inferido do conjunto de treinamento pela versão objeto-relacional do método Prism.

```
/* Executar a versão OR de PRISM para o conjunto de treinamento "Lens24". */
SELECT obj.Prism()
FROM   PrismsOR obj
WHERE obj.ConjTreinamento = 'Lens24';

/* Exibir as regras inferidas pela versão OR de PRISM para o conjunto de treinamento "Lens24".*/
SELECT  obj.ImprimeRegras()
FROM    PrismsOR obj
WHERE  obj.ConjTreinamento = 'Lens24';
```

FIGURA 19. Consultas OQL

Os objetos das classes PrismR e PrismOR podem ser consultados de diferentes maneiras, empregando para isso a linguagem de consulta a objeto OQL/ODMG. A figura 19 ilustra duas consultas OQL elaboradas sobre o esquema conceitual definido.

4.3 PRISM implementado no SGBDOR Oracle9i

A implementação foi feita utilizando PL/SQL como linguagem de programação. PL/SQL é a extensão ao SQL da Oracle adicionando construções procedurais, variáveis e outras funcionalidades ao SQL. O PL/SQL é projetado para ter uma íntima integração com o sistema de banco de dados Oracle e com a SQL. Consequentemente, ele suporta todos os tipos de dados internos nativos ao SGBD Oracle. Apresenta, dessa forma, as principais vantagens de suporte ao SQL, suporte a programação orientada a objeto, melhor performance, alta integração com o Oracle e segurança. Em PL/SQL a programação orientada a objeto é baseada em “object types”.

Tipos objeto (‘object types’), diferentemente dos tipos SQL nativos tais como, NUMBER, VARCHAR ou DATE, são tipos definidos pelo usuário. Eles especificam os dados persistentes (chamados atributos do ‘object type’) e os comportamentos relacionados (chamados métodos do ‘object type’). Facilitam a modelagem de objetos complexos do mundo real, encapsulando os atributos de acordo com seu comportamento semântico.

Um ‘object type’ apresenta as seguintes características: (i) um ou mais atributos que definem a estrutura do objeto, que podem ser tipos (ou coleções desses tipos) de dados nativos, outros ‘object types’, ‘Large Objects’ (LOBs) ou referência a outros tipos. (ii) Podem ter zero ou mais métodos para acessar e manipular seus atributos, esses métodos podem ser executados dentro do ambiente de execução do SGBD ou no lado cliente da aplicação.

Uma vez que o ‘object type’ foi criado, ele poderá ser usado na definição de uma tabela ou de uma coluna e desse modo permitir o armazenamento dos objetos instanciados.

O modelo objeto-relacional, permite criar tabelas com colunas cujo tipo de dado de domínio é outra tabela. Isto é, as tabelas podem ser “aninhadas” dentro de outras tabelas como valores na coluna. Este conceito é definido pelo SGBD Oracle com o nome de ‘nested tables’. Uma tabela aninhada é um conjunto de elementos do mesmo tipo que tem as características de uma tabela relacional. Ela tem uma única coluna cujo domínio é um tipo estruturado definido pelo usuário. O fato a ser observado é que, no SGBD Oracle, uma tabela pode conter mais que uma tabela aninhada, mas nenhuma tabela aninhada pode conter outra tabela aninhada. Dessa forma, não se pode ter coleções de coleções, Oracle permite apenas um nível de aninhamento direto de coleções. Entretanto, um atributo de uma coleção pode ser uma referência para um objeto que tenha uma coleção como atributo. Assim, pode-se ter múltiplos níveis de coleções, indiretamente, através do uso de REF (referências).

Para as implementações relacional e objeto-relacional das classes ODL/ODMG — PrismR e PrismOR, respectivamente — no Oracle9i, foram criados dois tipos de dados Oracle dinamicamente definidos, TabelaR e TabelaOR, figura 20.

Da mesma forma que as linguagens de programação orientadas a objeto, o modelo objeto-relacional também suporta o conceito de herança de tipos de dados estruturados definidos pelo usuário, permitindo criar um subtipo de tipos existentes. Desta forma, um subtipo herda especificações de atributos e métodos de todos os supertipos associados. A herança no Oracle9i, especificada pela cláusula UNDER, agrupa os tipos em uma hierarquia. As seções 4.3.1 e 4.3.2 descrevem os dois modelos.

As definições detalhadas dos tipos e dos corpos dos métodos encontram-se no apêndice A.

```
CREATE OR REPLACE TYPE Prism AS OBJECT (
  AtributosConjTreinamento   atributosVarray,
  dominiosAtribConjTrein     valoresVarray,
  cardinalidadeDominios      deslocamento,
  classes                    classeVarray,
  regras                     regrasVarray,
  acuraciaAtribValor         condVarray,
  MEMBER PROCEDURE NovaRegra(classeCorrente IN VARCHAR2),
  MEMBER PROCEDURE IncluiTermo (indice IN INTEGER,
                          regras IN OUT regrasVarray,
                          acuraciaAtribValor IN condVarray),
  MEMBER PROCEDURE GravaNovaRegra (regras IN regrasVarray),
  MEMBER FUNCTION  CalculaAcuraciaAtribValor (
                          acuraciaAtribValor IN condVarray)
         RETURN INTEGER,
  MEMBER FUNCTION  ImprimeNovaRegra(regras IN regrasVarray)
         RETURN VARCHAR2
) NOT INSTANTIABLE NOT FINAL;
/

CREATE OR REPLACE TYPE PrismR UNDER Prism (
  ConjTreinamento  VARCHAR2(10),
  MEMBER PROCEDURE Prisma,
  MEMBER PROCEDURE TodosOsTermos (
                        regras IN OUT regrasVarray,
                        acuraciaAtribValor IN OUT condVarray),
  MEMBER PROCEDURE RemoveInstancias (
                   regras IN regrasVarray)
);
/

CREATE OR REPLACE TYPE PrismOR UNDER Prism (
  ConjTreinamento   TabelaORref,
  MEMBER PROCEDURE Prisma,
  MEMBER PROCEDURE TodosOsTermos(regras IN OUT regrasVarray,
                        acuraciaAtribValor IN OUT condVarray),
  MEMBER PROCEDURE RemoveInstancias (regras IN regrasVarray)
);
/

CREATE TABLE PrismsR OF PrismR;

CREATE TABLE PrismsOR OF PrismOR
    NESTED TABLE ConjTreinamento STORE AS conj_ntab;
```

FIGURA 20. Mapeamento das Classes/ODL para 'Object Types' do Oracle 9i (PL/SQL)

4.3.1 Versão PRISMR do Prism

Os dados foram organizados num esquema no formato de uma tabela puramente relacional. Para isso, o tipo TabelaR foi definido com todos os atributos (colunas) sendo atômicos. A tabela 9 ilustra esse tipo, instanciado para o conjunto de treinamento 'Lens24'.

A definição do tipo relacional é a seguinte:

```
CREATE OR REPLACE TYPE tabelaR AS OBJECT(
  age         VARCHAR2(15),
  spectacle   VARCHAR2(15),
  astigmatism VARCHAR2(15),
  tear        VARCHAR2(15),
  classe      VARCHAR2(8)
)
/
```

Todos os atributos de classificação do conjunto de treinamento são definidos como atributos do tipo tabelaR. Este tipo será usado para construção de uma 'object table', denominada 'E', que armazenará todas as instâncias do conjunto de treinamento. A definição desta tabela é a seguinte:

```
CREATE TABLE E OF TabelaR;
```

Graficamente, podemos observar esta definição na tabela 9.

TABELA 9. TabelaR

Tabela E

AG	SP	AS	TP	Classe
Young	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Normal	Soft
...	...	...	...	...
Presbyopic	Myope	No	Reduced	None
Prepresbyopic	Myope	No	Normal	Soft
Young	Hypermetrope	No	Reduced	None
Young	Hypermetrope	Yes	Normal	Hard

Para manipulação das instâncias foram definidos no tipo PrismR os métodos:

- Prisma: procedimento que encapsula o algoritmo PRISM e realiza chamadas aos outros métodos definidos no tipo Prism e PrismR;

- TodosOsTermos: procedimento que percorre todo o conjunto de treinamento para levantamento das probabilidades de ocorrências dos pares atributo-valor;

- RemoveInstancias: procedimento que deleta as instâncias que foram cobertas pela regra induzida na última iteração.

São herdados, ainda, do tipo Prism os seguintes métodos:

- NovaRegra: procedimento que cria uma nova regra com o lado esquerdo vazio e que prediz a classe que está sendo induzida;

- IncluiTermo: procedimento que inclui o par atributo-valor encontrado com maior probabilidade de ocorrência no conjunto de treinamento;

- GravaNovaRegra: procedimento que grava a última regra induzida na tabela de regras;

- CalculaAcuraciaAtribValor: função que analisa o ‘array’ de condições (acuraciaAtribValor) e realiza a divisão entre os valores dos atributos P (exemplos positivos) e T (todos os exemplos) para obtenção da probabilidade de ocorrência, retornando o maior valor encontrado;

- ImprimeNovaRegra: função que retorna uma ‘string’ exibindo a regra induzida no formato ‘SE-ENTÃO’;

O tipo PrismR possui o atributo ConjTreinamento, um VARCHAR que contém o nome da tabela que armazena o conjunto de treinamento. Pelo mecanismo de herança PrismR possui, também, os seguintes atributos da superclasse Prism:

- AtributosConjTreinamento: atributo que lista os nomes de todos os atributos do conjunto de treinamento;

- DominiosAtribConjTrein: atributo que lista todos os valores que cada atributo poderá assumir;

- CardinalidadeDominios: atributo utilizado como indexação entre os atributos AtributosConjTreinamento e DominiosAtribConjTrein para que se possa correlacioná-los;

- Classes: atributo que lista todos os valores possíveis que o atributo de classificação poderá assumir;

- Regras: atributo que armazenará uma regra induzida a cada iteração do algoritmo;

- AcuraciaAtribValor: atributo que relaciona o par atributo-valor com os valores P e T.

Além da tabela ‘E’, foi utilizada uma outra tabela para armazenamento das instâncias positivas que são eliminadas no decorrer do processamento. Esta tabela é definida da mesma forma que a tabela ‘E’ sendo denominada tabela ‘P’.

Para armazenamento dos objetos do tipo PrismR uma tabela, denominada PrismsR, foi definida.

4.3.2 Versão PRISMOR do Prism

O tipo TabelaOR foi criado tendo um atributo não atômico — coleção ‘nested table’ do Oracle9i —, em que cada elemento da coleção é uma instância do conjunto de treinamento da mesma classe, ou do mesmo valor do atributo correspondente “Classe”. Na tabela 10, os atributos dos elementos da coleção (AG, SP, AS e TP) estão implícitos. A tabela, assim definida, possui o número de linhas (objetos) igual ao número das possíveis classes. A principal vantagem dessa definição de dados é a forma de recuperação dos dados, visto que o Prism opera sobre todas as instâncias de uma classe por vez. Assim, elas já se encontram agrupadas numa única estrutura, tabela aninhada, facilitando a recuperação.

A definição dos tipos utilizados na construção do tipo TabelaOR é dada a seguir e encontra-se detalhada no Apêndice A. Para definição da coleção aninhada foram definidos os tipos ‘atributos’ e ‘coleção’. O tipo *atributo* terá o nome dos atributos do conjunto de treinamento, a exceção do atributo de classificação. O tipo *coleção* irá agrupar estes atributos numa ‘nested table’ que consistirá em um atributo do tipo TabelaOR. Uma tabela, denominada ‘E’, armazenará as instâncias criadas do objeto TabelaOR.

```
CREATE OR REPLACE TYPE atributos AS OBJECT(
  age             VARCHAR2(15),
  spectacle       VARCHAR2(15),
  astigmatism     VARCHAR2(5),
  tear            VARCHAR2(8)
);
/

CREATE OR REPLACE TYPE colecao AS
       TABLE OF atributos;
/

CREATE OR REPLACE TYPE tabelaOR AS OBJECT(
   aninhada  colecao,
   Classe    VARCHAR2(5)
);
/

CREATE TABLE E OF tabelaOR
   NESTED TABLE aninhada STORE AS colecao_ntab;
```

Graficamente, podemos observar esta definição na tabela 10.

TABELA 10. TabelaOR

TabelaOR

Coleção "Nested Table"				Class
Young	Myope	Yes	Normal	Hard
. . .	. . .	. . .	. . .	
Young	Hypermetrope	Yes	Normal	
Presbyopic	Hypermetrope	No	Normal	Soft
. . .	. . .	. . .	. . .	
Prepresbyopic	Myope	No	Normal	
Presbyopic	Myope	No	Reduced	None
. . .	. . .	. . .	. . .	
Young	Hypermetrope	No	Reduced	

O tipo PrismOR possui o atributo ConjTreinamento, uma referência para a coleção de objetos do tipo tabelaOR. Para o exemplo das lentes de contado, ConjTreinamento armazena apontadores para os três objetos possíveis.

Para armazenamento dos objetos do tipo PrismOR uma tabela, denominada PrismsOR, foi definida. Esta tabela possui uma 'nested table' para armazenamento da coleção de referências aos objetos tabelaOR.

Como os conjuntos de treinamento variam, cada uma das tabelas pode ter um número variável de linhas e um número diferente de colunas, os tipos TabelaR e TabelaOR devem ser dinamicamente definidos. O procedimento 'defineTipos' foi codificado para construção dinâmica dos tipos. Este procedimento, mostrado na figura 21, recebe como parâmetro de entrada a coleção de atributos responsáveis pela classificação.

A figura 20 descreve os 'Object Types' e as 'Object Tables' do Oracle9i, em PL/SQL, correspondendo ao esquema ODL/ODMG da figura 17. O código dos corpos dos métodos e a definição dos tipos encontram-se detalhados no Apêndice A.

```
CREATE OR REPLACE PROCEDURE defineTipos(
        Atributos IN atributosVarray) IS
 statement       VARCHAR2(2000);
BEGIN
     -- Define tipo tabelaR
     statement :='CREATE OR REPLACE TYPE tabelaR AS OBJECT(';

     FOR i IN Atributos.FIRST .. Atributos.LAST-1 LOOP
       statement := statement || '  ' ||
                    Atributos(i) || ' ' || 'VARCHAR2(15), ';
     END LOOP;

     statement := statement || ' Classe VARCHAR2(15)) ';
     EXECUTE IMMEDIATE  statement;

     -- Define tipos para construcao do tipo tabelaOR
     statement := 'CREATE OR REPLACE TYPE atributos AS OBJECT( ';
     FOR i IN Atributos.FIRST .. Atributos.LAST-2 LOOP
       statement := statement || '  ' ||
                    Atributos(i) || ' ' || 'VARCHAR2(15), ';
     END LOOP;

     statement := statement || Atributos(Atributos.LAST-1)
                  || ' VARCHAR2(15)) ';
     EXECUTE IMMEDIATE  statement;

     Statement := 'CREATE OR REPLACE TYPE colecao AS
                   TABLE OF atributos';
     EXECUTE IMMEDIATE  statement;

     statement := 'CREATE OR REPLACE TYPE tabelaOR AS
                   OBJECT(aninhada colecao, Classe VARCHAR2(15)) ';
     EXECUTE IMMEDIATE  statement;

     -- Colecao de referencias para o objeto tabelaOR
     statement := 'CREATE OR REPLACE TYPE TabelaORref AS
                   TABLE OF REF tabelaOR';
     EXECUTE IMMEDIATE  statement;
END;
```

FIGURA 21. Definição dinâmica de tipos

4.4 Sintonização do PRISM com o SGBD Oracle9i

A função do Oracle, bem como qualquer SGBD é armazenar dados e ser capaz de recuperá-los quando se faz necessário. Existem dois métodos básicos para localizar dados, também chamados caminhos de acesso: (i) ‘Full table scan’: realiza a leitura seqüencial, bloco por bloco, de toda a tabela, verificando registro por registro os critérios de seleção; (ii) ‘Index search’: procura de um índice sobre a coluna definida no critério de seleção, identificando a localização na tabela todos os registros qualificados.

O Oracle usa seu próprio mecanismo de otimização, chamado otimizador, para executar comandos SQL e dinamicamente determinar que caminho de acesso seguir, dependendo das informações disponíveis. É possível ajudar o otimizador a procurar o melhor caminho de acesso através da criação de índices.

Para acelerar os tempos de resposta das consultas às 'Object Tables' PrismsR e PrismsOR, um processo sistemático de sintonização aplicação-Oracle9i foi seguido.

A indexação das tabelas foi um poderoso mecanismo de aceleração dos tempos de resposta das consultas. Para os atributos indexados dos tipos, foram criados histogramas exatos da distribuição dos seus valores, através do comando 'Analyzer' do Oracle9i. Dessa forma, o banco de dados coleciona e mantém estatísticas sobre os dados nas tabelas, tais como o número de valores distintos, valor mínimo e máximo, histogramas de distribuição entre outros, para auxiliar o otimizador em suas estimativas.

```
alter session set optimizer_mode = choose;
analyze table E compute statistics;
analyze index classe_idx compute statistics;
```

O otimizador também avalia o custo de vários planos de acesso enquanto escolhe o plano ótimo. A seleção do método de acesso toma como base o conhecimento sobre os dados, sua distribuição e o modo como o usuário irá acessá-los. Por exemplo, o otimizador pode computar o custo de usar um índice bem como a varredura completa de uma tabela de forma a ser capaz de decidir entre ambas as opções. Oracle possui dois métodos de otimização: otimização baseada em Regras e otimização baseada em Custo.

A otimização baseada em regras trabalha de acordo com um conjunto pré-definido de regras que ditam qual é o melhor modo de acesso. A otimização baseada em custo escolhe um plano de execução que é, em geral, tão bom ou melhor que o plano escolhido pelo baseado em regras. Para isso, dicas ('hints') são fornecidas no código PL/SQL para guiar o otimizador na realização da consulta (SELECT /*+CHOOSE*/...).

As coleções ‘nested table’ das instâncias do tipo TabelaOR (tabela 10) foram armazenadas sob a forma de árvores B (opção ‘index-organized table’ do Oracle9i), para facilitar a busca por elementos das coleções. Uma vez que a pesquisa tenha localizado a chave primária, as colunas restantes estão presentes no mesmo local. ‘Index-organized tables’ são como tabelas convencionais no Oracle exceto pelo fato que os dados nas tabelas estão organizados como um índice árvore B construído sobre a chave primária da tabela. Uma tabela ‘index-organized’ armazena seus dados como se a tabela inteira fosse armazenada como um índice. Um índice normal apenas armazena as colunas indexadas no índice, uma tabela ‘index-organized’ armazena todas as colunas da tabela no índice.

Cada linha consiste de colunas chaves e não-chaves, todas armazenadas conjuntamente sob a forma de uma árvore-B, tornando a estrutura da tabela inteira numa organização de índice. Essa forma de organização provê como benefício o acesso aleatório rápido, sobre a chave primária, uma única varredura no índice é suficiente. Uma vez que um bloco folha é alcançado, as colunas chave como as não-chave podem ser recuperadas. Uma vez que a pesquisa tenha localizado a chave primária, as colunas restantes estão presentes na mesma localização. Isso elimina a necessidade de um identificador para cada linha da tabela, como poderia ser o caso com tabelas convencionais e estruturas de índices, evitando assim I/O adicional.

Para uma coluna ‘nested table’, Oracle internamente cria uma tabela de armazenamento que contém todas as linhas da ‘nested table’. As linhas pertencentes a uma instância da ‘nested table’ são identificadas por uma coluna NESTED_TABLE_ID. Se uma tabela simples é usada como armazenamento da coluna ‘nested table’, as linhas da ‘nested table’ ficaram provavelmente desagrupadas. Com o uso de ‘index-organized tables’ as linhas podem ser agrupadas baseadas na coluna NESTED_TABLE_ID. A especificação de armazenamento da ‘nested table’ como uma ‘index-organized table’ é mostrada a seguir.

```
CREATE TABLE E OF tabelaOR
NESTED TABLE aninhada STORE AS
    colecao_ntab((PRIMARY KEY(Nested_table_id))
                 ORGANIZATION INDEX)
    RETURN AS LOCATOR;
```

A supervisão dos planos de execução das consultas, gerados pelo otimizador de consultas do Oracle9i, foi feita com a ajuda do comando ‘Explain Plan’, permitindo verificar quais índices não estavam sendo utilizados, e por quê.

Embora não diretamente ligados à questão da sintonização, outros recursos do Oracle9i intensivamente utilizados foram as construções dinâmicas de diversos comandos de definição e de manipulação de dados — comandos ‘Execute Immediate’ —, necessários para conferir generalidade aos métodos Prism.

Capítulo

5

Avaliação Experimental

" Predicting the future is easy.
Getting it right is the hard part".
Howard Frank, 1945-.

Este capítulo descreve os experimentos realizados com as versões PrismR e PrismOR do PRISM, e também com uma versão 'stand-alone' do PRISM desenvolvida em Java — PrismJ[5] — aplicadas a algumas bases de dados de domínio público obtidas do repositório de dados 'Machine Learning Repository' (Murphy, 2002), amplamente utilizada em experimentos de mineração de dados.

5.1 Plano de Teste

Os dois principais objetivos dos experimentos realizados foram: (i) confirmar a viabilidade da integração do algoritmo PRISM com o SGBDOR Oracle9i, do ponto de vista do custo de processamento do algoritmo; e (ii) investigar em que medida uma estrutura sob a forma de tabela com colunas não-atômicas (TabelaOR), suportada pelo PrismOR, pode ser melhor para o desempenho do PRISM do que uma estrutura sob a forma de tabela puramente relacional ou com colunas atômicas (TabelaR), a cargo do PrismR.

No que concerne ao objetivo (i), os desempenhos dos métodos PrismR e PrismOR, ambos estreitamente acoplados ao Oracle9i, foram comparados com o

[5] Descrita no Apêndice C.

desempenho da versão ‘stand-alone’ PrismJ; no que diz respeito ao objetivo (ii), os métodos PrismR e PrismOR foram comparados entre si.

O ambiente computacional utilizado para a realização dos experimentos foi uma máquina com 256MB de memória principal, sistema operacional Windows NT Server 4.0, processador Pentium III, 850MHz. Foram utilizados o SGBD Oracle9i, Release 9.0.1, e o ambiente JBuilder 4.0, este último para o desenvolvimento dos códigos Java.

Inicialmente, para validar as implementações do PRISM em relação à sua funcionalidade, foram utilizadas as bases ‘lens24’ e ‘weatherData’ (Witten, 2000). As regras induzidas por PrismOR, PrismR e PrismJ, foram comparadas com os resultados do algoritmo de regras de classificação de (Witten, 2000), com resultados absolutamente idênticos.

As bases de dados de domínio público utilizadas para os testes comparativos do PrismOR, PrismR e PrismJ foram:

- Tic-tac-toe: contém o conjunto completo de configurações possíveis do tabuleiro de jogos de ‘tic-tac-toe’. Possui 958 instâncias e 9 atributos, que podem assumir três valores. Não há valores desconhecidos (‘null’).

- ‘Mushroom’: contém a descrição de exemplos hipotéticos de 23 espécies de cogumelos. Cada espécie é identificada como comestível, venenosa ou não comestível. Possui 8.124 instâncias e 22 atributos, sendo 2.480 valores desconhecidos, denotados pelo caractere ‘?’.

- ‘Nursery’: contém um conjunto de dados que foi derivado de um modelo de decisão hierárquico, originalmente desenvolvido para classificar aplicações em escolas de maternal. Possui 12.960 instâncias e 8 atributos, não há valores desconhecidos.

- ‘Connect-4’: contém as 8 posições legais possíveis do jogo ‘connect-4’, no qual nenhum jogador tenha ainda vencido e na qual a próxima jogada ainda não é fornecida. Possui 67.557 instâncias e 42 atributos com três valores possíveis para cada um, não há valores desconhecidos.

O principal critério para seleção das bases de dados foi a de que todos os atributos fossem categóricos, i. e. tivessem um número finito de valores discretos. O Prism não aceita atributos contínuos, na prática esta limitação pode ser superada pela discretização de atributos contínuos.

A tabela 11 resume as características dessas bases de dados. Maiores informações sobre elas podem ser obtidas em (Murphy, 2002).

TABELA 11. Características das bases de dados

Database	Número de Instâncias	Número de atributos	Número de Classes
Tic-tac-toe	958	09	02
Poço 7NA007RJS[6]	1.250	06	03
Mushroom	8.124	23	02
Nursery	12.960	09	05
Connect-4	67.557	43	03

5.1.1 Síntese do Desempenho dos Algoritmos Prism

As medidas dos tempos de execução para cada base de dados, e para cada versão do PRISM, podem ser observadas na tabela 12.

Para uma análise detalhada do desempenho das diferentes implementações do algoritmo PRISM foi selecionada a base de dados *connect-4.* A escolha teve

[6] Significado do código de nomeação de um poço, e. g. 7-NA-0007-RJ-S:
7 Poço para desenvolvimento (produção) do campo;
NA Sigla do campo Namorado;
0007 Sétimo poço do campo;
RJ Namorado é no Rio de Janeiro; S É submarino.

como métrica o tamanho da base de dados. Permitiu-nos, dessa forma, fragmentá-la em conjuntos de menor tamanho e acompanhar o tempo de execução mantendo-se fixas as características do conjunto de treinamento (atributos, valores por atributo, número de classes). Dividimos em arquivos de tamanhos contendo, em média, a seguinte proporção de instâncias: 10.000, 20.000, 30.000, 40.000, 50.000 e 60.000. Pôde-se perceber através da linha de tendência do gráfico 01 o comportamento que descreve as diferentes implementações submetidas a um mesmo domínio do problema.

TABELA 12. Tempos de execução (em milhares de segundos)

Conjunto de Treinamento	PRISM		PrismJ
	TabelaR	TabelaOR	
Tic-tac-toe	0, 018	0,022	0,010
Poço 7NA007RJS	0,011	0,006	0,002
Mushroom	0,916	0,989	0,155
Connect-4 (11.634 instâncias)	3,242	2,945	1,738
Nursery	4,954	3,057	3,059
Connect-4 (20.358 instâncias)	10,684	9,063	8,367
Connect-4 (23.976 instâncias)	15,141	12,589	12,452
Connect-4 (32.412 instâncias)	28,788	23,069	25,038
Connect-4 (41.038 instâncias)	47,604	37,061	42,320
Connect-4 (50.781 instâncias)	74,960	56,856	67,210
Connect-4 (67.557 instâncias)	149,473	116,640	126,233

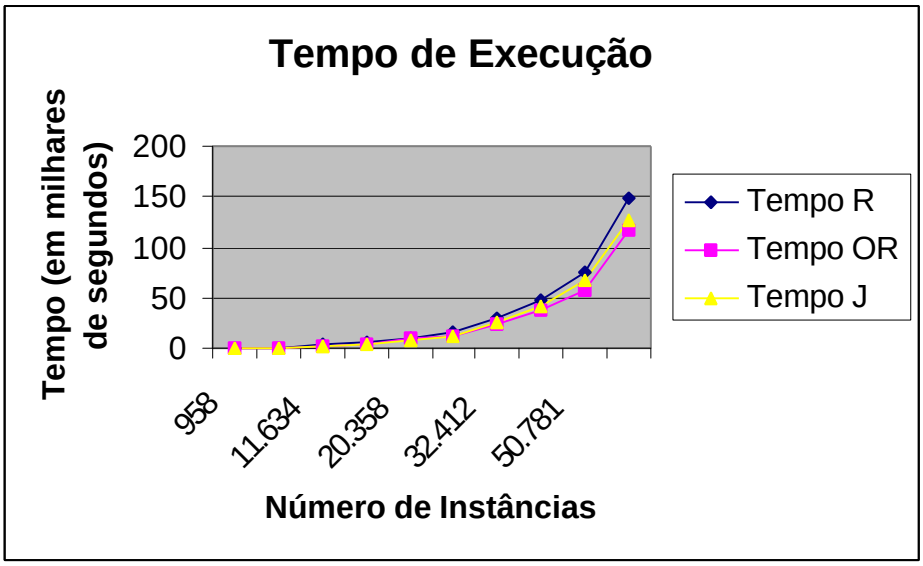


Gráfico 1. Comparação dos tempos de execução

Podemos perceber, pela tabela 12 e pelo gráfico 01, que a versão do PRISM com o tipo TabelaOR (ou versão objeto-relacional PrismOR, do PRISM) teve um desempenho pelo menos tão bom quanto o do PrismJ, com a tendência a se tornar mais eficiente que o PrismJ, à medida que os conjuntos de treinamento crescem em volume. A tendência é ainda mais clara quando se compara a versão do PRISM com o tipo TabelaR (versão relacional PrismR) com PrismOR, ou seja, PrismOR é superior a PrismR, para grandes volumes de dados.

Desta maneira, os dois objetivos traçados para os experimentos foram atingidos: (i) confirmar a viabilidade da integração do algoritmo PRISM com o SGBDOR Oracle9i, do ponto de vista do custo de processamento do algoritmo; e (ii) investigar em que medida uma estrutura sob a forma de tabela com colunas não-atômicas (TabelaOR), suportada pelo PrismOR, pode ser melhor para o desempenho do PRISM do que uma estrutura sob a forma de tabela puramente relacional ou com colunas atômicas (TabelaR), a cargo do PrismR.

Capítulo

6

Estudo de Caso: Regras de Classificação de Litofácies de Poços de Petróleo

"I do the very best I know how
the very best I can;
and I mean to keep doing so
until the end."
Abraham Lincoln

Neste capítulo serão apresentados experimentos e resultados com os dados do Campo Escola de Namorado para resolver o problema de identificação de litofácies. Para uma melhor compreensão discute-se inicialmente alguns conceitos importantes da industria de petróleo.

6.1 Petróleo: Gênese e Geologia

A utilização em larga escala dos derivados do petróleo, a partir da segunda metade do século XVIII, deu início a uma corrida desenfreada pela procura de acumulações petrolíferas. No início, sem nenhum conhecimento de como o petróleo existia na natureza, só a prática forneceu as bases necessárias para a procura do óleo.

O petróleo constitui a matéria-prima essencial à vida moderna, sendo o componente básico de mais de 6.000 produtos. Dele se produz a gasolina, o

combustível de aviação, o gás de cozinha, os lubrificantes, borrachas, plásticos, tecidos sintéticos, tintas — e até mesmo energia elétrica. O petróleo é responsável ainda por cerca de 34% da energia utilizada no Brasil e por 45% de toda a energia nos Estados Unidos[7].

O petróleo é formado e encontrado em regiões denominadas bacias sedimentares nas quais a crosta se afundou e se inundou possibilitando a acumulação de sedimentos (argilas, areias) trazidos pelos rios. A origem aceita para o petróleo é a orgânica, pela qual os tecidos de animais ou vegetais aquáticos do passado foram soterrados por sedimentos caídos no fundo de antigos mares ou lagos, sem que a presença de oxigênio os destruísse. Nem por isso a matéria orgânica de tais tecidos deixou de sofrer transformações drásticas, graças à temperatura e à pressão causada pelo profundo e prolongado soterramento. Assim, dos restos de antigos seres vivos, praticamente só restaram o carbono e o hidrogênio, que tiveram oportunidade de se combinarem até formarem o petróleo.

Para que grandes quantidades de petróleo se formem faz-se necessário a presença de pelo menos três fatores iniciais: existência de vida num passado longínquo, sedimentação por detritos (argilas principalmente) e o contínuo rebaixamento dessa acumulação de resíduos. O resultado disso é a formação de espessas camadas de sedimentos, mais tarde transformados em rochas sedimentares ricas em matéria orgânica e sujeitas a fortes pressões e altas temperaturas. Um quarto fator importantíssimo: o tempo. As condições descritas para transformação da matéria orgânica original em petróleo demanda milhões de anos.

O petróleo, entretanto, não é procurado nas rochas em que se formou. No longo processo de sua formação, ele foi adequadamente expulso da chamada rocha-mãe ou rocha-geradora, praticamente impermeável, para as rochas porosas e permeáveis porventura existentes em suas proximidades (acima, abaixo ou ao lado delas).

Para que se forme uma jazida de petróleo, pelo menos do ponto de vista comercial, é necessário que ele se concentre numa área restrita do subsolo, num campo de petróleo. É necessária uma trapa (armadilha) nas camadas porosas para que

[7] Disponível em: <http://www2.petrobras.com.br/minisite/sala_de_aula/ > Acessado em: fev./2002.

o petróleo fique retido e não suba até a superfície. O petróleo é menos denso que a água que embebe todas as rochas porosas existentes no subsolo, buscando sempre se sobrepor a ela.

Expulso da rocha geradora e fluindo pelos poros das rochas reservatório, o petróleo tende a chegar à superfície, só não o fazendo se houver, sobre a rocha porosa, uma camada impermeável. Esta camada, que pode ser a própria geradora do petróleo, é de fundamental importância. A ela dá-se o nome de rocha capeadora. As trapas, na realidade, só são efetivas se recobertas por uma rocha capeadora. Elas correspondem a deformações (domos, anticlinais), a grandes fraturas (falhas) ou a bruscas mudanças laterais nas camadas do subsolo, capazes de acumular petróleo, que nelas ficam retido.

O gás, muito menos denso que o óleo, também se separa em sua andança pelos poros das rochas, ocupando sempre o ponto mais alto das trapas. Os três fluidos que podem preencher os poros das rochas obedecem a uma ordem ditada por sua densidade: por baixo a água (geralmente muito mais salgada que a marinha), no meio o óleo e por cima o gás. São muito comuns os campos de óleo com uma capa de gás, enquanto a água subjacente é praticamente universal.

Para se perfurar um local a procura de petróleo, é preciso, antes, que os geólogos e geofísicos façam um complexo estudo geológico da bacia, para definir o ponto com melhores chances de ser perfurado. Mesmo com todas essas evidências, só depois da perfuração pode-se confirmar a existência de petróleo em uma determinada região. Ainda assim, essa ocorrência pode ser comercial ou não, dependendo do volume descoberto. Condições geológicas tão especiais determinaram a distribuição do petróleo de maneira bastante irregular na superfície terrestre.

Um poço para produzir apenas óleo, é revestido por tubos de aço que protegem suas paredes de alto abaixo. O revestimento, depois, é perfurado a tiros, lateralmente, na zona que existe o óleo. O gás porventura existente acima desta zona não será produzido, mas terá oportunidade de expandir-se e ajudará a saída do óleo.

A exploração para produção de petróleo é um jogo com apostas altas e prêmios elevados.

6.2 Perfuração

Desde a antigüidade o homem perfurou poços na crosta da terra, com diversos propósitos. A necessidade de revesti-los total ou parcialmente fez-se necessário para proteger suas paredes e garantir a integridade do trecho perfurado e a continuidade dos trabalhos.

Como os problemas encontrados durante a perfuração do poço exigem que este seja revestido antes de se atingir a profundidade final projetada, o poço é perfurado em fases, cada uma delas encerrada com a descida do revestimento e sua cimentação para proteger o trecho de poço aberto, sendo retomada a perfuração com diâmetro inferior na fase seguinte.

Existem basicamente dois métodos para a perfuração de um poço de petróleo: o percussivo e o rotativo. No primeiro, as rochas são golpeadas por uma broca de aço com movimentos alternados ocasionando faturamento ou esmagamento. Periodicamente é preciso remover os detritos cortados pela broca.

No método rotativo uma broca fragmenta a rocha quando comprimida e girada sobre ela. Os cascalhos são levados até a superfície por um fluido, o fluido de perfuração, que é bombeado por dentro da coluna de perfuração e retorna pelo espaço anular entre a coluna e o poço — o chamado sistema de circulação. O peso sobre a broca é aplicado através de tubos pesados chamados comandos, colocados logo acima da broca. Os meios de se impor rotação à broca podem ser vários: girando toda a coluna de perfuração através da mesa rotativa; girando apenas a broca através de um motor de fundo; girando toda a coluna de perfuração através do “Top Driver”.

A coluna de perfuração é formada pela junção de vários elementos tubulares, com as funções de: aplicar peso sobre a broca; transmitir rotação para a broca no método rotativo convencional e permitir a circulação do fluido de perfuração. Seus principais elementos constituintes são: elementos tubulares (tubos de perfuração, comandos, tubos de perfuração pesados) e elementos acessórios.

Os tubos de perfuração são tubos de aço unidos por soldas em suas extremidades.

A circulação é a operação de apenas circular fluido sem avançar na perfuração. É o que ocorre quando existe um acúmulo de cascalhos no espaço anular potencialmente causador de uma prisão na coluna de perfuração. Nestes casos, pára-se a perfuração e, com a broca um pouco acima do fundo do poço, promove-se a circulação do fluido de lama a fim de limpar o espaço anular.

O sistema de circulação é o responsável pelo bombeamento do fluido de perfuração a pressão e vazão adequadas para as operações de perfuração. Além disso, neste sistema estão os equipamentos que promovem o tratamento do fluido de perfuração após a saída do poço, livrando-o de sólidos e fluidos indesejáveis. Os principais elementos constituintes desse sistema são:

- Tanques de lama: feitos de chapas de aço para armazenamento da lama na superfície.
- Bombas de lama: responsáveis pelo fornecimento de energia ao fluido para circulação.
- Manifold: um conjunto de válvulas que recebe os mangotes de descarga das bombas.

Na perfuração convencional o poço é perfurado de tubo em tubo, de aproximadamente de 9 em 9 metros. Na perfuração com “Top Driver” o poço é perfurado de secções em secções de aproximadamente 27 metros.

Um poço de petróleo, dependendo de sua finalidade, pode ser classificado nas seguintes categorias (Lima, 2002):

1. Pioneiros: são perfurados com o objetivo de descobrir petróleo baseado em indicadores obtidos por métodos geológicos e/ou geofísicos.

2. Estratigráficos: são perfurados visando obter dados sobre a disposição seqüencial das rochas de subsuperfície, sem necessariamente dispor-se de informações geológicas completas da área. Estes dados serão utilizados para programações exploratórias

posteriores ou estudos específicos. Eventualmente, o poço poderá converter-se em produtor de óleo se descobrir novo campo.

3. Extensão: são perfurados fora dos limites provados de uma jazida visando ampliá-la ou delimitá-la. Poderá resultar como descobridor de uma nova jazida, independentemente daquela para qual foi locado.

4. Pioneiro adjacente: perfurado após a delimitação preliminar do campo visando descobrir novas jazidas adjacentes. Se tiver sucesso, será descobridor de nova jazida; se ficar provado que se trata da mesma jazida anterior será classificado novamente como poço de extensão.

5. Jazida mais rasa ou mais profunda: são poços perfurados dentro dos limites de um campo, visando descobrir jazidas mais rasas ou mais profundas daquela já conhecida.

6. Desenvolvimento: perfurados dentro dos limites do campo para drenar racionalmente o petróleo (atende aos preceitos econômicos e de espaçamento entre poços).

7. Injeção: perfurado com a intenção de injetar fluidos na rocha reservatório para ajudar na recuperação de óleo.

8. Especiais: são todos os poços que são perfurados sem o objetivo de procurar e produzir petróleo e que não estejam enquadrados em qualquer das categorias anteriores, e.g. poço para produção de água.

6.3 Testemunhagem

A testemunhagem é o processo de obtenção de uma amostra real de rocha de subsuperfície. O objetivo principal nesta operação é obter um testemunho com alterações mínimas nas propriedades naturais da rocha. Através da análise do testemunho é possível conseguir-se dados geológicos, de engenharia de reservatórios, de completação e de perfuração.

Quando o geólogo quer obter uma amostra da formação que está sendo perfurada, a equipe de sonda coloca uma coroa de testemunho no barrilete. A coroa de testemunho é uma broca com um furo no meio que permite que a broca corte o testemunho. O barrilete de testemunho é um tubo especial que geralmente mede 09, 18 ou 27 metros. O barrilete, que é o local em que irá se alojar o testemunho, é colocado na parte interna da coluna de perfuração. Durante a operação, à medida que a coroa avança, o cilindro de rocha não fragmentado é encapsulado pelo barrilete interno e posteriormente trazido à superfície.

Os testemunhos permitem que os geólogos analisem uma amostra real da rocha da formação. A partir dessa amostra eles muitas vezes podem saber se o poço será produtivo ou não. No testemunho com barrilete convencional, ao final de cada corte de um testemunho é necessário trazer a coluna à superfície através de uma manobra, o que aumenta o tempo e o custo da operação. Assim, foi desenvolvido o testemunho a cabo, onde o barrilete interno pode ser removido até à superfície sem a necessidade de se retirar a coluna (Thomas, 2001).

Algumas vezes pode haver a necessidade de se testemunhar alguma formação já perfurada. Nestes casos, emprega-se o método de testemunho lateral. Neste caso, cilindros ocos, presos por cabos de aço a um canhão, são arremessados contra a parede da formação para retirar amostras da rocha. Ao se retirar o canhão até a superfície, são arrastados os cilindros contendo as amostras retiradas da formação (Thomas, 2001).

A Figura 22 apresenta um exemplo dos dados de testemunho, onde a coluna 1 representa a profundidade em relação ao perfil, a coluna 2 representa a profundidade em relação à ferramenta de testemunho, a coluna 3 representa o número da caixa de armazenamento do testemunho, a coluna 4 representa o tamanho

das caixas em que o testemunho foi armazenado, a coluna 5 representa a granulometria, a coluna 6 representa as estruturas das rochas e, finalmente, a coluna 7 representa as litofácies encontradas.

Existem 28 tipos de litofácies nos testemunhos dos poços do Campo Escola de Namorado. A Tabela 13 apresenta os nomes das litofácies utilizadas nos experimentos. Os principais tipos de rocha no contexto da exploração de óleo são os arenitos e os folhelhos. Os arenitos são rochas que possuem alta porosidade e permeabilidade, ficando evidente que estes tipos de rocha podem constituir os reservatórios petrolíferos. Os folhelhos são rochas que oferecem as condições físico-químicas necessárias para a geração do óleo. Na maioria das vezes as rochas geradoras de óleo são folhelhos. Este tipo de rocha também constitui as rochas selantes, rochas que garantem o aprisionamento do óleo devido a sua baixa permeabilidade. No caso da Tabela 13, as litofácies 6, 7 e 8, por exemplo, podem apresentar indícios de hidrocarbonetos, observados a partir dos testemunhos. Como exemplo dos folhelhos na Tabela 13 temos as litofácies 14 e 20.

A testemunhagem é realizada para apenas alguns poços escolhidos estrategicamente, enquanto os perfis estão disponíveis para todos os poços. As informações detalhadas das litofácies de poços testemunhados raramente são extrapoladas e incorporadas quantitativamente durante o estágio de modelagem de um reservatório.

Geralmente, nos poços não testemunhados, as modelagens de litofácies são construídas usando as interpretações baseadas em procedimentos de correlação padrão, onde são usados apenas dados limitados de perfilagem de baixa resolução, como por exemplo, raios gama. Isto acontece porque, além da disponibilidade destes perfis, seus comportamentos são muito bem conhecidos.

TABELA 13. Litofácies presentes nos testemunhos do Campo Escola de Namorado

Litofácies	Nome
01	Interlaminado Lamoso Deformado
02	Conglomerado e Brechas Carbonáticas
03	Diamictito Areno Lamoso
04	Conglomerados Residuais
05	Arenito com Intraclastos Argilosos
06	Arenito Grosso Amalgamado
07	Arenito Médio Laminado
08	Arenito Médio Maciço Gradado
09	Arenito Médio Cimentado
10	Arenito / Folhelho Interestratificado
11	Arenito / Folhelho Finamente Interestratificado
12	Siltito Argiloso Estratificado
13	Interlaminado Siltito Argiloso e Marga
14	Folhelho Radioativo
15	Interlaminado Arenoso Bioturbado
16	Interlaminado Siltito e Folhelho Bioturbado
17	Marga Bioturbada
18	Ritmito
19	Arenito Glauconítico
20	Folhelho com Níveis de Marga Bioturbados
21	Arenito Cimentado com Intraclastos
22	Siltito Argiloso / Arenito Deformado
23	Arenito Médio / Fino Laminado Cimentado
24	Interestratificado Siltito / Folhelho Intensamente Bioturbados
25	Folhelho Carbonoso
26	Arenito Maciço muito fino
27	Siltito Areno-Argiloso
28	Interlaminado Siltito Folhelho

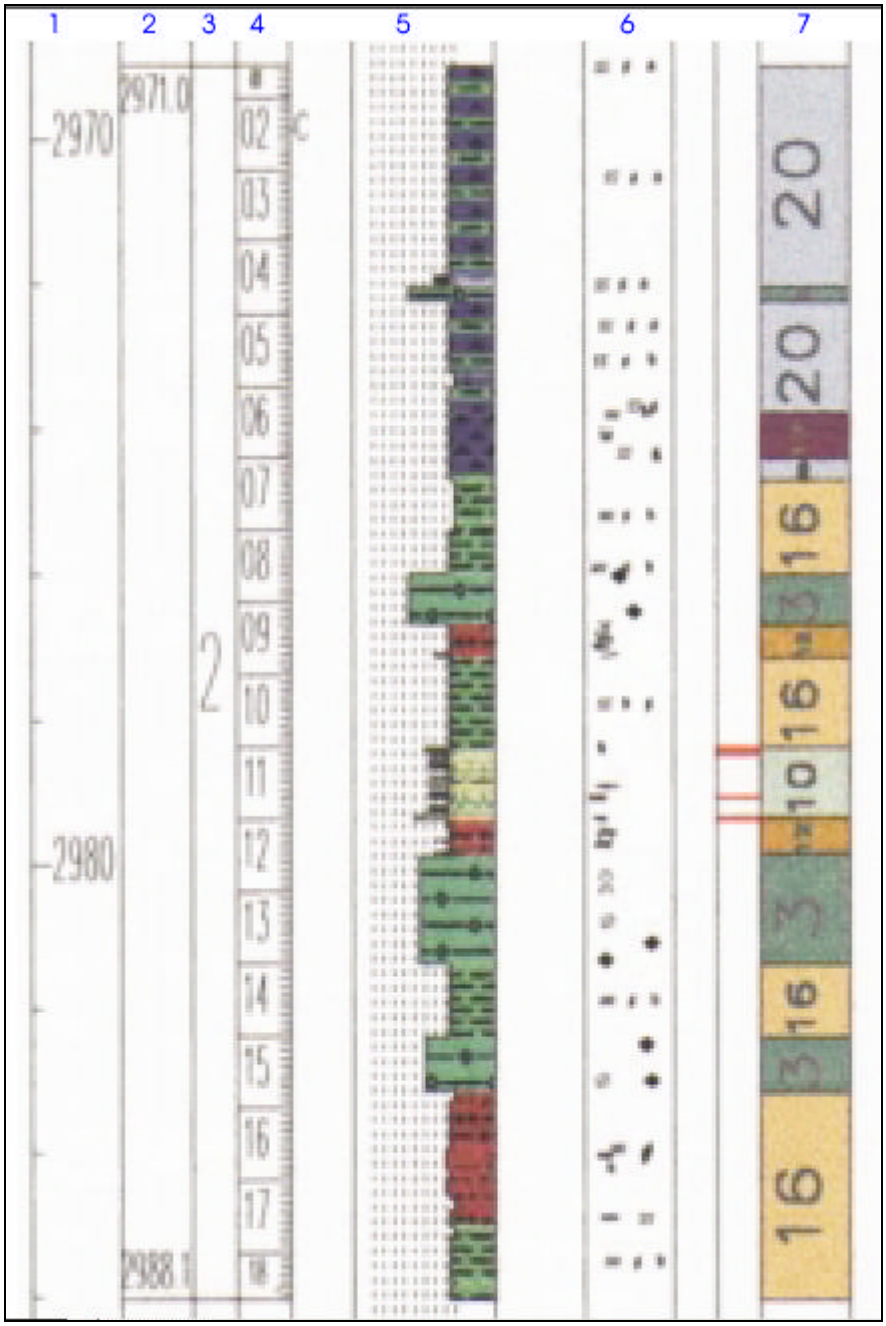


FIGURA 22. Testemunho

6.4 Perfilagem

O potencial de uma jazida petrolífera para ser avaliado em termos quantitativos e qualitativos (a valoração das reservas de óleo e gás) baseiam-se em vários testes principalmente na perfilagem a poço aberto.

A perfilagem é a operação realizada após a perfuração de uma fase do poço para obter uma imagem visual de uma ou mais características (propriedades) de uma formação em relação a profundidade. Tais imagens (perfis elétricos) são obtidas através do deslocamento contínuo de um aparelho de perfilagem (sonda) descido a cabo até a profundidade de interesse. As propriedades medidas podem ser elétricas (resistividade elétrica, potencial eletroquímico natural), acústicas (tempo de transito das ondas sonoras) e radioativas (radioatividade natural e induzida).

Através de interpretação dos dados de perfis pode-se conhecer a geometria do poço e da estrutura adjacente., e estimar a porosidade, litologia e resistividade das rochas e a resistividade, identificando os fluidos e das formações.

Os dados são transmitidos por meio de cabo de perfilagem até a unidade de controle, na superfície, na qual são decodificados e interpretados eletronicamente e registrados em fitas magnéticas ou em papel, na forma de uma série em que as propriedades medidas são mostradas na forma de curvas contínuas.

Os serviços de perfilagem são executados por companhias especializadas contratadas, sendo seu acompanhamento geralmente feito por um geólogo.

Para fazer uma perfilagem[8] em um poço, são usadas diversas ferramentas (sensores) acopladas a sofisticados aparelhos eletrônicos. Esses sensores são introduzidos poço adentro, registrando, a cada profundidade, as diversas informações relativas às características físicas das rochas e dos fluidos em seus poros. As ferramentas utilizam diversas características e propriedades das rochas, que podem ser elétricas, nucleares ou acústicas. Com os sensores elétricos, detecta-se, por exemplo, a resistividade das rochas e a identificação das mesmas se dá através de comparações dos valores obtidos na perfilagem com os valores das resistividades de diversas rochas conhecidas e determinadas em testes de laboratório. Com os sensores

[8] Processo utilizado para obtenção de algumas propriedades das formações, fundamentais para sua caracterização e avaliação econômica.

nucleares, detecta-se a intensidade de radioatividade das rochas e dos fluidos em seus poros, podendo-se inferir a composição mineralógica das mesmas. Com as ferramentas acústicas, ultra-sons são emitidos em uma ponta da ferramenta a intervalos regulares e detectados em sensores na outra ponta. O tempo que o sinal sonoro levou para percorrer esta distância fixa e conhecida (chamado de tempo de trânsito) através da parede do poço (ou seja, pela rocha) é medido e gravado no perfil.

Os técnicos analisam o grande volume de informações gerado nas etapas de perfilagem, reunindo um razoável conhecimento sobre a espessura, profundidade e comportamento das camadas de rochas existentes numa bacia sedimentar.

Com base nesse conhecimento, são escolhidos os melhores locais para se perfurar na bacia. Porém, mesmo com o rápido desenvolvimento tecnológico, ainda hoje não é possível determinar a presença de petróleo a partir da superfície. Os métodos científicos podem, no máximo, sugerir que certa área tem ou não possibilidades de conter petróleo, mas jamais garantir sua presença. Esta somente será confirmada pela perfuração dos poços exploratórios. Por isso, a pesquisa para a exploração de petróleo é tida como uma atividade de alto risco. No Brasil, a exploração de petróleo é tarefa muito complexa, não só pela extensão da área sedimentar, superior a seis milhões de quilômetros quadrados, como pela natureza das bacias, que possuem um tipo de rocha de difícil exploração e com pouco petróleo. No mar, a dificuldade se mede pela profundidade da água em que se encontram os maiores reservatórios já localizados, de 200 a 2 mil metros de lâmina d'água (PETROBRAS, 2002).

Existem vários tipos de perfis utilizados para as mais diversas aplicações, todos com o objetivo de avaliar melhor as formações geológicas quanto à ocorrência de uma jazida comercial de hidrocarbonetos. Os perfis mais comuns são: Potencial Espontâneo, Reflexão, Radioatividade, Resistividade, Porosidade, Densidade e Caliper.

1. Potencial Espontâneo (SP): é o registro da diferença de potencial entre um eletrodo móvel descido dentro do poço e outro fixo na superfície. Este perfil permite determinar as camadas permoporosas, calcular a argilosidade das

rochas, determinar a resistividade da água da formação e auxiliar na correlação de informações com poços vizinhos.

2. Reflexão (Sônico ou DT): descreve o comportamento das ondas sísmicas, medindo as diferenças nos tempos de trânsito de uma onda magnética através das rochas (velocidade). As ondas após penetrarem na crosta são refletidas, em contatos de duas camadas de diferentes propriedades elásticas retornam à superfície, sendo então detectadas por sensores. Fornece detalhes da estrutura da crosta, bem como de propriedades físicas das camadas que a compõem. É utilizada, também, para estimativas de porosidade, correlação poço a poço, estimativas do grau de compactação das rochas ou estimativas das constantes elétricas, detecção de fraturas e apoio à sísmica.

3. Radioatividade (GR): detectada por raios gama, esse atributo descreve as propriedades radioativas de certos minérios constituintes da formação geológica. Auxiliando na identificação da litologia e no cálculo do volume de argilas (argilosidade).

4. Resistividade (ILD): fornece informações sobre corpos rochosos que tenham condutividade elétrica anômala. É empregado pela engenharia para estudos de salinidade de lençóis de água subterrânea. Fornece leitura aproximada de resistência, através da medição de campos elétricos e magnéticos induzidos nas rochas. Valores altos de resistividade podem ser indicativos de hidrocarbonetos, enquanto que resistividades muito baixas atestam a presença de água salgada. Atributo presente em todos os poços.

5. Porosidade (NPHI): medida através da técnica denominada neutrônico auxilia na detecção de hidrocarbonetos leves ou gás. Os neutros emitidos colidem com os elementos que constituem as rochas perdendo sua energia. Quanto mais poroso o material, menor será a taxa de colisão.

6. Densidade (RHOB): detecta os raios gama defletidos pelos elétrons dos elementos componentes das rochas, após terem sido emitidos por uma fonte situada dentro do poço. Além da densidade das camadas, permite o cálculo da porosidade e a identificação das zonas de gás. É utilizado também como apoio à sísmica.

7. Caliper: fornece o diâmetro do poço. É aplicado no cálculo do volume de cimento para tampões ou cimentação do revestimento, apoio a operações de teste de formação, controle de qualidade de perfis e indicações das condições do poço em um determinado intervalo.

6.5 Base de Dados do Campo Escola de Namorado, da Petrobrás

Namorado foi o segundo campo descoberto na Bacia de Campos em dezembro de 1975. Da mesma forma que as cidades, os sítios geológicos — no caso, as bacias sedimentares — recebem nomes de acidentes geográficos ou cidades próximas. Este procedimento é seguido internacionalmente e regido pelo “Código de Nomenclatura Estratigráfica”, adotado pelos geólogos.

Seguiram-se diversas outras descobertas, que foram constituindo as dezenas de campos de óleo e gás batizados com nomes de peixes e outras criaturas do mar, como um imenso e rico cardume, da maior importância para a economia brasileira (PETROBRAS, 2002).

Os dados, fornecidos para universidades e centros de pesquisa pela Agência Nacional de Petróleo, são disponibilizados em CD-ROM e em fita magnética de 8mm (dados sísmicos). Constam no CD-ROM as seguintes informações:

- Perfis de poços: são cinqüenta e seis (56) arquivos de extensão “las” (formato ASCII) contendo dados de perfis de poços, em formato padrão da indústria do petróleo.

- Mapas base e secções: são arquivos de extensão “cgm” (arquivos de imagem).

- Outros dados: são arquivos que possuem descrições de testemunhos, dados petrofísicos, estudo de propriedades dos fluidos e históricos das vazões de óleo, gás e água. Estes arquivos encontram-se no formato de imagem “tif”.

- Mapas e Listagens: são mapas de posicionamento do campo e de poços, incluindo listagens de perfuração de poços em formato pdf.

- Legislação: lei nº 9.478, de 6 de agosto de 1997, que dispõe sobre a política energética nacional, as atividades relativas ao monopólio do petróleo, institui o Conselho Nacional de Política Energética e a Agência Nacional do Petróleo e dá outras providências.

Os dados caracterizam os perfis geofísicos de 56 (cinqüenta e seis) poços. A base de dados é composta por arquivos, nomeados por códigos numéricos (os poços dos quais foram extraídos), e descrevem cinco respostas petrofísicas de sedimentos de rochas: reflexão, radioatividade, resisitividade, porosidade e densidade.

6.6 Inferência de Regras de Classificação de Litofácies de Poços de Petróleo

Durante a perfuração de um poço de petróleo, as amostras de calha que saem na lama de perfuração são continuamente analisadas, sendo registradas as profundidades associadas a cada tipo de rocha identificada. Após a perfuração são descidas várias ferramentas para medir algumas propriedades da formação, os perfis.

Com os perfis registrados e com base nas observações das amostras de calha, realizadas durante a perfuração, as curvas de perfilagem são interpretadas por um geólogo para saber que tipos de rocha existem naquela formação e a que profundidade. Desta forma pode-se saber se há ou não indícios de hidrocarbonetos naquela formação (Thomas, 2001).

A caracterização de reservatórios de petróleo é uma tarefa muito complexa devido à heterogeneidade dos mesmos. Os reservatórios heterogêneos são conhecidos pelas grandes mudanças em suas propriedades dentro de uma pequena área. Estas mudanças ocorrem, principalmente, devido às idades geológicas distintas, à natureza da rocha e aos ambientes deposicionais. A caracterização de reservatórios tem um papel muito importante na indústria do petróleo, particularmente para o sucesso econômico do gerenciamento e dos métodos de produção.

Para predizer o desempenho do reservatório de forma confiável, é necessário fazer uma descrição precisa do mesmo. A testemunhagem é uma das técnicas mais antigas e ainda praticadas para extrair características de um reservatório. No entanto, testemunhar todos os poços em um campo muito grande pode ser economicamente inviável, além disso, o tempo consumido pode ser muito grande. Já os perfis de poços, estão disponíveis para todos os poços.

A identificação manual de litofácies de um reservatório de petróleo é um processo intensivo que envolve o gasto de uma quantidade considerável de tempo por parte de um especialista experiente. O problema se torna muito mais difícil à medida que aumenta o número de perfis (medidas de determinadas propriedades da formação geológica) simultâneos a serem analisados.

A identificação de litofácies pode ser feita a partir de dados de perfis ou de dados de testemunhos. Os dados de perfis e testemunhos carregam informações diferentes sobre a litologia, por isso a determinação de litofácies a partir destas duas fontes é diferente.

A identificação de litofácies a partir de dados de perfis não permite um nível de detalhamento idêntico ao do testemunho. Isto ocorre porque, nos testemunhos, as litofácies são identificadas e classificadas com base em características medidas em pequena escala e em escalas microscópicas. No entanto,

para este problema, o alto nível de detalhamento dos testemunhos prejudica o processo de aprendizagem.

Devido aos problemas citados acima, torna-se de grande utilidade fazer o reconhecimento automático das litofácies (tipo de formação da rocha) de um reservatório utilizando-se perfis de poços.

6.7 Conjuntos de Treinamento e Teste

Com o intuito de resolver o problema de reconhecimento de litofácies, foram selecionados os arquivos de perfis e descrição dos testemunhos do Campo Escola de Namorado. Nem todos os poços que possuíam dados de perfis, possuíam dados de testemunho, mas todos os poços que possuíam dados de testemunho, possuíam dados de perfis. Para alguns poços com dados de testemunho, os dados de perfis estavam incompletos, portanto, foram selecionados apenas os poços que, além do testemunho, possuíam todos os perfis. No final restaram apenas 9 poços com descrições dos testemunhos, além dos seguintes perfis: Raios Gama (GR), Sônico (DT), Indução (ILD), Densidade (RHOB) e Porosidade Neutrônica (NPHI), conforme descreve a Tabela 14. Além desses atributos, conta-se ainda com o atributo *profundidade*, que fornece os intervalos medidos (em metros) nos quais os testes foram realizados. Esse atributo está presente em todos os poços.

Visando inferir o conhecimento necessário para identificar automaticamente as litofácies dos poços, unimos os dados de testemunho com os dados de perfis (figura 23) dos poços selecionados. Desta forma foi formado um único grande banco de dados.

Como apresentado na figura 22, os dados de testemunho encontram-se em uma planilha construída manualmente. Essa forma de apresentação torna difícil a identificação, em muitos casos, do tipo de litofácies de uma determinada região do testemunho, ou ainda, difícil definir com exatidão a profundidade exata de uma determinada litofácies no poço — camadas muito finas — assim como determinar com exatidão a profundidade de troca de um tipo de litofácies para outra.

```
~VERSION INFORMATION
 VERS.           2.0: CWLS Log ASCII Standard - Version 2.0
 WRAP.            NO: One Line per Depth Step
~WELL INFORMATION
#MNEM.UNIT              DATA            DESCRIPTION OF MNEMONIC
#----------             ----            -----------------------
 STRT.M          3300.0000              :Start Depth
 STOP.M          3550.0000              :Stop  Depth
 STEP.M             0.2000              :Step
 NULL.             -99999.0             :Null Value
 COMP  .   PETROLEO BRASILEIRO S/A      :Company
 WELL  .   7NA  0009D RJS               :Well
 FLD   .   NAMORADO                     :Field
 LOC   .                                :Location
 STAT  .   RIO DE JANEIRO               :State
 SRVC  .                                :Service Company
 DATE  .                                :Log Date
 API   .   742810091600                 :API Code
~CURVE INFORMATION
 DEPT.M                                 :Measured Depth
 DT.                                    :01
 GR.                                    :02
 ILD.                                   :03
 NPHI.                                  :04
 RHOB.                                  :05
~ASCII LOG DATA
  3300.000    83.2539    49.2444     2.2952    20.9873     2.4951
  3300.200    83.4924    48.8516     2.3049    20.8120     2.5011
  3300.400    83.1250    48.8594     2.3328    19.9844     2.5047
  3300.600    81.9609    51.9453     2.3613    18.2422     2.5127
  3300.800    80.9688    52.9180     2.3826    17.6157     2.5260
  3301.000    81.0469    47.8125     2.3257    18.9727     2.5244
  3301.200    81.8438    43.5625     2.3012    19.9529     2.4954
  3301.400    82.5391    41.8164     2.3012    19.5820     2.4663
  3301.600    82.7734    42.0742     2.3012    19.4648     2.4598
  3301.800    83.4844    44.2148     2.3012    19.7383     2.4653
   . . .       . . .       . . .       . . .     . . .       . . .
```

FIGURA 23. Exemplo de arquivo no formato .LAS

Para a indução de regras de classificação, foi realizado treinamento com 2/3 (dois terços) dos poços selecionados, reservando-se restante para testes, técnica *holdout*. As instâncias de treinamento e teste (1.323) foram inseridas na tabela do banco de dados, para indução das regras de classificação. Os poços utilizados nesta fase foram:

TABELA 14. Poços selecionados

Poço	Profundidade
3NA 0001A RJS	2950,0 - 3200,0
3NA 0002 RJS	2975,0 - 3200,0
3NA 0004 RJS	2950,0 - 3150,0
7NA 0007 RJS	3025,0 - 3275,0
7NA 0011A RJS	3000,0 - 3200,0
7NA 0012 RJS	2970,0 - 3175,0
7NA 0037D RJS	3170,0 - 3400,0
4RJS 0042 RJ	3000,0 - 3215,0
4RJS 0234 RJ	3150,0 - 3352,2

As regras induzidas são armazenadas na tabela de *regras* e consultadas durante a fase de teste para estimativa da acurácia do modelo gerado.

Os valores dos atributos da tabela de teste (excetuando-se o atributo de classificação) foram comparados aos atributos correspondentes da tabela de regras. Dessa forma, pretende-se verificar a existência de regras que apresentem condições equivalentes na tabela de teste. Verificando-se a existência de uma regra para a instância de teste em questão, o valor assumido pelo atributo de classificação é comparado ao valor induzido na fase de treinamento. Dessa maneira, computamos as instâncias que foram corretamente e incorretamente classificadas, e consequentemente o número de instâncias não classificadas.

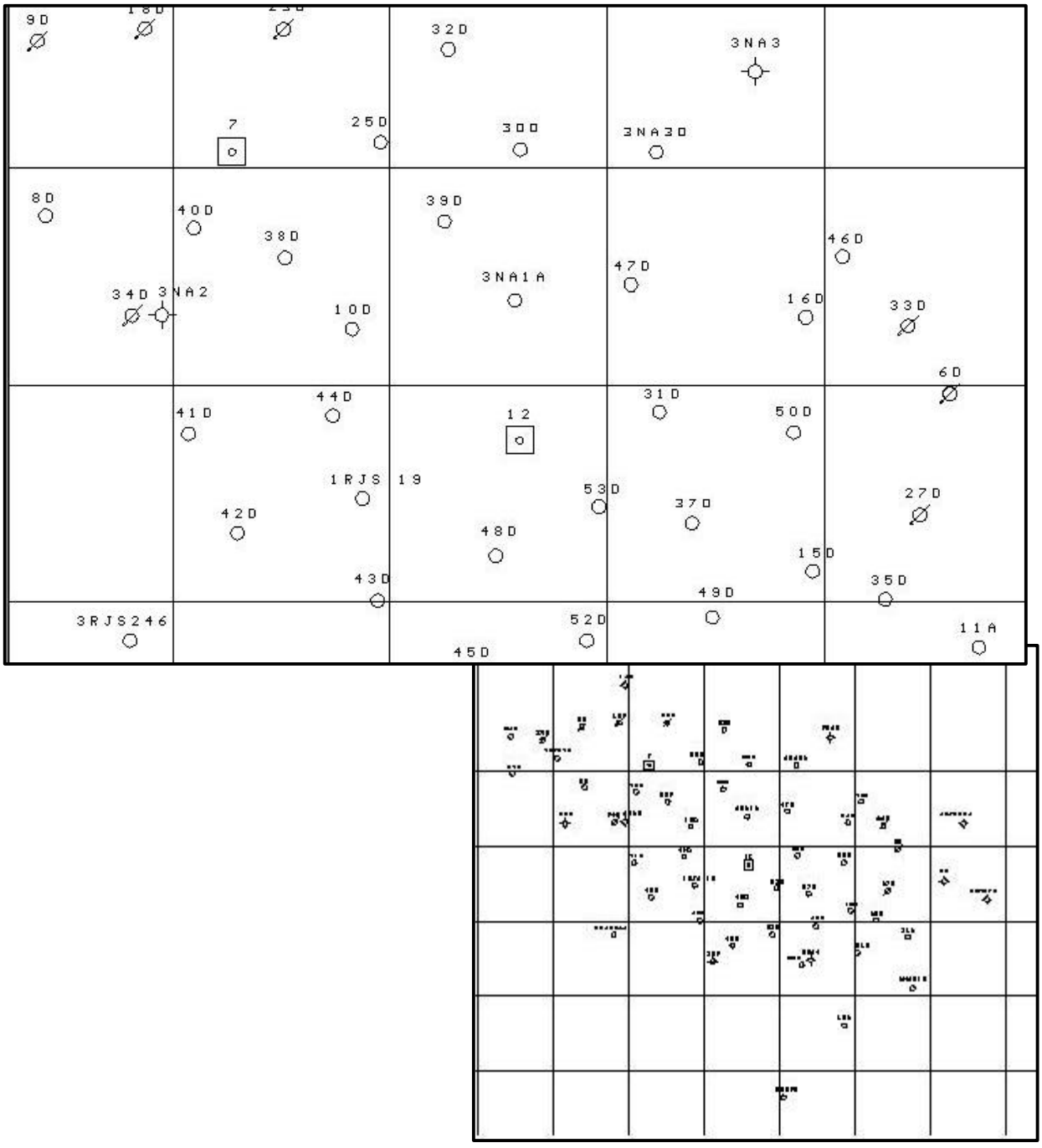


FIGURA 24. Mapa base do Campo Escola de Namorado com foco am alguns poços selecionados

6.8 Resultados Obtidos

Para o conjunto de treinamento e teste com 1.323 instâncias, foram selecionadas 850 instâncias para treinamento e o restante foi dedicado a testes. Os atributos constituintes dessa base de dados foram os perfis: sônico, indução, densidade, raios gama, porosidade neutrônica, densidade, os respectivos intervalos de profundidade; associado a cada instância fez-se corresponder a litofácie encontrada para o intervalo perfurado. Discretizou-se os dados de cada atributo em cinco faixas distintas. O atributo *litofácie* foi eleito como atributo alvo da classificação, assumindo 22 (vinte e dois) valores distintos.

A taxa de acurácia alcançada pela execução do algoritmo PRISM sobre essa base de dados foi de 83,25% instâncias corretamente classificadas; 6,27% instâncias incorretamente classificadas e 10,48% instâncias não classificadas. Valores corretamente classificados, corresponde às instâncias contidas nos dados de teste que foram cobertas pelas regras construídas na fase de treinamento. Valores incorretamente classificados, corresponde às instâncias que embora apresentem uma regra formada na fase de treinamento, o valor predito por esta regra discorda do apresentado nos dados de teste. Valores não classificados, corresponde às regras não construídas a partir dos dados de treinamento, não permitindo a classificação dos valores que apareceram nos dados de teste.

As regras inferidas pelo algoritmo PRISM encontram-se no apêndice B desta dissertação.

Para explicar o que ocorreu com o valor encontrado para a acurácia de treinamento/teste, devemos relembrar a estratégia de geração de regras adotada pelos algoritmos que seguem a abordagem ‘covering’, como é o caso do algoritmo PRISM. O critério básico para escolher uma condição a ser adicionada ao antecedente de uma regra é encontrar aquela que faça a regra cobrir o maior número possível de exemplos positivos[9], o que concomitantemente permite reduzir os exemplos

[9] Isto é, exemplos que verificam a regra.

negativos[10]. Assim, um conjunto de condições que leva a uma regra exata — só exemplos positivos — será preferido a qualquer outro conjunto com ao menos um exemplo negativo, não importando quantos exemplos positivos a regra possa cobrir. Por exemplo, se existir uma regra que cubra um único caso, e que este seja positivo (acurácia do treinamento = 1/1 = 100%), a regra será preferida sobre uma outra que cubra mil casos positivos, mas que apresente apenas um caso negativo (acurácia do treinamento = 1000/1001 = 99,90%).

A constatação é então que os algoritmos 'covering' supervalorizam regras exatas ou com acurácia de treinamento de 100%, as quais podem não ser sempre relevantes, o que acontece quando os exemplos positivos são pouco freqüentes ('overfitting'). Para que regras exatas possam ser relevantes, é preciso também que elas sejam suficientemente genéricas, ou que cubram muitos casos de treinamento.

Em síntese, a acurácia do treinamento precisa ser confirmada pela acurácia medida com o conjunto de testes, ou acurácia de teste. Se uma regra não é suficientemente genérica, muito provavelmente ela terá baixa acurácia de teste.

Em relação aos dados para os quais o algoritmo PRISM inferiu regras pouco confiáveis, o que ocorreu foi exatamente que as regras inferidas, embora exatas, não eram suficientemente genéricas. Em conseqüência, as acurácias dos testes com essas regras não atingem valores de 100% de acerto.

[10] Ou exemplos que não verificam a regra.

Capítulo

7

Conclusões

"Whatever you do will be insignificant, but it is very important that you do it."
Mahatma Gandhi

Esta dissertação descreve a integração do algoritmo de regras de classificação PRISM com SGBDs, e particularmente sua integração com o SGBDOR Oracle9i. Uma das implementações é suportada por estruturas de dados puramente relacional, enquanto a outra explora o tipo "nested table" do modelo objeto-relacional do Oracle9i.

Precedendo as implementações, relacional e objeto-relacional, do PRISM estreitamente integradas com o Oracle9i, foi definida conceitualmente uma versão genérica e orientada a objeto do PRISM, utilizando para isto os formalismos UML (parte gráfica) e ODMG (parte analítica).

No esquema conceitual proposto, o Prism é um método polimorfo de duas classes PrismR e PrismOR, que estendem a classe abstrata Prism. O estado de um objeto PrismR é definido grosso modo por um conjunto de treinamento, e um conjunto de regras de classificação inferido do conjunto de treinamento pela versão relacional do método Prism. Por sua vez, o estado de um objeto PrismOR é definido grosso modo por um conjunto de treinamento, e um conjunto de regras de classificação inferido do conjunto de treinamento pela versão objeto-relacional do método Prism.

A visão conceitual do Prism foi então transformada em “object types” e “object tables” do Oracle9i.

Foram realizados testes comparativos de desempenho das versões relacional e objeto-relacional do PRISM, e da versão “stand-alone” em Java do PRISM. No cômputo geral, as versões objeto-relacional e Java do PRISM tiveram desempenhos equivalentes e melhores que os da versão relacional do PRISM, com a tendência de a versão objeto-relacional do PRISM ser melhor que a versão Java, à medida que os conjuntos de treinamento crescem em volume.

A utilização de estruturas objeto-relacional (coleções do tipo “nested table” do Oracle9i) representou, em termos de desempenho, grande vantagem quando comparadas com estruturas puramente relacionais. Apesar de estarmos ainda na infância da tecnologia objeto-relacional, os resultados obtidos podem ser considerados como altamente promissores para a consolidação dessa nova tecnologia.

O algoritmo Prism foi utilizado para geração de regras de classificação de litofácies de poços de petróleo sobre a base de dados do Campo Escola de Namorado. A identificação de litofácies é uma das etapas fundamentais do processo de caracterização de um reservatório. A caracterização de reservatórios é muito importante no processo de avaliação econômica de um poço. Assim, fica evidente a necessidade de pesquisas relacionadas a este tema. Os dados foram divididos em conjuntos de treinamento e teste. A taxa de acurácia atingida foi de 83,25%.

Continuidade a esta pesquisa em integração de algoritmos de regras de classificação com SGBDs pode ser feita através da exploração de SGBDs e hardware paralelos.

Referências Bibliográficas e Bibliografia

AGÊNCIA NACIONAL DO PETRÓLEO — ANP. Base de dados do Campo Escola de Namorado – CD-ROM, Bacia de Campos, Rio de Janeiro-RJ, Brasil.

AGRAWAL, R. et al. An Interval Classifier for Database Mining Applications. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 18th, 1992, San Francisco. *Proceedings…* San Francisco: Morgan Kaufmann Publishers, 1992. p. 560-573.

AGRAWAL, R.; SHIM, K. Developing Tightly-Coupled Data Mining Applications on a Relational Database System. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY IN DATABASES AND DATA MINING, 2nd, 1996, Portland-Oregon. *Proceedings.…* Portland: MIT PRESS, 1996a. p. 146-160.

AGRAWAL, R. et al. The Quest Data Mining System. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY IN DATABASES AND DATA MINING. 2nd, 1996b, Portland. *Proceedings.…* Portland: AAAI Press, 1996b. p. 244-249.

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining Association Rules between Sets of Items in Large Databases. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 5th, 1993, Washington. *Proceedings.…* Washington: SIGMOD, 1993. p. 207-216.

AGRAWAL, R.; SRIKANT, R. Fast Algorithms for Mining Association Rules. In: VLDB CONFERENCE, 20th, Santiago. *Proceedings.…* Santiago/Chile: Morgan Kaufmann, 1994. p. 487-499.

BEZERRA, E. et al. An Analysis of the Integration between Data Mining Applications and Database Systems. In: INTERNATIONAL CONFERENCE ON DATA MINING, 2nd, 2000, Cambridge-UK. *Proceedings*… Cambridge: WIT Press, 2000. p. 151-160.

BRAMER, M. A. Automatic Induction of Classification Rules from Examples Using N-Prism. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE BASED SYSTEMS AND APPLIED ARTIFICIAL INTELLIGENCE, 19th, 1999, London. *Proceedings.*…London: British Conference Series/Springer-Verlag, 1999. p. 99-121.

BRAMER, M. A. Inducer: a Rule Induction Workbench for Data Mining. In:

IFIP WORLD COMPUTER CONGRESS CONFERENCE ON INTELLIGENT INFORMATION PROCESSING, 2000, Beijing. *Proceedings*… Beijing: Publishing House of Electronics Industry, 2000. p. 499-506.

BRATKO, Ivan. Handling noise in Inductive Logic Programming. In: International Workshop on Inductive Logic Programming. *Proceedings*… Tokyo, Japan. 1992.

BREIMAN, L., J. Friedman, R. Olshen and C. Stone. Classification and Regression Trees, Pacific Grove: Wadsworth and Brooks, 1984, Monterey, CA.

CABENA, P. (Ed.) *Discovering Data Mining*: From Concept to Implementation, New Jersey: Prentice Hall-USA, 1997. 200 p.

CARVALHO, Juliano V. de; SAMPAIO, Marcus C.; MONGIOVI, Giuseppe. Utilização de Técnicas de Data Mining para o Reconhecimento de Caracteres Manuscritos. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 14º, 1999, Florianópolis-SC. *Proceedings*... Florianópolis: Sociedade Brasileira de Computação, 1999. p.235-249.

CATTELL, R. G. (Ed.) et al. The Object Data Standard: ODMG 3.0. San Diego: Morgan Kaufmann Publishers, San Francisco/USA, 2000. 280 p.

CENDROWSKA, J. PRISM: An Algorithm for Inducing Modular Rules, International Journal of Man-Machine Studies, Number 4, Volume 27, p. 349-370, 1987.

CESTNIK, G., KONONENKO, I., BRATKO, I. Assistant 86: A knowledge acquisition tool for sophisticated users. In Bratko, I., & Lavrac, N. (Eds.), Progress in Machine Learning. Sigma Press, 1987.

CLARK, Peter.; NIBLETT, Tim. The CN2 Induction Algorithm, Machine Learning Journal , Volume 3, p. 261-283, 1989.

COHEN, W. W. Fast effective rule induction. In Prieditis, A., & Russell, S. (Eds.). *Proceedings* of the 12th International Conference on Machine Learning

FAYYAD, U. (Ed.); PIATETSKI-SHAPIRO (Ed.). *Advances in Knowledge Discovery in Databases and Data Mining*, Massachusetts: AAAI Press, The MIT Press, 1996a. 611 p.

FAYYAD, U. et al. From Data Mining to Knowledge Discovery in Databases, AI Magazine, Rhode Island, V. 17, n. 3, p. 37-54, Fall, 1996b.

FAYYAD, U. Data Mining and Knowledge Discovery in Databases: Implications for Scientific Databases. In: International Conference on

Scientific and Statistical Database Management, 9th, Olympia/Washington/USA, 1997. *Proceedings*… Olympia: IEEE Computer Society, 1997. pp. 2-11.

FRANÇA, Júnia Lessa. Manual para Normalização de Publicações técnico-Científicas. Belo Horizonte: Editora UFMG, 2001. 211p.

FRAWLEY, W. J. et al. Knowledge discovery in databases: An Overview. AI Magazine 13(3), p. 57-70. 1992.

FREITAS, A. *Generic, Set-Oriented Primitives to Support Data-Parallel Knowledge Discovery in Relational Database Systems.* 1997, 233 f. Tese (Doutorado em Informática), Department of Computer Science, University of Essex, England.

FREITAS, A. Notas de Aula da disciplina Data Mining. Programa de Pós-Graduação em Informática Aplicada PUC-PR, 2000.

FREITAS, A. Understanding the Crucial Differences between Classification and Discovery of Association Rules – A Position Paper. In: SIKDD EXPLORATIONS, ACM, Volume 2, Issue 1, 2000. pp 1-5.

FÜRNKRANZ, Johannes. A Brief Introduction to Knowledge Discovery in Databases. *ÖGAI-Journal* 14(4): 14-17, 1995.

FÜRNKRANZ, Johannes. Separate-and-conquer Rule Learning. Artificial Intelligence Review. Journal, V. 13, N. 1, p. 3-54, 1999.

GANTI, V. et al. Mining Very Large Databases. Computer IEEE magazine, Volume: 32 Issue: 8, p. 38-45. August/1999.

GIFFRIDA, Giovanni. et al. Mining Classification Rules from Datasets with Large Number of Many-Valued Attributes. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY — EDBT, 7th, 2000, Konstanz, Germany. *Proceedings*... Konstanz: Springer, 2000. p. 335-349.

HAN, Jiawei et al. DBMiner: Interactive Mining of Multiple-Level Knowledge in Relational Databases. In: International Conference on Management of Data. SIGMOD'96, Montreal. Proceedings… Montreal: ACM Press, 1996, pp. 550.

HAN, Jiawei et al. DMQL: A data mining query language for relational databases. In Proc. of the 1996 SIGMOD workshop on research issues on data mining and knowledge discovery, Montreal, Canada, 1996.

HAN, J.; KAMBER, M. *Data Mining:* Concepts and Techniques. San Diego: Morgan Kaufmann Publishers, San Francisco/USA, 2001. 550 p.

HIPP, Jochen; GUNTZER, Ulrich; GRIMMER, Udo. Integrating Association Rule Mining Algorithms with Relational Database Systems. In: International Conference on Enterprise Information Systems — ICEIS, 3th, 2001, Setúbal/Portugal. Proceedings… Setúbal: DBLP, 2001, pp.130-137.

IMIELINSKI, T.; MANNILA, H. A Database Perspective on Knowledge Discovery. In: COMMUNICATIONS OF THE ACM. V. 39, p. 58-64, 1996.

LARMAN, C. *Applying UML and Patterns*: An Introduction to Object-Oriented Analysis and Design. New Jersey: Prentice Hall, 1998. 509p.

LI, Cen.; BISWAS, Gautam. Knowledge-based scientific discovery from geological database. In: THE FIRST INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY FROM DATABASES, 1st, 1995, Canada. *Proceedings*… Canada: AAAI Press, 1995. p. 204-209.

LIMA, Heitor Rodrigues de Paula. Fundamentos de Perfuração. Apostila do curso de Engenharia de Petróleo — Perfuração. Centro de Desenvolvimento de Recursos Humanos do Norte-Nordeste (CEN-NOR). 2002. 331 p.

LOPES, Carlos Henrique Pereira. Classificação de Registros em Banco de Dados por Evolução de Regras de Associação Utilizando Algoritmos Genéticos. *Departamento* de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro. p. 136. 1999.

MEHTA, M. SLIQ: A fast scalable classifier for data mining. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY — EXTENDING DATABASE *TECHNOLOGY* (EDBT), 5th, 1996, Avignon, France. *Proceedings*… Avignon: Springer, Vol. 1057, 1996. p.18-32.

MICHALSKI, R. S. et al. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In: NATIONAL CONFERENCE ON ARTIFITIAL INTELLENGENCE. 5th. *Proceedings.*.. Philadelphia, PA, 1986. p. 1041-1045.

MICHALSKI, R. S. On the quasi-minimal solution of the covering problem. In: INTERNATIONAL SYMPOSIUM ON INFORMATION PROCESSING — FCIP, 5th, 1969, Bled. *Proceedings*… Bled/Yugoslavia: [s.d.], Vol. A3 (Switching Circuits), 1969. p. 125-128.

MICHALSKI, R. S. Pattern recognition and rule-guided inference. IEEE Transactions on Pattern Analysis and Machine Intelligence 2, 349-361. 1980.

MONGIOVI, G. *Uso de Relevância Semântica na Melhoria da Qualidade dos Resultados Gerados pelos Métodos Indutivos de Aquisição de Conhecimento a* partir *de Exemplos*. 1995. Tese Doutorado, Universidade Federal da Paraíba, Campina Grande.

MURPHY, P. M. *UCI Repository of Machine Learning Databases and Domain Theories*. [http://www.ics.uci.edu/~mlearn/MLRepository.html.] Department of Information and Computer Science, University of California at Irvine. Irvine, CA.

NESTOROV, S.; TSUR, S. Integrating Data Mining with Relational DBMS: A Tightly-Coupled Approach. In: NEXT GENERATION INFORMATION TECHNOLOGIES AND SYSTEMS — NGITS, 4th, 1999, Zikhron-Yaakov/Israel. *Workshop.…* Zikhron-Yaakov: Springer, 1999. p. 295-311.

NETZ, Amir. et al. Integration of Data Mining and Relational Databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 26th, 2000, Cairo/Egypt. *Proceedings.…* Cairo: Morgan Kaufmann, 2000. p. 719-722.

ONODA, M.; EBECKEN N. Implementação em Java de um Algoritmo de Árvore de Decisão Acoplado a um SGBD Relacional. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 16º, 2001, Rio de Janeiro-R.J. *Proceedings...* Rio de Janeiro: Sociedade Brasileira de Computação, 2001. p.55-64.

ORACLE. Oracle*9i:* Oracle Documentation Library, Release 1 (9.0.1), June 2001.

PAGALLO, G.; HAUSSLER, D. Boolean feature discovery in empirical learning. Machine Learning, 5(1), 1990. p. 71-99.

PETROBRAS. Petróleo na Natureza. Documento eletrônico, disponível em: <http://www2.petrobras.com.br/minisite/sala_de_aula/petroleo/>. Acessado em: fev./2002.

PYLE, Dorian. Data Preparation for Data Mining. Morgan Kaufmann Publishers, San Francisco/California. 1999. 540 p.

QUINLAN, J.R. Learning efficient classification procedures and their application to chess end-games, in: Machine Learning: An Artificial Intelligence Approach, Vol. 1, Michalski, J.S., Carbonell, J.G., and Mirchell, T.M. (Eds.), Morgan Kaufmann, Palo Alto, CA, pp. 463-482, 1983.

QUINLAN, J. R. Learning First-Order Definitions of Functions. In: Journal of Artificial Intelligence Research, Volume 5,p. 139-161, 1996.

QUINLAN, J. R. *C4.5:* Programs for Machine Learning. San Francisco/USA: Morgan Kaufmann, 1993. 302 p.

QUINLAN, J. R. Learning logical definitions from relation, Machine Learning, 5, 1990. p. 239-266.

RAJAMANI, K. et al. Efficient Mining for Association Rules with Relational

Database Systems. In: INTERNATIONAL DATABASE ENGINEERING AND APPLICATIONS SYMPOSIUM — IDEAS, 1999, Montreal. *Proceedings* ... Montreal: IEEE Computer Society, 1999. p. 148-155.

RASTOGI, R.; SHIM. K. Public: A Decision Tree Classifier that Integrates Building and Pruning. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 24th, 1998, San Francisco. *Proceedings.*.. San Francisco: Morgan Kaufmann, 1998. p. 404-415.

SAMPAIO, M. C. Notas de Aula da disciplina Data Mining. Programa de Pós-Graduação em Informática. Universidade Federal da Paraíba – Campus II. 2001.

SANTOS, Rafael Valle dos. et al., Predição de propriedades de perfis com redes neurais: uma aplicação no campo escola de namorado, bacia de campos, Brasil. In: CONGRESSO BRASILEIRO DE P&D EM PETRÓLEO E GÁS – EXPOPETRO, 1º., 2001, Natal-RN. *Proceedings.*.. Natal: Sociedade Brasileira de Química Regional RN. 2001. p. 143.

SARAWAGI, S.; THOMAS, S.; AGRAWAL, R. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA — COMAD, 1998, Seattle. *Proceedings.*..Seattle: ACM Press, 1998. p.343-354.

SAVASERE, Ashoka. An Efficient Algorithm for Mining Association Rules in Large Databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES — VLDB, 21st, 1995, Zurich/Swizerland. *Proceedings*… Zurich: VLDB Journal, 1995. p. 432-444.

SBG — Sociedade Brasileira de Geologia. Documento eletrônico, disponível em <www.sbgf.org.br/>, acessado em Maio/2002.

SHAFER, John C. SPRINT: A sclable parallel classifier for data mining. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES — VLDB, 22nd, 1996, Mumbai/India. *Proceedings*… Mumbai: Morgan Kaufmann, 1996. p. 544-555.

SILVA, Eduardo Bezerra da; XEXÉO, Geraldo B. Uma Primitiva para dar Suporte à Obtenção de Resumos Estatísticos para Classificação em Mineração de Dados. In: Simpósio Brasileiro de Banco de Dados, XIV, Florianópolis. *Proceedings*… Sociedade Brasileira de Computação, Florianópolis, 1999.

SOUSA, Mauro. et al. Data Mining: A Tightly-Coupled Implementation on a Parallel Database Server. In: INTERNATIONAL CONFERENCE AND WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS — DEXA, 9th, 1998, Vienna/Austria. *Proceedings*...

Vienna: IEEE CS Press, 1998, p.711-716.

SOUZA, Roberto Gonçalves de. PETRÓLEO: Histórias das Descobertas e o Potencial Brasileiro. Labouré Lima, Niterói/RJ, 1997. 272 p.

SRIKANT, Ramakrishnan; AGRAWAL Rakesh. Mining Quantitative Association Rules in Large Relational Tables. INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 6th, 1996, Montreal. *Proceedings*… Montreal: SIGMOD, 1996. p.1-12.

THOMAS, J. E. (Org.) *Fundamentos de Engenharia de Petróleo.* Rio de Janeiro: Editora Interciência, 2001. p. 271.

THOMAS, S.; SARAWAGI, S. Mining Generalized Association Rules and Sequential Patterns Using SQL Queries. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING — KDD, 1998, New York City. *Proceedings.*.. New York City: AAAI Press, 1998. p. 344-348.

TOK, W. H. et al. Predator-Miner: Ad hoc Mining of Associations Rules Within a Database Management System. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 18th, 2002, San Jose-USA. *Proceedings.*.. San Jose: IEEE Press, 2002. p. 327 –328.

TURBAN, Efraim; ARONSON, Jay E. *Decision Support Systems and Intelligent Systems.* New Jersey: Fifth Edition, Prentice Hall - USA, 1998. 890 p.

VASCONCELOS, B. SAMPAIO, Marcus C. Mineração Eficiente de Regras de Classificação com Sistemas de Banco de Dados Objeto-Relacional. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 17º, 2002, Gramado-RS. *Proceedings...* Gramado: Sociedade Brasileira de Computação, 2002. p. 278-292.

WANG, M. IYER, B. VITTER, J. S. Scalable Mining for Classification Rules in Relational Databases. INTERNATIONAL DATABASE ENGINEERING AND APPLICATIONS SYMPOSIUM — IDEAS, 1998, Cardiff/UK. *Proceedings.*.. Cardiff: IEEE Computer Society, 1998. p. 58-67.

WITTEN, I.H.; FRANK, E. *Data Mining:* Practical Machine Learning Tools and Techniques with Java Implementations. San Diego: Morgan Kaufmann Publishers, 2000. 369 p.

WITTEN, Ian H. Weka: Practical Machine Learning Tools and Techniques with Java Implementations. In: WORKSHOP ON EMERGING KNOWLEDGE ENGINEERING AND CONNECTIONIST-BASED INFORMATION SYSTEMS — ICONIP/ANZIIS/ANNES'99, 1999, Dunedin/ New Zealand. *Proceedings*… Dunedin: [s.d.], 1999. p. 192-196.

YOSHIZAWA, T. et al. SQL Based Association Rule Mining using Commercial RDBMS (IBM DB2). In: INTERNATIONAL CONFERENCE ON DATA WAREHOUSING AND KNOWLEDGE DISCOVERY (DAWAK), 2nd, 2000, London. *Proceedings.*..London: Springer-Verlag, 2000. p. 301-306.

ZAKI. Mohammed J.; OGIHARA, Mitsunori. Theoretical foundations of association rules. In: WORKSHOP ON RESEARCH ISSUES IN DATA MINING AND KNOWLEDGE DISCOVERY — DMKD, 3rd, 1998, Washington. *Proceedings*… Seattle: SIGMOD Record, Volume 27, Issue 2, 1998. p. 7:1-7:8

ZAKI. Mohammed Javeed. Parallel Classification for Data Mining on Shared-Memory Multiprocessors. In: ICDE, 15th, 1999, Sydney/Australia. *Proceedings*… Sydney: IEEE Computer Society Press, 1999. p. 198-205.

ZHANG, H. *Mining and Visualization of Association Rules over Relational DBMSs.* 2000. 100 f. Dissertação (Mestrado em Informática) – Graduate School of the University of Florida.

Apêndice A

Implementação do PRISM em PL/SQL

Os scripts dos códigos para a construção da integração do algoritmo PRISM ao SGBD Oracle9i estão aqui descritos.

Definição dos tipos

```
-- Varray que armazena os valores de todos os atributos.
CREATE OR REPLACE TYPE atributosVarray AS
    VARRAY(10) OF VARCHAR2(20);
/
```

```
-- Varray que armazena os valores possíveis de cada atributo.
CREATE OR REPLACE TYPE valoresVarray AS
    VARRAY(30) OF VARCHAR2(20);
/
```

```
-- Tipo que armazena a quantidade de valores existentes
-- para cada atributo.
CREATE OR REPLACE TYPE quantAtributoValorObj AS OBJECT(
  atributo        VARCHAR2(20),
  quantidade     INTEGER
)
/
```

```
-- Varray que armazena os valores possíveis de
-- classificação.
CREATE OR REPLACE TYPE classeVarray IS
     VARRAY(10) OF VARCHAR2(20);
/
```

```
-- Auxilia no deslocamento no array de valores.
CREATE OR REPLACE TYPE deslocamento AS VARRAY(4)
    OF quantAtributoValorObj
/
```

```
-- Varray que armazena uma regra formada a cada iteração do
-- algoritmo.
CREATE OR REPLACE TYPE regrasVarray IS
```

```
     VARRAY(10) OF VARCHAR2(20);
/

-- tipo que armazenará as condições de pares atributo-valor
-- e as respectivas ocorrências.
CREATE OR REPLACE TYPE condicao AS OBJECT(
  atributo     VARCHAR2(20),
  valor        VARCHAR2(20),
  P            INTEGER,
  T            INTEGER
);
```

```
-- Um array de condições armazenará a ocorrência de cada
-- par atributo-valor na base-exemplo.
CREATE OR REPLACE TYPE condVarray AS
     VARRAY(50) OF condicao;
/
```

```
CREATE TYPE tabelaR AS OBJECT(
  age             VARCHAR2(15),
  spectacle       VARCHAR2(15),
  astigmatism     VARCHAR2(15),
  tear       VARCHAR2(15),
  classe          VARCHAR2(8)
)
/
```

```
CREATE TABLE e OF tabelaR;
```

```
CREATE TABLE P OF tabelaR;
```

```
CREATE TABLE regras OF tabelaR;
```

```
CREATE OR REPLACE TYPE atributos AS OBJECT(
  age          VARCHAR2(15),
  spectacle   VARCHAR2(15),
  astigmatism VARCHAR2(5),
  tear   VARCHAR2(8)
);
/
```

```
CREATE OR REPLACE TYPE colecao AS TABLE OF atributos;
/
```

```
CREATE OR REPLACE TYPE tabelaOR AS OBJECT(
   aninhada   colecao,
   Classe    VARCHAR2(5)
);
```

```
/
```

```
-- Colecao de referencias para o objeto tabelaOR
-- em que, cada objeto pertence a uma classe ('hard',
-- 'soft', 'none')

CREATE OR REPLACE TYPE TabelaORref AS TABLE OF REF tabelaOR
/
```

```
CREATE TABLE e OF tabelaOR
NESTED TABLE aninhada STORE AS colecao_ntab;
```

```
-- conter os elementos da classe atual
CREATE TABLE p OF tabelaOR
NESTED TABLE aninhada STORE AS p_ntab;
```

Definição do tipo Prism

```
CREATE OR REPLACE TYPE Prism AS OBJECT (
  AtributosConjTreinamento   atributosVarray,
  dominiosAtribConjTrein     valoresVarray,
  cardinalidadeDominios      deslocamento,
  classes                    classeVarray,
  regras                     regrasVarray,
  acuraciaAtribValor         condVarray,


  MEMBER PROCEDURE NovaRegra(classeCorrente IN VARCHAR2),
  MEMBER PROCEDURE IncluiTermo (indice IN INTEGER,
                       regras IN OUT regrasVarray,
                       acuraciaAtribValor IN condVarray),
  MEMBER PROCEDURE GravaNovaRegra (regras IN regrasVarray),
  MEMBER FUNCTION  CalculaAcuraciaAtribValor (
                        acuraciaAtribValor IN condVarray)
         RETURN INTEGER,

  MEMBER FUNCTION  ImprimeNovaRegra(regras IN regrasVarray)
         RETURN VARCHAR2

) NOT INSTANTIABLE NOT FINAL;
```

Definição do tipo PrismR

```
CREATE OR REPLACE TYPE PrismR UNDER Prism (
  ConjTreinamento  VARCHAR2(10),

  MEMBER PROCEDURE Prisma,
  MEMBER PROCEDURE TodosOsTermos (
                   regras IN OUT regrasVarray,
                   acuraciaAtribValor IN OUT condVarray),
  MEMBER PROCEDURE RemoveInstancias (
                   regras IN regrasVarray)
);
/


CREATE TABLE PrismsR OF PrismR;
```

Definição do tipo PrismOR

```
CREATE OR REPLACE TYPE PrismOR UNDER Prism (
  -- colecao de objetos do tipo referencia para tabelaOR
  ConjTreinamento   TabelaORref,

  MEMBER PROCEDURE Prisma,
  MEMBER PROCEDURE TodosOsTermos(regras IN OUT regrasVarray,
                   acuraciaAtribValor IN OUT condVarray),
  MEMBER PROCEDURE RemoveInstancias (regras IN regrasVarray)
);
/


CREATE TABLE PrismsOR OF PrismOR
    NESTED TABLE ConjTreinamento STORE AS conj_ntab;
```

Corpos dos Métodos do Tipo Prism

```
CREATE OR REPLACE TYPE BODY Prism AS
  MEMBER PROCEDURE NovaRegra( classeCorrente IN VARCHAR2) IS
  BEGIN
      -- Crie uma regra R com o lado esquerdo vazio
      FOR i IN SELF.regras.FIRST .. SELF.regras.LAST LOOP
        regras(i) := ' ';
      END LOOP;

      -- R prediz a classe C
      regras(SELF.regras.LAST-1) := 'Classe';
      regras(SELF.regras.LAST):= classeCorrente;
  END;



  MEMBER PROCEDURE IncluiTermo(indice IN INTEGER,
                           regras IN OUT regrasVarray,
                           acuraciaAtribValor IN condVarray)
  IS
    posicao   INTEGER;
    atributos  atributosVarray;
  BEGIN
    FOR cont IN 0..(SELF.AtributosConjTreinamento.LAST-2)
    LOOP
       IF SELF.AtributosConjTreinamento(cont+1) =
          acuraciaAtribValor(indice).atributo
       THEN
          posicao:=2*cont+1;
       END IF;
    END LOOP; --cont

    -- Adicione A=v a R
    regras(posicao):= acuraciaAtribValor(indice).atributo;
    regras(posicao+1):= acuraciaAtribValor(indice).valor;
  END  IncluiTermo;



  MEMBER PROCEDURE GravaNovaRegra (regras IN regrasVarray)
  IS
   query_str   VARCHAR2(1000);
   valores     VARCHAR2(300);
  BEGIN
    query_str := ' ';
    valores  := ' ';

      FOR  rg IN regras.FIRST .. (regras.LAST)/2 LOOP
```

```
        IF valores = ' '  THEN
          IF regras(2*rg) != ' ' THEN
            valores  := valores || '''' ||
                        regras(2*rg) || '''';
          ELSE
            valores  := valores || 'NULL';
          END IF;
        ELSE
          IF regras(2*rg) != ' ' THEN
            valores  := valores || ', ' || '''' ||
                        regras(2*rg) || '''';
          ELSE
            valores  := valores || ', ' || 'NULL';
          END IF;
        END IF;
      END LOOP;
      query_str := 'INSERT INTO regras VALUES( '
                   || valores || ')';
      EXECUTE IMMEDIATE query_str;
END;



MEMBER FUNCTION  CalculaAcuraciaAtribValor(
                 acuraciaAtribValor IN condVarray)
RETURN INTEGER   IS
    acuracia       REAL:=0;
    relacaoPT      REAL:=0;
    relacaoPTmaior REAL:=0;
    indiceMaior    INTEGER;
BEGIN

     --Selecione A e v para maximizar a acurácia p/t
     relacaoPT:=0;
     indiceMaior := acuraciaAtribValor.FIRST;
     FOR c IN SELF.acuraciaAtribValor.FIRST ..
              SELF.acuraciaAtribValor.LAST
     LOOP
       IF SELF.acuraciaAtribValor(c).P=0 OR
          SELF.acuraciaAtribValor(c).T=0
       THEN
           relacaoPT := 0;
       ELSE
          relacaoPT := SELF.acuraciaAtribValor(c).P/
                       SELF.acuraciaAtribValor(c).T;

          IF relacaoPT = 1 THEN
             IF SELF.acuraciaAtribValor(c).P  >
                SELF.acuraciaAtribValor(indiceMaior).P  THEN
                relacaoPTmaior :=
                              SELF.acuraciaAtribValor(c).P/
                              SELF.acuraciaAtribValor(c).T;
```

```
                indiceMaior := c;
              END IF;
            END IF;

           IF relacaoPT > relacaoPTmaior THEN
              relacaoPTmaior := relacaoPT;
              indiceMaior := c;
           END IF;
        END IF; -- =0
      END LOOP; -- c
      RETURN indiceMaior;
  END CalculaAcuraciaAtribValor;


  MEMBER FUNCTION  ImprimeNovaRegra(regras IN regrasVarray)
  RETURN VARCHAR2 IS
   regraInduzida VARCHAR2(2000);
   aux  INTEGER;
  BEGIN
      regraInduzida := ' SE ';
      FOR i IN (regras.FIRST-1) .. ((regras.LAST-4)/2) LOOP
        IF regras(2*i+2) != ' ' THEN
          regraInduzida := regraInduzida ||
                           regras(2*i+1)|| ' =  ' ||
                           regras(2*i+2);
        END IF;
        aux := 2*i+4;
        IF aux < regras.LAST THEN
           IF regras(2*i+2)!=' ' AND regras(aux) != ' ' THEN
             regraInduzida := regraInduzida || ' AND ';
           END IF;
        END IF;
      END LOOP;

       regraInduzida :=  regraInduzida || ' ENTAO ' ||
                         regras(SELF.regras.LAST-1) ||
                         ' = ' || regras(SELF.regras.LAST);
       RETURN regraInduzida;
  END;

END;
```

Corpos dos Métodos do Tipo PrismR

```
CREATE OR REPLACE TYPE BODY PrismR AS
   MEMBER PROCEDURE Prisma  IS
       classeAtual     VARCHAR2(20);
       numInstancia    INTEGER;
       acuracia        REAL;
       maior           INTEGER;
       haMaisAtributos BOOLEAN;
       statement       VARCHAR2(2000);
   BEGIN
     -- Para cada classe C
     FOR c IN SELF.classes.FIRST .. SELF.classes.LAST LOOP
        classeAtual := SELF.classes(c);

        -- Inicialize o conjunto de instancias
        DELETE FROM P;
        statement := 'INSERT INTO P SELECT /*+ CHOOSE */ *
                      FROM ' || SELF.ConjTreinamento || '
                      WHERE classe = '  || ''''||
                            ClasseAtual || '''';
        EXECUTE IMMEDIATE  statement;

        SELECT /*+ CHOOSE */ COUNT(*) INTO numInstancia
        FROM P;

        WHILE (numInstancia != 0) LOOP
          -- Crie uma regra R com o lado esquerdo vazio
          self.NovaRegra(classeAtual);

          -- Até R estar perfeita
          haMaisAtributos := FALSE;
          LOOP
            TodosOsTermos(SELF.regras,
                          SELF.acuraciaAtribValor);

            -- Adicione A=v a R
            SELF.IncluiTermo(maior, regras,
                             acuraciaAtribValor);

            -- Selecione A e v para maximizar a acurácia P/T
            maior := SELF.CalculaAcuraciaAtribValor(
                                  SELF.acuraciaAtribValor);

            IF acuraciaAtribValor(maior).P=0 OR
                acuraciaAtribValor(maior).T=0 THEN
                Acuracia := 0;
            ELSE
                Acuracia := acuraciaAtribValor(maior).P/
```

```
                          acuraciaAtribValor(maior).T;
         END IF;

         IF HaMaisAtributos=FALSE OR acuracia=1  THEN
            EXIT;
         END IF;
       END LOOP; -- regra esta perfeita?

        SELF.GravaNovaRegra(regras);
        RemoveInstancias(regras);

        SELECT /*+ CHOOSE */ COUNT(*) INTO numInstancia
        FROM P;

      END LOOP; -- while (numInst !=0)
 END LOOP; -- cada classe C
END Prisma;



MEMBER PROCEDURE TodosOsTermos(
                  regras IN OUT regrasVarray,
                  acuraciaAtribValor IN OUT condVarray)
IS
 query_str    VARCHAR2(2000);
 where_clause VARCHAR2(300);
 from_clause  VARCHAR2(300);
 elemento     INTEGER;
 Limite       INTEGER;
BEGIN

  FOR i IN acuraciaAtribValor.FIRST ..
           acuraciaAtribValor.LAST
  LOOP
      acuraciaAtribValor(i).atributo := ' ';
      acuraciaAtribValor(i).valor := ' ';
      acuraciaAtribValor(i).P := 0;
      acuraciaAtribValor(i).T := 0;
  END LOOP;

  elemento:=1;
  -- Para cada atributo A e cada valor v
   FOR a IN 0..(SELF.cardinalidadeDominios.LAST-1) LOOP
      IF (SELF.regras(2*a+1) !=
          SELF.cardinalidadeDominios(a+1).atributo) THEN
         limite := 0;
         FOR i IN 1..(a+1) LOOP
           limite:= limite +
               SELF.cardinalidadeDominios(i).quantidade;
         END LOOP;
```

```
-- Considere adicionar a condicao A=v a R
FOR t IN (limite –
  SELF.cardinalidadeDominios(a+1).quantidade+1)
  .. limite LOOP
   acuraciaAtribValor(elemento).atributo:=
      SELF.cardinalidadeDominios(a+1).atributo;
   acuraciaAtribValor(elemento).valor :=
      SELF.dominiosAtribConjTrein(t);
   acuraciaAtribValor(elemento).P := 0;
   acuraciaAtribValor(elemento).T := 0;

   from_clause := ' ';
   where_clause := ' ';
   query_str := ' ';

 FOR  rg IN regras.FIRST .. ((regras.LAST-2)/2)
 LOOP
     IF where_clause = ' ' THEN
       IF regras(2*rg) != ' ' THEN
          Where_clause := where_clause ||
           SELF.AtributosConjTreinamento(rg) ||
           ' = ' || '''' || regras(2*rg) ||'''';
       END IF;

      IF acuraciaAtribValor(elemento).atributo =
         SELF.AtributosConjTreinamento(rg)  THEN
          Where_clause := where_clause ||
             SELF.AtributosConjTreinamento(rg)||
             ' = '  ||''''||
       acuraciaAtribValor(elemento).valor||'''';
       END IF;
    ELSE
       IF regras(2*rg) != ' ' THEN
          Where_clause:=where_clause|| ' AND '||
            SELF.AtributosConjTreinamento(rg) ||
            ' = ' || '''' ||regras(2*rg) ||'''';
       END IF;
       IF acuraciaAtribValor(elemento).atributo
        = SELF.AtributosConjTreinamento(rg) THEN
          Where_clause:=where_clause||' AND ' ||
              SELF.AtributosConjTreinamento(rg)
              ||' = '||''''||
              acuraciaAtribValor(elemento).valor
              ||'''';
       END IF;
    END IF;
 END LOOP;

 IF where_clause != ' ' THEN
   FOR c IN SELF.classes.FIRST ..
            SELF.classes.LAST
   LOOP
```

```
              IF SELF.classes(c) !=
                 regras(SELF.regras.LAST) THEN
                 query_str := ' SELECT /*+ CHOOSE */
                               count(*) FROM ' ||
                               SELF.ConjTreinamento
                              ||' WHERE '||where_clause;

                 EXECUTE IMMEDIATE query_str INTO
                       acuraciaAtribValor(elemento).T;
              END IF;
            END LOOP; -- para cada classe

            query_str := ' SELECT /*+ CHOOSE */
                          COUNT(*) FROM p WHERE'
                         || where_clause;

            EXECUTE IMMEDIATE query_str INTO
                   acuraciaAtribValor(elemento).P;
         ELSE
            acuraciaAtribValor(elemento).P :=0;
            acuraciaAtribValor(elemento).T :=0;
         END IF;

         Elemento := elemento+1;

         END LOOP; -- for t
      END IF;  --regra jah contem este atributo.
   END LOOP; -- for a
 END TodosOsTermos;



MEMBER PROCEDURE RemoveInstancias(regras IN regrasVarray)
IS
   query_str       VARCHAR2(2000);
   where_clause    VARCHAR2(300);
   atributos       atributosVarray;
BEGIN
     where_clause := ' ';

     FOR  rg IN regras.FIRST .. ((regras.LAST-2)/2) LOOP
       IF where_clause = ' ' THEN
          IF regras(2*rg) != ' ' THEN
             Where_clause := where_clause ||
                  SELF.AtributosConjTreinamento(rg) ||
                  ' = ' ||  '''' || regras(2*rg) ||'''';
          END IF;
       ELSE
          IF regras(2*rg) != ' ' THEN
             Where_clause := where_clause || '  AND ' ||
                    SELF.AtributosConjTreinamento(rg) ||
```

```
                    ' = ' || '''' || regras(2*rg) ||'''';
            END IF;
         END IF;
      END LOOP;

      query_str := ' ';
      query_str := 'DELETE FROM (SELECT /*+ CHOOSE */ *
                    FROM P WHERE ' ||
                    where_clause || ' ) ';
      EXECUTE IMMEDIATE query_str;

   END RemoveInstancias;
END;
```

Corpos do Tipo PrismOR

```
CREATE OR REPLACE TYPE BODY PrismOR AS
   MEMBER PROCEDURE Prisma  IS
     objeto          tabelaOR;
     classeAtual     VARCHAR2(20);
     numInstancia    INTEGER;
     acuracia        REAL;
     maior           INTEGER;
     haMaisAtributos BOOLEAN;
     statement       VARCHAR2(2000);
   BEGIN
       -- Para cada classe C
       FOR c IN SELF.classes.FIRST .. SELF.classes.LAST LOOP
          classeAtual := SELF.classes(c);

         -- Inicialize o conjunto de instancias
         DELETE FROM P;

         FOR i IN SELF.ConjTreinamento.FIRST ..
                  SELF.ConjTreinamento.LAST
         LOOP
            SELECT DEREF(SELF.ConjTreinamento(i))
                   INTO objeto
            FROM dual;

            IF objeto.classe = classeAtual THEN
                 INSERT INTO P
                 VALUES (objeto.aninhada,objeto.classe);
            END IF;
         END LOOP;

         SELECT /*+ CHOOSE */ COUNT(*) INTO numInstancia
         FROM THE (SELECT aninhada from P);

         WHILE (numInstancia != 0) LOOP
            -- Crie uma regra R com o lado esquerdo vazio
            SELF.NovaRegra(classeAtual);

            -- Até R estar perfeita
            haMaisAtributos := FALSE;
            LOOP
               TodosOsTermos(SELF.regras,
                             SELF.acuraciaAtribValor);

               -- Adicione A=v a R
               SELF.IncluiTermo(maior, regras,
                                acuraciaAtribValor);

               -- Selecione A e v para maximizar a acurácia
```

```
            maior :=
                    SELF.CalculaAcuraciaAtribValor(
                          SELF.acuraciaAtribValor);

            IF acuraciaAtribValor(maior).P=0 OR
               acuraciaAtribValor(maior).T=0 THEN
              Acuracia := 0;
            ELSE
              Acuracia := acuraciaAtribValor(maior).P/
                          acuraciaAtribValor(maior).T;
            END IF;

            IF HaMaisAtributos=FALSE OR acuracia=1  THEN
              EXIT;
            END IF;
          END LOOP; -- regra esta perfeita?

          SELF.ImprimeNovaRegra(regras);

          SELF.GravaNovaRegra(regras);

          RemoveInstancias(regras);

          SELECT /*+ CHOOSE */ COUNT(*) INTO numInstancia
          FROM THE (SELECT aninhada from P);
       END LOOP; -- while (numInst !=0)
    END LOOP; -- cada classe C
END;



MEMBER PROCEDURE TodosOsTermos(
                   regras IN OUT regrasVarray,
                   acuraciaAtribValor IN OUT condVarray)
IS
 query_str    VARCHAR2(2000);
 where_clause VARCHAR2(300);
 from_clause  VARCHAR2(300);
 contaT       NUMBER;
 aux          NUMBER;
 elemento     INTEGER;
 Limite       INTEGER;
BEGIN
  FOR i IN acuraciaAtribValor.FIRST ..
           acuraciaAtribValor.LAST LOOP
      acuraciaAtribValor(i).atributo := ' ';
      acuraciaAtribValor(i).valor := ' ';
      acuraciaAtribValor(i).P := 0;
      acuraciaAtribValor(i).T := 0;
  END LOOP;

  Elemento := 1;
```

```
-- Para cada atributo A e cada valor v,
 FOR a IN 0..(SELF.cardinalidadeDominios.LAST-1) LOOP
    IF (SELF.regras(2*a+1) !=
       SELF.cardinalidadeDominios(a+1).atributo) THEN
       limite := 0;
       FOR i IN 1..(a+1) LOOP
         limite := limite +
              SELF.cardinalidadeDominios(i).quantidade;
       END LOOP;

       -- Considere adicionar a condicao A=v a R
       FOR t IN (limite –
          SELF.cardinalidadeDominios(a+1).quantidade+1)
         ..limite
       LOOP
          acuraciaAtribValor(elemento).atributo:=
              SELF.cardinalidadeDominios(a+1).atributo;
          acuraciaAtribValor(elemento).valor   :=
              SELF.dominiosAtribConjTrein(t);
          acuraciaAtribValor(elemento).P := 0;
          acuraciaAtribValor(elemento).T := 0;

          from_clause := ' ';
          contaT := 0;
          where_clause := ' ';
          query_str := ' ';

        FOR rg IN regras.FIRST .. ((regras.LAST-2)/2)
       LOOP
            IF where_clause = ' ' THEN
              IF regras(2*rg) != ' ' THEN
                 Where_clause := where_clause || 'nt.'
                  || SELF.AtributosConjTreinamento(rg)
                  ||' = '||''' || regras(2*rg) ||''';
              END IF;

              IF acuraciaAtribValor(elemento).atributo
              = SELF.AtributosConjTreinamento(rg)  THEN
                 Where_clause := where_clause || 'nt.'
                 || SELF.AtributosConjTreinamento(rg)
                 || ' = ' ||''''||
                 acuraciaAtribValor(elemento).valor
                         ||''';
              END IF;
            ELSE
              IF regras(2*rg) != ' ' THEN
                 Where_clause := where_clause ||
                 '  AND nt.' ||
                 SELF.AtributosConjTreinamento(rg)
                 || ' = ' ||
                 ''' ||regras(2*rg) ||''';
              END IF;
```

```
                IF acuraciaAtribValor(elemento).atributo
                = SELF.AtributosConjTreinamento(rg)  THEN
                   Where_clause := where_clause ||
                      '  AND nt.' ||
                      SELF.AtributosConjTreinamento(rg)
                      || ' = ' || ''''||
                      acuraciaAtribValor(elemento).valor
                      ||'''';
                END IF;
              END IF;
            END LOOP;


            IF where_clause != ' ' THEN
              FOR c IN SELF.classes.FIRST ..
                       SELF.classes.LAST
             LOOP
                IF SELF.classes(c) !=
                   regras(SELF.regras.LAST) THEN
                   from_clause:='SELECT aninhada FROM e
                          WHERE classe  = '
                          ||''''|| SELF.classes(c) ||'''';
                   query_str:=' SELECT /*+ CHOOSE */ count(*)
                               FROM THE (' ||from_clause ||
                             ' ) nt WHERE ' || where_clause;
                   EXECUTE IMMEDIATE query_str INTO aux;
                   ContaT := contaT + aux;
                END IF;
              END LOOP; -- para cada classe

              query_str := ' SELECT /*+ CHOOSE */ COUNT(*)
                             FROM  THE (SELECT
                             aninhada FROM P) NT
                             WHERE ' || where_clause;
              EXECUTE IMMEDIATE query_str INTO
                         acuraciaAtribValor(elemento).P;
              acuraciaAtribValor(elemento).T :=
                  acuraciaAtribValor(elemento).P + ContaT;
            ELSE
              acuraciaAtribValor(elemento).P :=0;
              acuraciaAtribValor(elemento).T :=0;
            END IF;

            Elemento := elemento+1;

            END LOOP; -- for t

        END IF;  --regra jah contem este atributo.
      END LOOP; -- for a

    END TodosOsTermos;
```

```
  MEMBER PROCEDURE RemoveInstancias (
                   regras IN regrasVarray)
  IS
    query_str       VARCHAR2(2000);
    where_clause    VARCHAR2(300);
    atributos       atributosVarray;
  BEGIN
      where_clause := ' ';

      FOR  rg IN regras.FIRST .. ((regras.LAST-2)/2) LOOP
        IF where_clause = ' ' THEN
           IF regras(2*rg) != ' ' THEN
              Where_clause := where_clause  ||
                       SELF.AtributosConjTreinamento(rg)
                       || ' = ' || '''' ||
                       regras(2*rg) ||'''';
           END IF;
        ELSE
           IF regras(2*rg) != ' ' THEN
              Where_clause := where_clause || '  AND ' ||
                     SELF.AtributosConjTreinamento(rg)
                     || ' = ' || '''' ||
                     regras(2*rg) ||'''';
           END IF;
        END IF;
     END LOOP;

     query_str := 'DELETE TABLE(SELECT aninhada
                   FROM p WHERE classe = '||''''||
                   regras(regras.LAST) ||''''
                   || ' )  WHERE ' ||
                   Where_clause;
     EXECUTE IMMEDIATE query_str;

  END RemoveInstancias;

END;
```

Apêndice B

Regras Induzidas da Base de Dados do Campo Escola de Namorado

Regras geradas para o conjunto de treinamento formado pela união dos dados dos poços 3NA0001ARJS, 3NA0002ARJS, 3NA0004ARJS, 7NA0007RJS, 7NA0011RJS, 7NA0012RJS, 7NA0037DRJS, 4RJS0042RJ e 4RJS0234RJ. Os valores correspondentes a cada uma das faixas utilizidas na indução das regras, para cada atributo, foram as seguintes:

Atributo / Faixa	F1	F2	F3
DEPT.M (Profundidade)	2980,0 - 3016,6	3016.61 - 3039.8	3039.81 - 3051.8
DT (Reflexão)	80.6530 - 83.0884	83.08841 - 86.9844	86.98441 - 92.3751
GR (Radioatividade)	45.7542 - 49.3945	49.39451 - 53.6562	53.65621 - 57.875
ILD (Resistividade)	1.0452 - 2.0781	2.07811 - 3.21	3.211 - 7.7439
NPHI (Porosidade)	18.8721 - 20.4688	20.46881 - 22.6267	22.62671 - 24.1016
RHOB (Densidade)	2.1575 - 2.1978	2.19781 - 2.2776	2.27761 - 2.3931

Atributo / Faixa	F4	F5
DEPT.M (Profundidade)	3051.81 - 3065.0	3065.01 - 3080.0
DT (Reflexão)	92.37511 - 95.4141	95.41411 - 99.3562
GR (Radioatividade)	57.8751 - 65.3125	65.31251 - 72.2843
ILD (Resistividade)	7.7439 - 32.4375	32.4375 - 40.964
NPHI (Porosidade)	24.10161 - 26.8633	26.8633 - 28.3757
RHOB (Densidade)	2.39311 - 2.4958	2.4958 - 2.6042

```
SE dept= F4 E ild = F2 E gr = F2 ENTAO 01
SE rhob = F5 E dept= F3 ENTAO 01 SE dept= F4 E rhob = F5 E gr
= F1 ENTAO 01
SE dept= F4 E rhob = F5 E dt = F2 ENTAO 01 SE dept= F4 E ild =
F2  E gr = F3 ENTAO 01
SE dept= F4 E rhob = F5 E dt = F5 ENTAO 01
SE dept= F4 E rhob = F4 E nphi = F2 ENTAO 01
SE dept= F4 E gr = F1 E dt = F3 ENTAO 01
SE gr = F2 E ild = F3 E dt = F1 ENTAO 01
```

```
SE gr = F2 E ild = F3 E rhob = F1 ENTAO 01
SE dept= F4 E gr = F1 E nphi = F2 ENTAO 01
SE gr = F2 E ild = F4 E nphi = F3 ENTAO 01
SE gr = F2 E ild = F2 E dt = F1 ENTAO 01
SE ild = F3 E nphi = F2 E gr = F1 ENTAO 01
SE gr = F2 E ild = F4 E dt = F2 E dept= F4 ENTAO 01
SE rhob=F1 E gr=F5 E ild = F4 E dept= F4 E nphi = F5 ENTAO 01
SE gr = F2 E ild = F4 E nphi = F5 E dept= F2 ENTAO 01
SE gr = F2 E ild = F3 E dt = F4 E dept= F2 ENTAO 01
SE ild = F2 E rhob = F5 E dept= F5 ENTAO 01
SE gr = F2 E ild = F2 E dept= F2 E dt = F2 ENTAO 01
SE ild = F3 E gr = F3 E nphi = F5 ENTAO 01
SE ild = F3 E dt = F2 E dept= F3 ENTAO 01
SE rhob = F4 E dept= F4 E gr = F3 ENTAO 01
SE rhob = F1 E dt = F4 E dept= F1 E gr = F3 ENTAO 01
SE rhob = F4 E nphi = F3 E gr = F4 ENTAO 01
SE gr = F2 E ild = F3 E rhob = F4 E nphi = F2 ENTAO 01
SE gr = F2 E ild = F4 E dept= F2 E rhob = F1 ENTAO 01
SE ild = F3 E gr = F3 E dept= F4 ENTAO 01
SE ild = F3 E gr = F3 E dept= F2 E dt = F3 ENTAO 01
SE dept= F4 E rhob = F5 E dt = F4 ENTAO 01
SE nphi = F3 E rhob = F5 ENTAO 01
SE dept= F3 E nphi = F3 E rhob = F4 E ild = F2 ENTAO 01
SE dept= F4 E nphi= F5 E gr = F4 E ild = F4 E dt = F5 ENTAO 01
SE dept= F3 E ild = F3 E dt = F3 E gr = F5 ENTAO 01
SE ild = F5 E gr = F5 E nphi = F3 ENTAO 02
SE ild = F5 E gr = F5 E dept= F4 E nphi = F5 ENTAO 02
SE dt = F2 E dept= F5 E nphi = F1 E gr = F2 ENTAO 02
SE ild = F5 E gr = F5 E dept= F4 E dt = F4 ENTAO 02
SE dt = F3 E gr =F1 E ild = F1 E dept= F2 E nphi = F3 ENTAO 02
SE dt = F2 E dept= F5 E rhob = F5 E gr = F2 ENTAO 02
SE ild = F5 E dept= F4 E nphi = F5 ENTAO 02
SE gr = F5 E dt = F1 E dept= F1 ENTAO 03
SE gr = F5 E dt = F2 E dept= F1 ENTAO 03
SE gr = F5 E dept= F3 E nphi = F2 E ild = F4 ENTAO 03
```

```
SE rhob = F1 E dept= F4 E nphi = F4 E gr = F5 ENTAO 03
SE dt = F4 E dept= F4 E gr = F2 E ild = F4 ENTAO 03
SE gr = F5 E ild = F4 E nphi = F3 E dept= F3 ENTAO 03
SE dept= F1 E dt = F1 E gr = F2 E ild = F5 ENTAO 03
SE dept= F1 E ild = F5 E dt = F2 ENTAO 03
SE gr = F5 E dept= F4 E ild = F4 E dt = F4 ENTAO 03
SE dept= F1 E ild = F5 E gr = F4 E dt = F5 ENTAO 03
SE gr = F5 E dept= F2 E dt = F4 E nphi = F3 ENTAO 03
SE nphi = F4 E dept= F4 E gr = F3 E dt = F4 ENTAO 03
SE gr = F5 E ild = F1 E nphi = F4 E dept= F1 ENTAO 03
SE rhob = F4 E nphi = F1 E dept= F4 E dt = F2 ENTAO 03
SE ild = F4 E nphi = F3 E dept= F3 E dt = F4 ENTAO 03
SE gr = F3 E nphi = F1 E dept= F5 E dt = F1 ENTAO 03
SE ild = F4 E gr = F5 E nphi = F3 E dept= F1 ENTAO 03
SE nphi = F5 E dept=F4 E rhob= F2 E gr = F5 E dt = F5 ENTAO 03
SE gr = F3 E rhob = F1 E dt = F3 E dept= F1 ENTAO 03
SE nphi = F4 E dept= F4 E gr= F3 E dt = F5 E ild = F4 ENTAO 03
SE dept= F5 E ild = F2 E dt = F3 ENTAO 04
SE dept= F5 E nphi = F3 E ild = F3 E dt = F2 ENTAO 04
SE dept= F5 E nphi = F3 E ild = F2 E gr = F3 ENTAO 04
SE dept= F5 E nphi = F3 E ild= F3 E gr = F3 E dt = F3 ENTAO 04
SE nphi = F2 E ild = F5 E gr = F4 ENTAO 04
SE dept= F5 E nphi = F2 E gr = F4 ENTAO 04
SE gr = F3 E dept= F5 E dt = F4 E nphi = F3 ENTAO 04
SE dt = F3 E nphi = F1 E dept= F4 ENTAO 04
SE gr = F3 E ild = F3 E dept= F5 E rhob = F2 ENTAO 04
SE dt = F3 E dept= F5 E nphi = F2 E gr = F3 ENTAO 04
SE gr = F3 E dept= F5 E dt = F5 ENTAO 06
SE gr = F3 E dept= F5 E dt = F1 E ild = F1 ENTAO 06
SE dept= F5 E ild = F1 E gr = F2 ENTAO 06
SE gr = F3 E dept= F5 E nphi= F2 E ild = F1 E dt = F2 ENTAO 06
SE gr = F3 E dt = F4 E dept= F4 E ild = F5 ENTAO 06
SE dt = F3 E ild = F3 E dept= F4 ENTAO 06
SE dept= F5 E ild = F1 E nphi = F3 E dt = F3 ENTAO 06
SE dept= F5 E gr = F5 E dt = F2 ENTAO 06
```

```
SE dept= F4 E ild = F5 E rhob = F5 ENTAO 06
SE dept= F5 E gr = F5 E dt = F3 E ild = F3 ENTAO 06
SE dept= F4 E ild = F5 E rhob = F4 ENTAO 06
SE gr = F3 E dt = F4 E dept=F4 E nphi = F3 E ild = F4 ENTAO 06
SE gr = F3 E dept= F5 E dt = F4 E nphi = F4 ENTAO 06
SE dept= F4 E ild = F5 E gr = F2 E nphi = F1 ENTAO 06
SE nphi = F3 E dept= F5 E dt = F2 E ild = F1 ENTAO 06
SE dept= F4 E gr = F4 E ild = F3 ENTAO 06
SE dept= F4 E ild = F5 E nphi = F3 E gr = F4 ENTAO 06
SE gr = F3 E ild = F4 E dt = F2 ENTAO 06
SE dt = F3 E ild = F2 E gr = F2 E dept= F3 ENTAO 07
SE ild = F3 E gr = F3 E dept= F3 E nphi = F3 ENTAO 07
SE nphi = F5 E dept= F4 E ild = F3 ENTAO 07
SE ild = F5 E dept= F2 ENTAO 08
SE dt = F5 E gr = F3 E ild = F5 E dept= F1 ENTAO 08
SE nphi = F5 E dt = F1 ENTAO 08
SE nphi = F5 E dept= F5 E dt = F3 ENTAO 08
SE dt = F5 E dept= F5 E ild = F5 ENTAO 08
SE dt = F5 E dept= F3 E nphi = F5 E ild = F4 ENTAO 08
SE dt = F5 E gr = F2 E dept= F1 ENTAO 08
SE nphi = F5 E dt = F2 E dept= F5 ENTAO 08
SE dt = F5 E gr = F2 E dept= F4 E ild = F4 ENTAO 08
SE nphi = F5 E gr = F1 E ild = F3 ENTAO 08
SE nphi = F5 E ild = F2 E dept= F1 ENTAO 08
SE nphi = F5 E ild = F1 E dept= F4 ENTAO 08
SE nphi = F4 E gr = F1 E dept= F2 ENTAO 08
SE nphi = F4 E gr = F2 E dt = F3 E ild = F3 ENTAO 08
SE nphi = F4 E dept= F1 E gr = F4 E dt = F4 ENTAO 08
SE dt = F5 E dept= F1 E gr = F5 E ild = F3 E nphi = F5 E rhob
= F2 ENTAO 08
SE nphi = F4 E gr = F2 E dt = F3 E ild = F4 ENTAO 08
SE gr = F4 E dept= F4 E nphi = F4 E ild = F5 ENTAO 08
SE gr = F3 E ild = F4 E nphi = F3 E dept= F1 ENTAO 08
SE gr = F4 E dept= F4 E dt = F2 E nphi = F4 ENTAO 08
SE gr = F3 E dept= F4 E nphi = F5 ENTAO 08
```

SE gr = F3 E dept= F4 E dt = F3 E nphi = F3 ENTAO 08
SE gr = F4 E ild = F4 E dept= F1 E dt = F4 ENTAO 08
SE nphi = F5 E dept= F5 E gr = F3 ENTAO 08SE dt = F5 E ild = F1 E dept= F4 ENTAO 08
SE nphi = F4 E dept= F5 E dt = F3 E gr = F3 ENTAO 08
SE dt = F5 E ild = F1 E gr = F5 E nphi = F5 ENTAO 08
SE gr = F4 E dept= F4 E dt = F2 E nphi = F5 ENTAO 08
SE gr = F4 E ild = F3 E dept= F3 E dt = F4 ENTAO 08
SE ild = F4 E nphi = F3 E dt = F2 E dept= F4 ENTAO 08
SE dt = F5 E gr = F3 E nphi = F3 E dept= F4 ENTAO 08
SE ild = F4 E gr = F2 E dept= F3 ENTAO 08
SE dt = F5 E gr = F2 E rhob = F2 E ild = F4 ENTAO 08
SE dept= F4 E ild = F1 E dt = F3 ENTAO 08
SE dt = F4 E gr = F1 E ild = F2 ENTAO 08
SE dept= F4 E ild = F1 E nphi = F2 E dt = F4 ENTAO 08
SE ild = F4 E gr = F4 E dt = F4 E rhob = F2 E dept= F4 E nphi = F3 ENTAO 08
SE dt = F3 E nphi = F5 E dept= F4 ENTAO 08
SE dept= F1 E gr = F2 E dt = F3 E ild = F4 ENTAO 08
SE dept= F5 E ild = F5 E dt = F4 ENTAO 09
SE ild = F4 E nphi = F2 E rhob = F5 ENTAO 09
SE ild = F4 E nphi = F2 E dt = F3 E dept= F4 ENTAO 09
SE ild = F4 E gr = F2 E dept= F5 ENTAO 09
SE nphi = F4 E gr = F1 E dt = F5 ENTAO 09
SE gr = F4 E ild = F3 E dept= F1 E rhob = F1 ENTAO 10
SE gr = F4 E ild = F3 E dept=F1 E dt = F5 E nphi = F5 ENTAO 10
SE gr = F3 E ild = F4 E nphi = F4 E dept= F3 ENTAO 10
SE gr = F4 E dt = F1 E dept= F4 E ild = F5 ENTAO 10
SE gr = F4 E ild = F4 E dt = F3 E nphi = F5 ENTAO 10
SE dept= F4 E ild = F1 E nphi = F3 E dt = F4 ENTAO 10
SE dept= F1 E rhob = F1 E ild = F3 ENTAO 10
SE nphi = F4 E dept= F1 E ild = F4 E dt = F5 ENTAO 10
SE dept= F4 E ild = F5 E nphi = F2 E gr = F3 ENTAO 10
SE gr = F4 E ild = F4 E nphi = F4 E dept= F4 ENTAO 10
SE ild = F5 E nphi = F2 E dept= F4 E gr = F2 ENTAO 10

SE gr = F4 E ild = F4 E nphi = F4 E dept= F3 E dt = F5 E rhob = F2 ENTAO 10
SE rhob = F4 E dept= F3 E ild = F2 E gr = F4 ENTAO 11
SE rhob = F4 E dept= F3 E gr= F3 E ild = F2 E dt = F2 ENTAO 11
SE dt = F4 E dept= F5 E ild = F3 ENTAO 11
SE gr = F4 E dt = F1 E ild = F4 ENTAO 11
SE dept= F1 E dt = F3 E gr = F3 E ild = F4 ENTAO 11
SE dt = F4 E dept= F5 E ild = F2 E gr = F5 ENTAO 11
SE ild = F5 E dept= F1 E gr = F4 E dt = F4 ENTAO 11
SE rhob = F4 E dept= F5 E dt = F2 E gr = F3 ENTAO 11
SE gr = F5 E dt = F1 E dept= F5 ENTAO 12
SE gr = F5 E dept= F2 E ild = F2 ENTAO 12
SE gr = F5 E dept= F2 E nphi = F5 E ild = F3 ENTAO 12
SE gr = F5 E dept= F3 E dt = F5 ENTAO 12
SE gr = F5 E dept= F3 E dt = F4 E ild = F4 ENTAO 12
SE gr = F4 E ild = F4 E dt = F3 E nphi = F3 ENTAO 12
SE gr = F4 E ild = F4 E dt= F3 E nphi = F4 E dept= F3 ENTAO 12
SE gr = F5 E dept= F5 E ild = F4 E dt = F3 ENTAO 12
SE gr = F4 E ild = F4 E nphi = F3 E rhob = F1 ENTAO 12
SE nphi = F4 E dept= F3 E dt = F5 E ild = F3 ENTAO 12
SE nphi = F4 E gr = F5 E dept= F2 E dt = F4 ENTAO 12
SE gr = F4 E ild = F3 E nphi = F3 E dept= F3 ENTAO 12
SE gr = F4 E ild = F4 E nphi=F4 E dept=F3 E rhob = F1 ENTAO 12
SE gr = F5 E dept= F3 E dt = F4 E ild = F3 ENTAO 12
SE gr = F4 E ild = F3 E dept= F1 E dt = F4 ENTAO 12
SE nphi = F4 E dt = F5 E dept= F3 E gr = F4 E ild = F4 E rhob = F2 ENTAO 12
SE gr = F5 E nphi = F4 E dept= F1 E dt= F5 E ild = F3 ENTAO 12
E gr = F5 E dept= F2 E nphi = F4 E dt = F3 E ild = F4 ENTAO 12
SE dept= F3 E ild = F1 E dt = F4 ENTAO 13
SE dept= F3 E ild = F5 E nphi = F5 ENTAO 13
SE dt = F3 E dept= F3 E ild = F1 E nphi = F4 ENTAO 13
SE dt = F3 E dept= F3 E ild = F1 E nphi = F5 ENTAO 13
SE gr = F4 E dept= F2 E dt = F3 E nphi = F1 ENTAO 13
SE gr = F4 E rhob = F4 E nphi = F4 ENTAO 13

SE nphi = F3 E dept= F3 E ild= F1 E gr=F4 E rhob = F2 ENTAO 13
SE nphi = F3 E dept= F3 E ild = F2 E gr = F5 ENTAO 13SE ild =
F1 E nphi = F4 E rhob = F4 ENTAO 13
SE nphi = F3 E rhob=F4 E dept=F3 E gr = F3 E ild = F1 ENTAO 13
SE gr = F4 E dept= F5 E ild = F4 ENTAO 13
SE nphi = F3 E gr= F2 E ild = F1 E dt = F3 E dept= F2 ENTAO 13
SE dept= F3 E dt = F1 E ild = F4 ENTAO 13
SE dt = F2 E nphi = F4 E gr = F3 E dept= F2 ENTAO 13
SE gr = F5 E dept= F5 E ild = F4 E dt = F4 ENTAO 13
SE gr = F4 E dt = F2 E nphi = F3 E ild = F1 ENTAO 13
SE ild = F2 E nphi = F4 E gr = F3 ENTAO 13
SE gr = F4 E ild = F2 E dept= F4 ENTAO 13
SE dept= F3 E ild = F1 E nphi = F4 E dt = F2 E gr = F3 E rhob
= F2 ENTAO 13
SE dept= F2 E ild = F2 E dt = F4 E gr = F3 ENTAO 13
SE gr = F4 E dt = F1 E ild = F2 ENTAO 13
SE gr = F5 E dt = F2 E dept= F2 E nphi = F2 ENTAO 13
SE nphi = F5 E dt = F2 E gr = F3 E dept= F2 ENTAO 13
SE rhob = F1 E ild = F3 E dept= F5 ENTAO 13
SE dt = F3 E rhob = F1 E gr = F5 ENTAO 13
SE dept= F1 E nphi = F1 E dt = F5 ENTAO 15
SE gr = F5 E nphi = F4 E ild = F5 E dept= F1 ENTAO 15
SE dt = F5 E ild = F3 E dept= F4 E gr = F5 ENTAO 15
SE dept= F1 E dt = F1 E gr = F1 E ild = F5 ENTAO 15
SE gr = F4 E dt = F2 E dept= F1 E ild = F4 ENTAO 15
SE gr = F4 E ild=F3 E rhob= F1 E dept= F3 E nphi = F5 ENTAO 15
SE dept= F1 E dt = F1 E ild = F4 E gr = F2 ENTAO 15
SE dept= F1 E rhob = F4 E dt = F2 ENTAO 16
SE dept= F1 E ild = F3 E nphi = F3 ENTAO 16
SE dept= F1 E dt = F3 E ild= F4 E gr = F5 E nphi = F2 ENTAO 16
SE rhob = F4 E dept= F3 E gr = F2 E ild = F1 ENTAO 16
E dept= F1 E ild = F3 E dt = F2 ENTAO 16
SE gr = F5 E dept= F1 E ild= F3 E dt = F4 E rhob = F2 ENTAO 16
SE gr = F5 E ild = F2 E dt = F5 E nphi = F3 ENTAO 16
SE dt = F3 E dept= F1 E ild = F3 E nphi = F4 ENTAO 16

SE dt = F3 E dept= F3 E gr = F3 E nphi = F3 E rhob = F2 E ild = F1 ENTAO 16
SE rhob = F4 E dt = F4 ENTAO 17
SE rhob = F4 E dept= F2 E nphi = F3 ENTAO 17
SE rhob = F4 E dept= F2 E ild = F2 ENTAO 17
SE rhob = F4 E dept= F2 E gr = F3 E nphi = F2 ENTAO 17
SE dept= F2 E rhob = F5 E dt = F2 ENTAO 17
SE dept= F2 E nphi = F2 E gr = F3 ENTAO 17
SE dept= F2 E nphi = F2 E ild = F2 ENTAO 17
SE dept= F2 E nphi = F2 E dt = F4 ENTAO 17
SE dept= F2 E nphi = F2 E ild = F1 E gr = F2 ENTAO 17
SE dt = F2 E dept= F1 E ild = F2 ENTAO 17
SE dept= F2 E ild = F1 E dt = F4 ENTAO 17
SE dt = F2 E dept= F1 E gr = F1 ENTAO 17
SE dept= F2 E nphi = F3 E gr = F4 ENTAO 17
SE dept= F2 E nphi = F3 E gr = F3 E dt = F3 ENTAO 17
SE dept= F3 E nphi = F2 E dt = F3 ENTAO 17
SE dt = F2 E dept= F2 E gr = F1 E ild = F2 ENTAO 17
SE dept= F3 E nphi = F2 E gr = F1 E rhob = F2 ENTAO 17
SE rhob = F4 E dept= F3 E gr = F1 E dt = F2 ENTAO 17
SE dept= F2 E dt = F2 E gr = F5 E nphi = F3 ENTAO 17
SE dept= F3 E ild = F5 E nphi = F4 ENTAO 17
SE dept= F2 E gr = F1 E dt = F5 ENTAO 17
SE dt = F2 E dept= F3 E ild= F1 E rhob = F2 E gr = F2 ENTAO 17
SE dept= F2 E gr = F5 E nphi = F4 E dt = F5 ENTAO 17
SE ild = F2 E dept= F2 E dt = F3 ENTAO 17
SE rhob = F4 E dept= F1 E dt = F1 ENTAO 17
SE dt = F2 E dept= F2 E gr = F3 E ild = F2 ENTAO 17
SE gr = F5 E dept= F2 E nphi= F4 E dt = F3 E ild = F3 ENTAO 17
SE rhob = F4 E dept= F3 E gr = F2 E ild = F2 ENTAO 17
SE gr = F5 E ild = F4 E dept= F1 E dt = F4 ENTAO 17
SE dt = F2 E dept= F3 E rhob = F2 E gr = F1 ENTAO 17
SE dept= F2 E gr = F4 E ild = F3 ENTAO 17
SE ild = F1 E gr = F1 E dept= F1 ENTAO 17
SE dept= F2 E nphi = F2 E gr = F1 E dt = F2 ENTAO 17

SE dept= F2 E gr = F5 E nphi = F3 E dt = F5 ENTAO 17
SE ild = F1 E dept= F4 E gr = F5 ENTAO 17
E dept= F3 E gr = F2 E ild = F3 ENTAO 17
SE rhob = F4 E ild = F1 E dt= F2 E gr = F3 E dept= F3 ENTAO 17
SE dept= F2 E ild = F1 E dt = F5 ENTAO 17
SE dt = F2 E dept= F2 E ild = F1 E gr = F2 ENTAO 17
SE ild = F2 E nphi =F2 E rhob=F2 E dept= F3 E dt = F2 ENTAO 17
SE nphi = F4 E dt = F2 E ild = F3 ENTAO 17
SE nphi = F4 E ild = F3 E gr = F3 E dt = F4 ENTAO 17
SE dt = F3 E dept= F1 E nphi = F4 E gr = F4 ENTAO 17
SE dept= F2 E ild = F4 E gr = F5 E dt = F5 ENTAO 17
SE dt = F3 E ild = F3 E dept= F3 E nphi = F4 ENTAO 17
SE dept= F5 E nphi = F1 E gr = F1 ENTAO 18
SE nphi = F4 E ild = F2 E gr = F4 ENTAO 18
SE nphi = F4 E dt = F1 E dept= F3 ENTAO 18
SE ild = F1 E nphi=F4 E dept=F2 E gr = F4 E rhob = F2 ENTAO 18
E dept= F5 E rhob = F5 E gr = F3 E dt = F2 ENTAO 18
SE rhob = F4 E gr = F1 E dt = F3 ENTAO 18
SE nphi = F4 E dt = F2 E dept= F3 E ild = F1 E gr = F3 E rhob
= F2 ENTAO 18
E dept= F1 E ild = F1 E gr = F4 ENTAO 20
SE dept= F1 E ild = F1 E gr = F3 ENTAO 20
SE dept= F1 E ild = F1 E rhob = F4 ENTAO 20
SE dept= F1 E ild = F2 E dt = F4 ENTAO 20
SE dept= F1 E ild = F1 E dt = F4 E nphi = F3 ENTAO 20
E dept= F1 E dt = F3 E ild = F2 ENTAO 20
SE dept= F1 E dt = F3 E ild = F1 E gr = F5 ENTAO 20
E dept= F1 E dt = F3 E ild = F3 E nphi = F2 ENTAO 20
SE dept= F1 E ild = F1 E dt = F4 E nphi = F2 ENTAO 20
SE dept= F1 E ild = F1 E nphi = F3 E dt = F5 ENTAO 20
SE gr = F5 E rhob = F1 E dept= F1 E ild = F4 ENTAO 21
SE gr = F5 E ild = F5 E dept= F1 E dt = F5 ENTAO 21
SE gr = F5 E dt = F2 E dept= F4 ENTAO 21
SE gr = F5 E dept= F2 E dt = F3 E nphi = F3 ENTAO 21
SE ild = F5 E rhob = F5 E dt = F3 ENTAO 21

```
SE nphi = F4 E gr = F3 E dept= F4 E ild = F5 ENTAO 21
SE gr = F4 E rhob = F5 E dept= F5 ENTAO 21
SE nphi = F4 E dept= F1 E dt= F4 E gr = F3 E ild = F5 ENTAO 21
SE gr = F5 E ild = F1 E dept= F5 ENTAO 21
SE gr = F4 E dt = F2 E ild =F4 E nphi = F2 E dept= F4 ENTAO 21
SE dept= F1 E ild = F4 E gr = F3 E nphi = F5 ENTAO 22
SE dept= F1 E nphi = F4 E dt= F4 E gr = F3 E ild = F4 ENTAO 22
```

Apêndice C

Versão Java do PRISM

A versão ‘stand-alone’ em Java do PRISM, PrismJ, utiliza quatro classes: *prism*, *tabela*, *regras* e *condições*. A classe *prism* encapsula todo o processamento do algoritmo. A classe *tabela* é a responsável pela leitura dos dados do arquivo e armazenamento em uma estrutura tipo matriz. A classe *regras* auxilia na construção das regras. A classe *condições* auxilia na construção dos pares atributo-valor que formarão as condições de escolha de qual par será o escolhido.

```
public Vector prism(TabelaIF tabela) {
  String valoresClasse[] =
         tabela.getValoresDistintos(tabela.getColunaAlvo());
  String classe;
  int colunaAlvo = tabela.getColunaAlvo();
  Vector regras = new Vector();
  TabelaIF E;
     for(int i=0; i<valoresClasse.length; i++){
        classe = valoresClasse[i];
        E = tabela;
        while( contemInstancias(E, classe) ){
             Regra regra = new Regra(classe);
             while (!regra.isPerfect()){
                 Vector v = getAllConditions(E, regra);
                if (v == null)
                     break;
                  Condition c = getBestCondition(v);
                  rule.addCondition(c);
             }
             E = E.retiraLinhas(E.getLinhas(rule));
             regras.addElement(regra);
         }
     }//for(i)
     return regras;
}//prism()
```

FIGURA 25. Versão ‘stand-alone’ em Java do Prism

Esta implementação do algoritmo encontra-se não integrada a banco de dados, os dados são lidos de um arquivo em formato ASCII que informa os atributos e a seqüência de dados a serem minerados.

Um objeto do tipo tabela apresenta dois atributos: um ‘array’ de caracteres, que armazena o nome de todos os atributos de classificação; e a matriz de dados, que armazena os valores lidos do arquivo em um ‘array’ em forma de tabela (matriz).

Um objeto do tipo tabela é passado como parâmetro para o método principal da classe prism. Métodos auxiliares de construção de regras e das condições dos possíveis pares atributo-valor que compõem as condições das regras foram implementados nas classes regras e condições, respectivamente.

O código principal do PrismJ é mostrado na figura 25.