EXPLORANDO PARALELISMO EM PRIMITIVAS DE MINERAÇÃO DE DADOS PARA A TAREFA DE CLASSIFICAÇÃO

Eduardo Bezerra da Silva

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Geraldo Bonorino Xexéo, D.Sc.
(Presidente)

Prof. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Gerson Zaverucha, Ph.D.

Prof. Alex Alves Freitas, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL de 1999

SILVA, EDUARDO BEZERRA DA

Explorando Paralelismo em Primitivas de Mineração de Dados para a Tarefa de Classificação, [Rio de Janeiro] 1999

XIII, 114 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1999)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Primitivas para Mineração de Dados

I. COPPE/UFRJ II. Título (série)

Dedico este trabalho aos meus pais, e a todos os meus professores.

Agradecimentos

Em 1994, recebi um e-mail começando com a seguinte frase: "Eduardo, está bom, mass pode melhorar...". Na época, eu era aluno de Iniciação Científica e a mensagem se referia à primeira tarefa que meu orientador, o então candidato a doutor Geraldo Xexéo, havia me passado. De lá para cá, muita coisa mudou, aprendi ("melhorei") muito. Agradeço a você, Xexéo, pelos bons conselhos (nem sempre seguidos...) e por todo este tempo de aprendizado. Agradeço também ao Jano e à Marta, demais professores de linha de Bancos de Dados que estão sempre me ajudando.

Obrigado à Fernanda Baião, ao Rafael Figueira e à Denise Nielebock por lerem versões preliminares desta tese e por seus comentários e sugestões sempre pertinentes. Obrigado à Inês pelas dicas com o LaTex e pela lição de perseverança na solução de um problema. Obrigado ao Renato Mauro pela ajuda na implementação de algumas funcionalidades do GOA++ necessárias ao desenvolvimento deste trabalho, e pelas dicas com o *GNUPLOT*. Obrigado ao Flavio Tavares e ao Jorge Soares pela ajuda com o PVM. Obrigado à Cláudia, à Solange, à Suely, e à Lúcia, secretárias do Programa, por toda a competência e prestatividade. Obrigado à Patrícia Leal pelos muitos favores e pelo carinho com que trata a todos. Obrigado à CAPES pelo apoio financeiro.

Obrigado a todos os autores que disponibilizam seus trabalhos científicos na Internet. A maioria das referências aqui utilizadas, sem as quais este trabalho não teria sido possível, foram "lá" obtidas. Dentre estes autores, destaco os autores em cujos trabalhos me baseei fortemente para desenvolver esta tese, Alex Freitas e George John. Em especial, agradeço ao Alex: Em outubro de 1998, tive a oportunidade de fazer algumas perguntas a ele no MLKDD, um tutorial organizado por Gerson Zaverucha, realizado aqui no Rio de Janeiro. Aqueles poucos minutos de conversa influenciaram em muito minha decisão em mudar o foco principal desta tese.

Agradeço a todos os meus amigos e à minha família, em especial aos meus pais, aos quais devo tudo. Aproveito para me desculpar pelo tempo que passei em "hibernação", "enfurnado em meu quarto", na produção deste trabalho.

Resumo da Tese apresentada ao COPPE/UFRJ como requisito parcial para obtenção do grau de Mestre em Ciências (M.Sc.)

EXPLORANDO O PARALELISMO EM PRIMITIVAS DE MINERAÇÃO DE DADOS PARA A TAREFA DE CLASSIFICAÇÃO

Eduardo Bezerra da Silva

Abril/1999

Orientador: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Este trabalho apresenta melhoramentos no que concerne à interação entre aplicações de mineração de dados para classificação e o sistema de banco de dados, através da definição de primitivas para a tarefa de classificação. Estas primitivas fornecem meios para aumentar a integração entre algoritmos de classificação e o sistema bancos de dados e facilitar a adaptação dos algoritmos a grandes volumes de dados. Para fins de validação, estas primitivas são implementadas e testadas utilizando-se os sistemas GOA++ e PVM, e são feitas comparações em relação a primitivas similares preexistentes.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EXPLORING PARALELISM ON DATA MINING PRIMITIVES FOR THE CLASSIFICATION TASK

Eduardo Bezerra da Silva

April, 1999

Advisor: Geraldo Bonorino Xexéo

Department: Computing and Systems Engineering

This work presents enhancements to the interaction between data mining applications for classification and the database system, by defining primitives for the classification task. These primitives allow the improvement of the integration between classification algorithms and the database system, and the adaptation of these algorithms to large volumes of data. For validation purposes, these primitives are implemented and tested by using the GOA++ and PVM systems, and the results are compared to existing, similar primitives.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Declarações

Capítulo 1

Introdução

Espalhei meus sonhos aos seus pés. Caminhe devagar, pois você estará pisando neles.

W.B. Yeats (1865-1939) Poeta irlandês.

1.1 Escopo da Tese

Nos últimos anos, computadores têm sido utilizados para armazenar quantidades imensas de dados relativos a transações de um grande número de empresas e organizações. Operações de venda a varejo, telecomunicações, transações bancárias e de cartões de crédito são apenas alguns exemplos de áreas de intensa produção de dados em forma digital. O valor destes dados armazenados está tipicamente ligado à capacidade de se extrair informação de mais alto nível que se encontra subjacente a estes dados, ou seja, informação útil que sirva para dar suporte a decisões, e para exploração e melhor entendimento do fenômeno gerador dos dados. Podem existir padrões ou tendências úteis e interessantes que, se descobertos, podem ser utilizados, por exemplo, para otimizar um processo de negócio em uma empresa, ajudar no entendimento dos resultados de um experimento científico, ajudar médicos a entender os efeitos de um tratamento, para citar alguns casos.

Até a década passada, o método tradicional de transformação de dados em in-

formação de alto nível se fundamentava na análise e interpretação manual. Por exemplo, geólogos planetários analisavam imagens de planetas e asteróides captadas remotamente, cuidadosamente localizando e catalogando objetos estelares de interesse. Estes analistas se tornavam intimamente familiares com os dados e, com a ajuda de técnicas estatísticas, forneciam sumários e relatórios acerca dos dados explorados. Com o passar dos anos, este tratamento manual se tornou impraticável em muitos domínios de aplicação conforme o volume e dimensionalidade dos dados foi crescendo.

A área de pesquisa em Mineração de Dados (MD) estuda como estes padrões, se existirem, podem ser extraídos, pelo menos semi-automaticamente, a partir de uma coleção de dados. Esta área de pesquisa é muitas vezes chamada de Extração de Conhecimento em Banco de Dados[1] em virtude de se assumir que os dados estejam armazenados em *Sistemas de Gerenciamento de Bancos de Dados* (SGBDs).

As diversas técnicas utilizadas em Mineração de Dados, na verdade, já existiam em outras áreas de pesquisas como Aprendizado de Máquina, Estatística, Reconhecimento de Padrões, etc. O que motivou o surgimento dessa nova área de pesquisa foi justamente a necessidade de adaptação das técnicas de análise preexistentes para sua aplicação a grandes volumes de dados acumulados.

Em Mineração de Dados, definem-se várias fases (passos) nas quais os dados são transformados até que, na fase final, resultem em conhecimento útil. Todas estas transformações constituem um processo. Este processo envolve vários passos, abrangendo a definição do problema, a obtenção e manipulação dos dados, inferência de padrões, e interpretação/avaliação dos resultados.

Um dos passos deste processo, considerado por muitos autores o mais importante, é o de *extração de padrões* a partir dos dados transformados em passos anteriores do processo. Esta extração é feita através de algoritmos que utilizam técnicas computacionais multidisciplinares provenientes de áreas de pesquisa como Estatística e Aprendizado de Máquina.

Dentro do passo de *extração de padrões*, pode-se encontrar diversas tarefas, de-

[1]Tradução aqui utilizada para o termo *Knowledge Discovery in Databases.*

pendendo de como o problema de mineração de dados foi definido. Uma destas tarefas é a de *classificação*. Nesta tarefa, o objetivo é prever o valor de um determinado atributo em função dos demais atributos de uma relação. Por exemplo, baseado nos dados acerca de um cliente de uma instituição bancária (idade, sexo, renda familiar, escolaridade, etc.) pode-se identificar, com algum grau de certeza, se este cliente irá honrar ou não um empréstimo feito.

Esta tese se insere no contexto de pesquisa em Mineração de Dados desenvolvido pelo grupo de Bancos de Dados do COPPE/UFRJ, e seu escopo está associado à tarefa de classificação do passo de extração de padrões do processo de mineração de dados. A motivação para esta tese é dada na próxima Seção.

1.2 Motivação da Tese

A questão de eficiência na realização do processo de MD é realmente importante, visto que a quantidade de dados manipulada em uma tarefa típica de MD é bastante grande e a maioria dos algoritmos utilizados atualmente são provenientes de algoritmos das áreas de Estatística e de Aprendizado de Máquina, onde é comum a presuposição de que os dados a serem analisados podem ser armazenados em memória principal. Em relações armazenadas em SGBDs cuja quantidade de dados seja muito grande, esta abordagem se torna impraticável. O mesmo se pode dizer com relação a arquivos que não estejam armazenados em SGBDs, mas que não possam ser armazenados em memória principal para serem processados. Isto torna o estudo de modificações naqueles algoritmos, com o objetivo de torná-los eficientes quando manipulando grandes quantidades de dados, uma questão importante.

Como notado por Silberschatz et al. [99], na medida em que as técnicas utilizadas em MD trabalham sobre bancos de dados reais, a interface entre estes últimos e os algoritmos de MD se torna uma questão de grande importância, que necessita de pesquisa e desenvolvimento adicionais. Já que a comunidade de bancos de dados vem desenvolvendo métodos paralelos e eficientes para armazenar, manter e recuperar informações de grandes bancos de dados, uma tendência natural é a de que os

algoritmos de MD se utilizem destes métodos, ou de adaptações destes.

Mas, paradoxalmente, a maioria das atuais aplicações de MD têm um esquema de acoplamento com o SGBD um tanto quanto incipiente, como mostra a Figura 1.1 que, embora simplista, representa bem a situação atual destas aplicações.

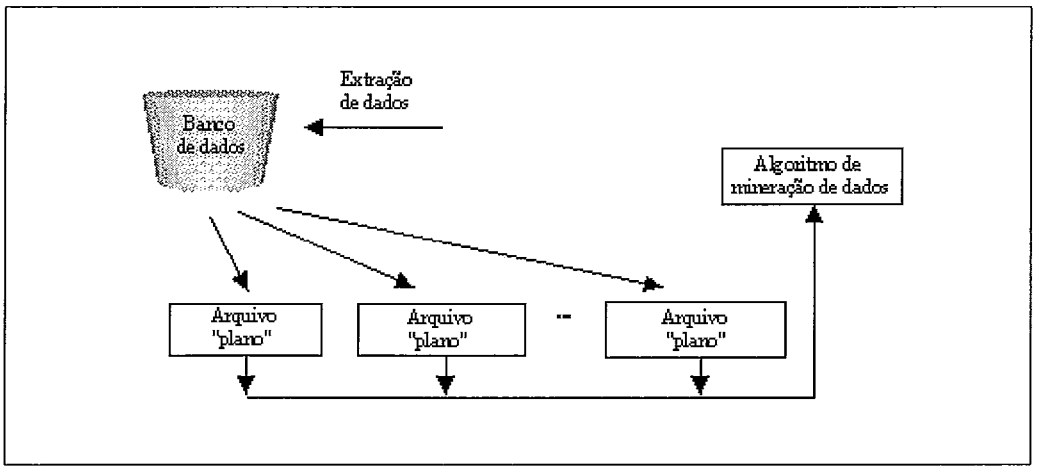

Figura 1.1: Abordagem usual para extração do conjunto de dados a ser utilizado por um algoritmo de mineração. Nesta abordagem, o banco de dados é um simples repositório.

Na Figura 1.1, os dados a serem analisados são obtidos a partir do envio de uma consulta na linguagem de manipulação de dados do SGBD (SQL, por exemplo). Estes dados constituem o *conjunto de dados alvo* do processo e correspondem aos dados relevantes à tarefa de MD a ser realizada. Esta coleção é então armazenada pela aplicação em arquivos planos (tabulares), no formato de entrada do algoritmo a ser utilizado. Algum tipo de processamento prévio pode ser realizado sobre estes dados. Após isso, é iniciada a extração de padrões.

Segundo Imielinski [61], esta abordagem de interação com o SGBD utilizada por muitas das aplicações de MD atualmente existentes poderia ser descrita como "mineração em arquivos", já que há um *acoplamento fraco* entre o sistema de MD e o SGBD no tocante ao processamento dos dados. No entanto, surgem algumas desvantagens provenientes desta abordagem, descritas a seguir:

1. As aplicações têm acesso ao SGBD de uma forma não uniformizada e improdutiva. Isto porque os algoritmos têm que ser construídos desde o início, cada

um tendo seu próprio método de acesso aos dados. Com o advento dos SGBDs, passou a existir uma camada de funcionalidades que proporciona uma forma de acesso padronizada aos dados e que pode ser utilizada para a construção destes algoritmos. Esta camada de funcionalidades constitui o conjunto de *primitivas* existentes no SGBD. Na abordagem acima, as aplicações não utilizam estas primitivas de acesso aos dados do SGBD. Como resultado, a produtividade e a uniformidade na construção destas aplicações tende a ser baixa.

2. Outro ponto fraco da abordagem acima diz respeito ao fato de que um algoritmo de MD pode ter um comportamento atípico de acesso aos dados que exigem operações especiais. Durante sua execução, pode ser necessário, por exemplo, o armazenamento de resultados parciais. Consequentemente, se tal algoritmo não tiver acesso a primitivas do SGBD que possam ajudar na implementação destas operações, funcionalidades semelhantes têm que ser implementadas no lado da aplicação. Isto, naturalmente, é indesejável dos pontos de vista de engenharia de *software* e de desempenho do algoritmo.

   No primeiro caso, porque as mesmas operações são implementadas em diversas aplicações, ao invés de estas aplicações utilizarem as primitivas do próprio SGBD. No segundo caso porque, como mencionado anteriormente, a maioria dos algoritmos utilizados em MD é proveniente de áreas de pesquisa que pressupõem que os dados a serem analisados estejam em memória principal, o que está se tornando cada vez mais impraticável. Em um SGBD, os algoritmos de manipulação de dados são preparados para operações em memória secundária. Em virtude disso, quanto maior for a parte do processamento dos dados que puder ser feita pelo SGBD antes do envio dos resultados à aplicação, mais eficiente tende a ser a execução da tarefa de MD.

3. Outra desvantagem que pode ser apontada é a de que pode ser simplesmente desnecessário transportar *todos* os dados de uma coleção para a aplicação, visto que a grande parte dos métodos de MD, notavelmente os métodos utilizados na tarefa de classificação, se baseiam em sumários estatísticos acerca daqueles

dados. Neste caso, a coleção completa é desnecessária do lado da aplicação, porque tudo que é necessário são resumos estatísticos associados aos dados da mesma. Sendo possível a utilização das próprias funcionalidades do SGBD para a obtenção destas informações estatísticas, o desempenho da aplicação em cuja tarefa de MD está sendo implementada tende a melhorar, em virtude de o tráfego de dados do SGBD para a aplicação diminuir.

1.3 Objetivos

Diante das desvantagens citadas na Seção anterior, é natural e necessário o estudo de abordagens diferentes para integração entre aplicações de MD e um SGBD. Uma abordagem que vem se mostrando uma tendência é a de se implementar as operações (primitivas) comuns a um grande número de algoritmos de MD utilizando funcionalidades do SGBD. Dentro desta abordagem, os trabalhos existentes na literatura implementam estas primitivas através da utilização da própria linguagem de consulta do SGBD, ou de uma extensão desta. Ou seja, cada primitiva é definida através de uma declaração na linguagem de manipulação de dados do SGBD (SQL, por exemplo).

O objetivo principal desta tese é a definição, implementação e validação de primitivas para dar suporte ao desenvolvimento de aplicações que realizem a tarefa de classificação em MD. Para alcançar este objetivo, é definida a seguinte estratégia:

1. Realizar um estudo sobre as primitivas já existentes na literatura, investigando suas características, vantagens e desvantagens;

2. Definir primitivas que possam, de alguma forma, generalizar ou melhorar as primitivas existentes;

3. Implementar as primitivas definidas no passo anterior, utilizando como sistema de bancos de dados o *GOA++* [30, 71, 73, 77]. Este sistema é um gerente de objetos, desenvolvido pela linha de Bancos de Dados do COPPE/UFRJ, compatível com o padrão ODMG (Object Database Management Group) [20],

possuindo as linguagens OQL (Object Query Language) e ODL (Object Definition Language) implementadas, além de uma interface para a linguagem de programação orientada a objetos C++. Para a simulação de um ambiente de processamento paralelo de consultas, utilizar o sistema PVM.

4. Validar as primitivas propostas, através de experimentos comparativos com primitivas já existentes na literatura.

1.4 Organização dos Capítulos

Os Capítulos desta tese estão organizados como segue. No próximo capítulo, é apresentada uma revisão dos principais conceitos e aspectos relacionados à área de Mineração de Dados.

No Capítulo 3, a tarefa de classificação é apresentada em detalhes. São descritos alguns dos métodos utilizados nesta tarefa, dentre estes, os métodos adaptáveis às primitivas definidas no Capítulo 4.

O Capítulo 4 apresenta a definição das primitivas para classificação. Esta definição é feita em alto nível, sem levar em conta os aspectos de implementação destas primitivas.

No Capítulo 5, os sistemas GOA++ e PVM são descritos em mais detalhes. São também apresentados os aspectos de implementação das primitivas. Para validação das mesmas, são apresentados resultados comparativos entre as primitivas definidas nesta tese e as de outros trabalhos.

O Capítulo 6 revisa alguns trabalhos de assuntos relacionados aos desta tese. Em cada tópico, são feitas comparações com o trabalho aqui apresentado.

Por fim, o Capítulo 7 apresenta as conclusões desta dissertação. São apresentadas observações quanto à importância e a aplicabilidade do trabalho e descritas algumas idéias para continuação futura do mesmo.

Capítulo 2

Mineração de Dados

The known is finite, the unknown infinite; intellectually we stand on an islet in the midst of an illimitable ocean of inexplicability. Our business in every generation is to reclaim a little more land.

T. H. Huxley (1825-1895) Biólogo inglês

2.1 Mineração de Dados: Processo ou Fase ?

No capítulo anterior foi mencionado que há um processo subjacente desde a definição até a resolução de um problema de mineração de dados, processo este constituído de várias fases ou passos. Este processo tem tido diferentes nomes em ambientes acadêmicos e comerciais. Parte da literatura (principalmente a do meio acadêmico [43]) se refere ao processo como um todo utilizando o termo *Extração de Conhecimento em Banco de Dados*, reservando o termo *Mineração de Dados* para nomear o passo deste processo que envolve a aplicação de algoritmos para extração de padrões aos dados. Já no ambiente comercial, o termo *Mineração de Dados* é mais utilizado para denotar o processo como um todo. Nesta dissertação, utiliza-se a segunda alternativa, onde o termo *Mineração de Dados* é utilizado para representar todo o processo. O passo deste processo onde são aplicados os algoritmos sobre os dados é aqui denominado *passo de extração de padrões.* Outra convenção aqui adotada é que

o termo *algoritmo de mineração de dados* é utilizado para denotar um determinado algoritmo de extração de padrões.

Muitas das técnicas utilizadas atualmente em MD já existem por muito tempo, tendo se originado em diversas áreas de pesquisa. Antes do termo *mineração de dados* se tornar popular, muitos pesquisadores em áreas como Estatística, Aprendizado de Máquina, Reconhecimento de Padrões, Redes Neurais e outras trabalhavam nos mesmos tipos de problemas sem uma integração entre uma e outra área. Adicionalmente, a comunidade de pesquisadores (com exceção da de redes neurais) não teve muito sucesso na tarefa de atrair o interesse comercial sobre suas respectivas tecnologias. Em função disto, até a década passada, o esforço na aplicação dos resultados de pesquisa obtidos nestas áreas a problemas reais nunca inspirou interesse em larga escala. O surgimento da pesquisa em Mineração de Dados reúne todas estas disciplinas sob a premissa de que bancos de dados guardam mais informação do que os dados neles armazenados, como, por exemplo, tendências de vendas em uma determinada região ou relacionamentos entre o sobe e desce de ações na bolsa e certos parâmetros monetários. Estas informações, que estão subjacentes aos dados armazenados, podem ser úteis no processo de tomada de decisões mas que, devido à enorme (e crescente) quantidade de dados envolvida, necessitam de adaptações às técnicas de análise de dados existentes para serem extraídas.

Este Capítulo faz uma revisão dos principais conceitos da área de Mineração de Dados. Os trabalhos que fornecem as melhores descrições sobre este processo são [41], [39] e [93]. A Seção 2.2 apresenta definições do processo de MD e descreve os seus passos. Na Seção 2.3, são discutidas as principais áreas correlatas à área de MD. A Seção 2.4 descreve as principais tarefas de mineração de dados existentes. Finalmente, a Seção 2.5 descreve os métodos de inferência indutiva e dedutiva e define o conceito de método de mineração de dados.

2.2 O Processo de Mineração de Dados

Esta Seção é dividida em duas partes. Na primeira, correspondente à Seção 2.2.1, são apresentadas duas definições do processo de MD e descritos alguns termos comumente utilizados nesta área. Na Seção 2.2.2, são apresentados os passos constituintes do processo de MD.

2.2.1 Definindo o Processo

Há, na literatura, diversas definições do termo mineração de dados. Considere a Definição 2.1 (adaptada de Frawley [39]). Uma descrição mais detalhada dos termos utilizados nesta definição é dada a seguir.

Definição 2.1 *O processo de Mineração de Dados pode ser definido como o processo não trivial de procura por padrões válidos, novos, potencialmente úteis e inteligíveis a partir de uma coleção de dados.*

- *Dados*: A coleção de dados utilizada pode ser *natural* ou *sintética*. Coleções de dados naturais são encontradas em bancos de dados, e são resultantes, por exemplo, de operações transacionais de uma determinada empresa ou de medições de fenômenos naturais. Já um conjunto de dados sintético, como o próprio nome indica, é gerado artificialmente. Neste tipo de conjunto de dados, os valores dos atributos são normalmente gerados aleatoriamente, mas seguindo alguma distribuição estatística.

  O termo *coleção de dados* utilizado na definição acima não faz menção à ordem de grandeza desta coleção. O que se pode dizer é que uma coleção de dados é grande o suficiente para justificar a aplicação de técnicas de mineração de dados se esta coleção é tão complexa que alguns de seus relacionamentos não podem ser extraídos manualmente ou por consultas convencionais (SQL, por exemplo) [17].

- *Padrão*: é uma expressão em uma determinada linguagem descrevendo fatos ou tendências associados a uma coleção dos dados, com algum grau de certeza.

Padrões devem ser mais simples que os dados propriamente ditos, implicando na necessidade de uma linguagem para se representar estes padrões.

Seja uma coleção de dados F, uma linguagem L, e uma medida de certeza C. Mais formalmente, um padrão é uma declaração S em L que descreve relacionamentos entre um subconjunto F_s de F com grau c de certeza [39]. Um padrão interessante (de acordo com alguma medida de interesse) é chamado *conhecimento* [102, 100].

Um exemplo de linguagem ou formalismo utilizado para a representação de padrões é uma linguagem de equações. Em uma linguagem de equações, operadores tais como "mais" e "vezes" podem ser utilizados para relacionar variáveis (por exemplo, $(a*X)+b$). Outro exemplo de uma linguagem de representação são regras, onde os operadores chave são condicionais (por exemplo, *If* C_1 *Then* C_2, onde C_1 e C_2 são condições envolvendo determinados atributos da coleção de dados). Algumas vezes, uma linguagem para representação de padrões está associada a uma tarefa de MD (Seção 2.4) em particular. (Por exemplo, uma linguagem de equações é tipicamente utilizada na tarefa de Regressão.) Outras vezes, uma mesma linguagem pode servir para representar os padrões gerados em diversas tarefas.

Podem-se classificar os padrões extraídos no processo de mineração de dados em dois tipos básicos [66]: padrões preditivos e padrões descritivos (ou informativos)[1].

Padrões preditivos são construídos com o intuito de se resolver um problema específico de predição dos valores de um ou mais atributos, em função dos valores de outros atributos. Já nos padrões descritivos, o ponto central está em se apresentar informações interessantes que um especialista do domínio da aplicação possa ainda não conhecer.

[1]Na terminologia utilizada na comunidade de aprendizado de máquina, a distinção é feita através da separação dos algoritmos em dois tipos, os de aprendizado *supervisionado* e os de aprendizado *não-supervisionado*, correspondendo aos algoritmos que geram padrões preditivos e descritivos, respectivamente [75].

O termo *modelo* também é utilizado na literatura para fazer referência a um padrão ou conjunto de padrões. Diz-se que se está *aprendendo*, *construindo*, ou *induzindo* um modelo a partir de uma coleção de dados quando para denotar o processo de procura por um conjunto de padrões ou modelo subjacente a esta coleção. Quando se faz uma estimativa dos valores para atributos de itens da coleção cujos valores não são conhecidos, diz-se que o modelo está sendo *aplicado* a esta coleção.

- *Processo*: o processo de Mineração de Dados é constituído de vários subprocessos (passos) que envolvem a preparação dos dados, a procura de padrões, avaliação dos padrões descobertos, entre outros. O processo é tido como *não trivial* porque tem algum grau de autonomia (semi-automático) no processamento e avaliação dos resultados.

  No processo de Mineração de Dados, geralmente há dois agentes humanos envolvidos [11, 66]: o *analista de mineração de dados* e o *especialista de domínio*. Usualmente, o especialista de domínio tem o entendimento claro do problema a ser resolvido enquanto que o analista de mineração de dados compreende o processo de mineração de dados e suas técnicas. [2]

- *Validade*: raramente um padrão descoberto por um algoritmo de mineração de dados é válido para todos os dados. Isto porque, na maioria das vezes, este algoritmo é aplicado em uma amostra retirada de uma coleção maior de dados. A representação do grau de certeza com o qual os padrões descrevem uma coleção de dados é essencial para se determinar o quanto um sistema ou usuário pode confiar neste padrões e tomar decisões a partir deles.

  Uma medida de certeza envolve vários fatores, incluindo a integridade dos dados, o tamanho da amostra utilizada no processo, e, possivelmente, a existência de *conhecimento de domínio* disponível. Sem um grau de certeza suficiente, os padrões descobertos não podem ser considerados conhecimento. A validade

[2] O analista de mineração de dados é chamado nas próximas Seções de *analista de dados* ou *analista* simplesmente, onde não houver ambigüidade.

de padrões preditivos e descritivos é medida diferentemente, como segue:

  - Padrões preditivos: podem ser avaliados pelo julgamento de quão efetivos eles são na predição. Visto que eles predizem o valor de um atributo, e o valor deste atributo existe no conjunto de dados utilizado para teste, o método para avaliar padrões preditivos é comparar suas predições com os valores reais no conjunto de teste.
  - Padrões descritivos: são mais difíceis de se avaliar que padrões preditivos, não apenas devido a sua natureza subjetiva, mas também em virtude de seu valor verdadeiro residir no fato de ele sugerir ações úteis ao especialista de domínio, e quão efetivas são aquelas ações. Neste caso, critérios matemáticos são utilizados para capturar os padrões mais interessantes. Por exemplo, na tarefa de Regras de Associação (Seção 2.4.4), há os parâmetros de precisão e confiança. Outro exemplo de tarefa de MD que gera padrões descritivos é a de Agrupamento (Seção 2.4.1).

- *Novo*: a questão de se um padrão descoberto é novo depende do ponto de vista do qual se está analisando, que pode ser ou o escopo do sistema ou o escopo do usuário. Para o sistema, um determinado padrão descoberto pode ser novo, mas para o usuário este padrão pode ser uma tautologia e não representar conhecimento. Por exemplo, um padrão relacionando a compra de pão à compra de leite (isto é, os produtos pão e leite são sempre comprados em conjunto, com um determinado grau de certeza) pode ser um padrão novo para o sistema, mas provavelmente não o é para o usuário. Os padrões descobertos devem ser novos, pelo menos para o sistema e preferencialmente para o usuário [42].

- *Potencialmente úteis*: os padrões descobertos são úteis somente se eles ajudarem a alcançar o objetivo do sistema ou do usuário. Por exemplo, um padrão relacionando a idade de um motorista à probabilidade de ocorrência de acidente, descoberto enquanto se tentava encontrar relacionamentos entre a quantidade de vendas e modelos de carros poderia não ser considerado útil.

A decisão de se os padrões extraídos são úteis é feita no passo de interpretação e avaliação (Seção 2.2.2) do processo de MD, atividade na qual o julgamento humano é usualmente requisitado. Para uma discussão mais detalhada da questão do nível de envolvimento do usuário com o processo de MD, vide [42] e [101].

- *Inteligíveis*: um dos objetivos do processo de MD é produzir conhecimento que possa ser compreendido facilmente, de forma a tornar mais fácil o entendimento dos dados que originaram este conhecimento. Isto implica em que a linguagem para representação dos padrões seja simples ou que estes padrões possam ser transformados de forma a se tornarem inteligíveis. Uma técnica bastante utilizada para alcançar este objetivo é apresentar os padrões em forma gráfica que facilite seu entendimento.

A Definição 2.1 enfoca as características do conhecimento resultante, ou seja, ressalta que os padrões extraídos devem ser válidos, novos, úteis, interessantes e inteligíveis. No entanto, em [11], Brachman e Anand advogam que, acima de tudo, Mineração de Dados é um *processo*, e limitar a atenção aos resultados pode levar à desconsideração da complexidade da extração, organização e apresentação do conhecimento extraído. Além disso, este processo é muito mais complexo que a "descoberta de padrões interessantes". Ele envolve, entre outras coisas, a negociação com os donos dos dados que se quer analisar [25], a estreita interação com os dados, manipulação de dados incompletos e incertos. Ainda em [11], é dada a seguinte definição, mais centrada na natureza de processo da mineração de dados:

Definição 2.2 *O processo de Mineração de Dados consiste de uma seqüência de interações complexas, que se estende sobre um determinado período de tempo, entre um "usuário" e uma coleção de dados, possivelmente auxiliado por um conjunto heterogêneo de ferramentas.*

Esta definição leva em conta o fato de o processo de MD ser interativo em todos os seus passos. Em outras palavras, o analista de dados (identificado como "usuário"

na definição 2.2) está sempre presente e intimamente envolvido com todos os passos do processo.

O termo "conjunto heterogêneo de ferramentas" da definição corresponde ao *sistema de mineração de dados* utilizado pelo analista. Dos sistemas de MD existentes, poucos são totalmente autônomos, e aqueles que o são têm aplicabilidade limitada [72]. Na maioria dos sistemas, a interferência e/ou influência humana é necessária em algum grau. Por esta razão é algumas vezes conveniente visualizar o usuário como parte (um dos componentes) do sistema. Um dos objetivos da pesquisa em Mineração de Dados é reduzir a quantidade de intervenção humana necessária no processo [41].

Ainda em [11], Brachman e Anand advogam que a interação do analista com os dados leva à formação de hipóteses sobre os mesmos. Neste sentido, o processo de MD tem mais semelhança com a atividade de um arqueólogo do que com a de um minerador: o "arqueólogo de dados" visualiza os dados como um todo e decide onde explorar baseado no que ele vê, em sua própria experiência, e no conhecimento fornecido pelo especialista de domínio. Uma vez feito isso, o analista reúne fragmentos que parecem se encaixar e decide o que vale a pena ser mais explorado e o que deve ser ignorado.

2.2.2 Visão Geral do Processo

Durante o processo de MD, diversas são as atividades com as quais um analista de dados está envolvido [11]. No nível mais alto, o analista de dados, em resposta a um objetivo[3] definido em conjunto com o especialista de domínio, consulta o banco de dados para extrair informações relevantes à realização deste objetivo. É feita, então, a análise dos dados utilizando-se ferramentas de análise e/ou visualização. Estas ferramentas de análise e visualização constituem os componentes do sistema MD utilizado. Conforme o analista de mineração de dados vai interagindo com os dados através do sistema de MD, ele vai ganhando algum nível de entendimento

[3]Este objetivo, mais comumente chamado na literatura de *tarefa de mineração de dados*, é mais detalhado na Seção 2.4.

(*insight*) acerca destes. Este entendimento permite que ele realize a construção de um modelo a partir destes dados. O analista então apresenta os resultados retirados deste modelo ao especialista de domínio que originou o objetivo inicial.

Esta atividade do analista de dados, auxiliada por um sistema MD, está relacionada a vários passos do processo de MD [42]. O processo de MD é *iterativo*, começando com o entendimento e definição do problema e terminando com a análise dos resultados. É importante notar que os passos do processo de MD se complementam, ou seja, o analista de dados pode ir e vir entre estas fases repetidamente. Este ciclo, tanto iterativo quanto interativo, é intrínseco ao processo de mineração de dados [41, 107].

Note ainda que, em sistemas de MD que possuem ferramentas de análise baseadas em visualização, a tarefa de mineração é especificada através de uma metáfora de visualização (por exemplo, um tipo específico de grafo) e da seleção dos elementos de dados a serem fornecidos à metáfora. A visualização produzida é, por si só, um modelo, e o analista de dados pode examinar a visualização para determinar seu poder exploratório. Visualização é uma funcionalidade importante para todas as fases do processo. A exibição apropriada de alguns dados e de seus relacionamentos pode dar ao analista de dados uma visão que é virtualmente impossível de se obter através da visualização de dados tabulares ou simples sumários estatísticos. De fato, para algumas tarefas, a visualização apropriada é o único item necessário para a resolução do problema. Facilidades como grafos e gráficos de barras, por exemplo, são encontradas em muitos sistemas de MD comerciais [11]. Em [67], pode-se encontrar uma descrição de diversas técnicas de visualização para mineração de dados.

Outro fator que pode influir nos resultados de execução dos passos do processo é a utilização de *conhecimento de domínio*. Alguns exemplos de conhecimento de domínio incluem [72]: definição de valores *default* a serem atribuídos a campos com valores inválidos ou de valor desconhecido; definições de novos campos (por exemplo, *Idade* = *AnoCorrente* - *AnoDeNascimento*); hierarquias de generalização (C é-um B; B é-um A); modelos funcionais ou causais, etc. Este tipo de informação pode reduzir de forma significante o tempo de execução de vários passos do processo [41].

Contudo, a extração, codificação e incorporação de conhecimento de domínio no processo de MD é um problema em aberto.

Cada um dos passos que constituem o processo de mineração de dados é descrito nos próximos parágrafos. É importante notar que esta divisão é apenas conceitual, e que, na prática, alguns dos passos do processo podem ser realizados simultaneamente.

1. **Entendimento e definição do problema**

   O processo de MD começa quando o especialista de domínio descreve o problema para o analista de dados e este, por sua vez, explica o processo de MD ao especialista. Dicionários de dados são utilizados pelo analista para que este obtenha informação acerca do conteúdo da coleção de dados a ser analisada, como nomes de campos, tipos, etc. Informações adicionais acerca da estrutura da coleção e de relacionamentos entre seus objetos, que podem servir de ajuda no passo de extração de padrões, podem existir fora deste dicionário de dados, em manuais, especificações, ou no cabedal de conhecimentos do especialista de domínio.

   O termo "definição do problema"significa que o analista de dados deve trabalhar em conjunto com o especialista de domínio para tornar a resolução do problema alcançável e mensurável. Como declarado pelo especialista de domínio, o problema pode parecer algo que poderia ser resolvido de diferentes maneiras. Algumas soluções, no entanto, podem ser completamente infactíveis. Cabe ao analista de dados a escolha da solução mais adequada ao problema [66].

   Este primeiro passo do processo pode despender uma quantidade significante de tempo. O processo de MD difere de outros processos analíticos no sentido de que é enganosamente fácil aplicar um algoritmo de extração de padrões a uma coleção de dados e se obter algum resultado. No entanto, sem um claro entendimento do problema, os resultados podem não ser compensatórios ou mesmo enganadores.

2. **Criação da "relação mina"**

   Inclui a seleção de um conjunto de dados sobre o qual a extração de padrões será realizada. Este conjunto de dados é normalmente chamada de *relação mina*[4] (Mine Relation). Nas aplicações de MD, assume-se que estes dados devem ser obtidos a partir de um banco de dados ou de um *depósito de dados* (vide Seção 2.3.4).

   Este passo é um exercício intelectual tanto da parte do analista de dados quanto da parte do especialista de domínio e tem o objetivo de definir que atributos são relevantes ao problema de mineração de dados a ser resolvido. Em função disto, este passo é bastante importante na medida em que o conjunto de dados escolhido como alvo vai refletir diretamente na qualidade dos padrões extraídos a partir da coleção.

3. **Pré-processamento dos dados**

   Pode-se dizer que este passo do processo de MD é constituído de subpassos a saber: limpeza dos dados (*data cleaning*) e transformação dos dados. Este passo inclui operações para retirada de eventuais inconsistências nos dados, tratamento de campos não disponíveis (ou nulos), mapeamento de valores inválidos em alguns atributos, discretização de atributos, e possivelmente uma seleção de atributos adicional à seleçào feita para criação da relação mina. Em virtude de um banco de dados ser projetado por motivos outros que o de mineração de dados (vide Seção 2.3.3) este é um dos passos mais complexos do processo de mineração de dados.

4. **Extração de padrões**

   Inclui a seleção de um ou mais métodos a serem aplicados na procura por padrões no conjunto de dados, assim como a decisão de que modelos e parâmetros podem ser apropriados. Após isto, o algoritmo de mineração de dados é executado e os padrões extraídos são representados em uma forma particular.

[4]Este termo tem vários sinônimos na literatura: *conjunto de treinamento, relação universal* e *conjunto de dados alvo.*

Este passo pode ser considerado o de mais alto grau de automação de todo o processo e o que tem tido mais atenção por parte da literatura em MD.

5. **Interpretação e avaliação dos resultados**

   Inclui a interpretação dos padrões extraídos e um possível retorno a alguns dos passos anteriores do processo. Além disto, neste passo faz-se a visualização dos padrões extraídos para a remoção de padrões redundantes ou irrelevantes e a representação dos padrões restantes em uma forma inteligível ao usuário. Esta representação pode variar desde uma descrição textual de uma tendência até um grafo elaborado capturando os relacionamentos no modelo. Pode ser também desejável que haja descrições de ações a serem tomadas como saída. Isto pode servir de guia para o especialista de domínio sobre que decisão tomar em função do resultado.[5]

2.3 Áreas de Pesquisa Relacionadas

Mineração de Dados é uma área de pesquisa interdisciplinar que combina métodos e técnicas de diversas áreas, dentre elas, pode-se citar: Aprendizado de Máquina, Estatística, Bancos de Dados é a de *Data Warehousing*. Nesta Seção, procura-se descrever os pontos de interseção (e de diferença) entre Mineração de Dados e suas áreas correlatas.

2.3.1 Aprendizado de Máquina

Um dos passos do processo de MD, o de extração de padrões, se utiliza de métodos de Aprendizado de Máquina (AM) para encontrar regularidades, padrões ou conceitos em coleções de dados. Técnicas desenvolvidas em AM, como a Indução de Regras e de Árvores de Decisão e o Aprendizado Baseado em Instâncias, formam o núcleo

[5]De fato, a solução ideal seria ações sendo tomadas diretamente pelo sistema a partir dos resultados. Em [11], encontra-se o conceito de "monitores" que têm a função de disparar um alarme ou ação no sistema quando determinados tipos de padrões são extraídos (detectados).

dos métodos utilizados em MD. No entanto, há algumas notáveis diferenças entre estas duas áreas de pesquisa [70].

A primeira delas é que grande parte da literatura em AM se concentra apenas no mecanismo de descoberta de padrões e/ou conceitos, sem se preocupar com o grau de utilidade dos mesmos. Em MD, os padrões extraídos são avaliados para se aferir o quão úteis eles são para o usuário.

Outra diferença diz respeito à complexidade do conhecimento obtido. Sistemas de MD geralmente têm aspirações bem mais modestas em termos do conhecimento obtido. Enquanto pesquisas em AM estudam formas de automatização de tarefas *inteligentes* que seriam difíceis de serem realizadas por seres humanos, a maioria dos estudos em MD está direcionada para a extração de conhecimento que um analista de dados competente estaria apto a encontrar, não levando em conta o tempo necessário para a realização de tal tarefa.

Outra questão freqüentemente citada na literatura diz respeito à quantidade de dados: tradicionalmente, pesquisas em AM têm se concentrado em coleções contendo centenas ou milhares de itens, e seus algoritmos pressupõem que os dados a serem utilizados podem estar em memória principal durante sua execução.[6] Em MD considera-se volumes de dados de ordem de grandeza maiores e, consequentemente, os algoritmos utilizados nesta área e provenientes da área de AM muitas vezes devem ser modificados para manipular esta quantidade maior de dados.

Por outro lado, a aplicação de métodos de MD pode ser útil mesmo em coleções pequenas. Em verdade, a fonte essencial de complexidade em um problema de mineração de dados pode não estar na quantidade de itens da coleção de dados, mas sim no número de atributos por item: a quantidade de padrões possíveis é uma função exponencial da quantidade de atributos [17].

[6] É bom notar, no entanto, que pode-se encontrar algoritmos de AM que funcionam bem em grandes quantidades de dados [75]. Um exemplo é o algoritmo para classificação SPRINT [89].

2.3.2 Estatística

A área de Estatística, junto com a área de Aprendizado de Máquina, são consideradas os "ancestrais" da área de pesquisa de mineração de dados [50]. Técnicas de reconhecimento de padrões e de análise exploratória de dados provenientes desta área são muito utilizadas em algoritmos de mineração de dados. Seleção de dados e amostragem, pré-processamento, transformação dos dados e avaliação de padrões extraídos são apenas alguns exemplos de métodos utilizados em Estatística por um longo tempo e que são utilizados em diversos passos do processo de mineração de dados.

O conhecimento que se infere a partir dos dados tem um componente estatístico fundamental, que é o grau de certeza com o qual se espera que este conhecimento descreva ou faça predições sobre os dados. A área de Estatística fornece uma linguagem para quantificação da incerteza resultante quando se tenta inferir padrões a partir de uma amostra de uma coleção de dados [59, 63].

Um fato importante a notar é que o termo mineração de dados teve uma conotação negativa durante algum tempo na área de Estatística, sendo associado ao termo *dragagem de dados* [70, 34]. Esta conotação surgiu do fato de que, no passo de extração de padrões do processo de MD, pode-se encontrar padrões que parecem ser estatisticamente relevantes, mas que de fato não o são se os dados não forem corretos ou relevantes. Esta questão é de fundamental importância para sistemas de mineração de dados. Na verdade, o passo de extração de padrões é uma atividade legítima, desde que ela esteja inserida no processo de mineração de dados como um todo [50]: (1) os padrões resultantes do passo de extração servindo de entrada para o passo de interpretação e avaliação dos mesmos e (2) os dados utilizados neste passo devem ter passado por todos os passos anteriores.

2.3.3 Bancos de Dados

Como enfatizado na Seção 2.1, as diversas técnicas utilizadas em mineração de dados já existiam há bastante tempo, antes de serem reunidas sobre este novo nome. Diante

deste fato, a seguinte pergunta surge naturalmente: por que uma nova área de pesquisa emergiu sobre o nome de *mineração de dados* ?

A pergunta acima pode ser respondida considerando-se o termo sinônimo do termo mineração de dados, *extração de conhecimento em bancos de dados*. A maioria das abordagens, técnicas e soluções utilizadas, por exemplo, em Estatística e Aprendizado de Máquina, não são adequadas para a manipulação de grandes quantidades de dados. A parte "bancos de dados" no termo acima refere-se justamente ao fato de que a área de MD considera o problema de se extrair padrões em bancos de dados *reais*, ao contrário da área de AM, por exemplo, onde se considera a hipótese de os dados estarem todos em memória principal. Em bancos de dados reais, esta hipótese não pode ser considerada.

Um banco de dados é uma coleção integrada de dados, organizada de tal forma a facilitar o armazenamento eficiente, assim como sua modificação e recuperação [28]. Normalmente, informações a respeito de cada nome de cada campo e seu domínio (meta-dados) são também armazenadas. Um *Sistema de Gerenciamento de Bancos de Dados* (SGBD) é uma coleção de procedimentos para recuperação, armazenamento e manipulação de bancos de dados. SGBDs, construídos especialmente para o gerenciamento de dados, podem fazer o papel de servidores de dados para aplicações envolvendo mineração de dados. Este fato abre diversas questões a serem tratadas pela comunidade de pesquisa em MD. Dentre elas, pode-se citar as seguintes:

- *Integrar técnicas de extração de padrões e SGBDs.* Um dos grandes problemas da área de MD é adaptar técnicas de extração de padrões provenientes de AM, Estatística e outras áreas em bancos de dados reais. Pesquisas com o objetivo de aumentar o grau de integração entre SGBDs e sistemas de mineração de dados, assunto principal desta tese, a ser tratado no Capítulo 4, estão no seu início. No Capítulo 6 são descritos alguns trabalhos na literatura que abordam este problema.

- *Bancos de dados são projetados com objetivos diferentes do de se fazer mineração de dados*: Isto é, a representação dos objetos do mundo real em um

banco de dados é escolhida de tal forma a atender às necessidades de aplicações que queiram obter acesso a estes dados e não para atender às necessidades de um sistema de MD. Em virtude disso, propriedades ou atributos que poderiam simplificar a tarefa de aprendizado (informações assumidas por alguns algoritmos de aprendizado de máquina) não estão necessariamente presentes.

- *Qualidade da informação armazenada no banco de dados*: Embora as partes teórica e prática de pesquisa em bancos de dados tenham evoluído bastante nas últimas duas décadas [104], a qualidade dos dados armazenados depende da qualidade da entrada destes dados no SGBD. Estes dados estão invariavelmente "contaminados" por erros, inconsistências, etc. Em função de como os dados foram coletados, campos não preenchidos em alguns itens serão encontrados, erros na entrada dos dados podem ocorrer, etc. Consequentemente, sistemas de mineração de dados devem ser *robustos* com relação a erros nos dados.

- *Gerenciamento de grandes dimensionalidades*: Bancos de dados de muitos gigabytes com milhões de itens e milhares de atributos por item estão se tornando mais e mais comuns (por exemplo, em ciências astronômicas e físicas [35, 36]). Estes bancos de dados armazenam grandes quantidades de informação que dificultam a indução de modelos por algoritmos convencionais, provenientes da área de AM, por exemplo. Há diversas soluções propostas na literatura para este problema. Uma delas é a utilização de técnicas de amostragem para diminuir a quantidade de dados processada [68, 79]. Outra solução é a utilização de processamento paralelo. Os trabalhos de Freitas em [38, 44] e de Sousa em [97, 96] são bons exemplos deste tipo de abordagem. Outras soluções adotadas são a seleção de atributos [18], e incorporação de conhecimento de domínio durante a realização dos passos do processo de MD [81].

- *Gerenciamento de mudanças nos dados*: Dados não-estacionários (que mudam com freqüência) podem invalidar padrões previamente descobertos. Soluções possíveis podem ser a criação de métodos incrementais para atualizar os pa-

drões e o tratamento de mudanças como uma oportunidade para se encontrar padrões relacionados a estas mudanças. Nestes casos, técnicas provenientes da área de pesquisa em SGBDs ativos podem ser úteis. Em [5], é introduzido o conceito de *paradigma de mineração de dados ativa* utilizando conceitos de sistemas de bancos de dados ativos. Neste paradigma, é armazenado um histórico de regras previamente extraídas de um banco de dados. Quando este histórico passa a exibir certas tendências, especificadas como *consultas moldes*, definidas pelo usuário, ações são inicializadas. Em [6], é apresentada uma linguagem para definição de consultas moldes. Em [108], apresenta-se um esquema de incorporação de regras extraídas de forma que a base de regras se mantenha consistente.

Outra questão envolvendo a área de Bancos de Dados é que esta última também pode se beneficiar de técnicas utilizadas em MD (principalmente das técnicas de Aprendizado de Máquina). Por exemplo, em [57] um algoritmo de aprendizado indutivo é utilizado para se gerar regras semânticas com o objetivo de otimizar o processamento de consultas em um banco de dados relacional. Outro exemplo desta simbiose entre Bancos de Dados e MD pode ser encontrado em [14], onde é proposta uma abordagem baseada em conhecimento para a fase de fragmentação do projeto de um SGBD distribuído orientado a objetos.

2.3.4 Data Warehousing

Uma outra área relacionada a MD é *Data Warehousing*, que se refere ao processo de coleta e pré-processamento dos dados armazenados em um ou mais bancos de dados operacionais, com o objetivo de servir de fonte para *Sistemas de Suporte a Decisão* [43]. O resultado deste processo é a criação de um *depósito de dados* (DD), uma coleção de dados integrados, possivelmente estruturados no tempo (dados históricos) [62].

A fase de integração deve levar em conta que os bancos de dados utilizados na

coleta dos dados para o DD podem ter esquemas [7] heterogêneos. Estes dados coletados sofrem um pré-processamento que se assemelha ao passo de pré-processamento (vide Seção 2.2.2) do processo de MD, não levando em conta o fato de um DD poder ser formado a partir de fontes heterogêneas. Diversas são as vantagens de se utilizar DD para se realizar uma tarefa de MD. Dentre elas, pode-se citar [23]:

- Em virtude de os dados armazenados em um DD já terem passado por fases de integração e de pré-processamento, a realização de tarefas de mineração de dados sobre um DD se torna uma atividade menos complexa.

- As informações em um DD são utilizadas com o único objetivo de se fazer análise dos dados para suporte a decisões, sendo que estes dados têm uma taxa de atualização muito mais baixa do que a encontrada em bancos de dados operacionais. Em um DD, portanto, encontra-se um ambiente muito menos complexo do ponto de vista da mutabilidade dos dados. Entretanto, em um DD, há a necessidade de armazenamento desses dados através do tempo, ao contrário de SGBDs operacionais onde, normalmente, somente os dados mais atuais são mantidos.

- Em virtude de haver dados históricos em um DD, um analista de dados pode detectar tendências nestes dados, o que provavelmente não seria possível em um banco de dados operacional, onde só se encontra a versão mais atual do conjunto de dados em questão.

2.4 Tarefas de Mineração de Dados

Uma tarefa de MD está relacionada a um problema a ser resolvido [40]. Esta tarefa é definida no passo de definição do problema do passo de MD (Seção 2.2.2). Existem inúmeras tarefas de MD descritas na literatura, sendo que as mais comumente encontradas são: Classificação, Agrupamento, Modelagem de Dependências,

[7] Um esquema de banco de dados é constituído de um conjunto de meta-informações sobre estes dados, como, por exemplo, nome e tipo de cada atributo, e de que forma atributos se interrelacionam.

Regressão, Análise de Ligações, e Análise de Desvios. Para cada tarefa, há diversos métodos (Seção 2.5) que podem ser aplicados para se alcançar o objetivo definido. Esta seção descreve sucintamente estas tarefas, à exceção da tarefa de classificação, que por ser parte do escopo desta dissertação, é descrita em maiores detalhes no Capítulo 3.

2.4.1 Agrupamento (*Clustering*)

Esta é uma tarefa descritiva, ou seja, que gera padrões descritivos (vide Seção 2.2.1). Na tarefa de agrupamento, o objetivo é particionar uma coleção de dados em um número finito de grupos, tais que as tuplas dentro de um mesmo grupo sejam similares de acordo com alguma métrica. Geralmente esta métrica é definida de tal forma que as similaridades dentro de um grupo são maximizadas e as similaridades entre grupos são minimizadas. Estes grupos têm que ser determinados a partir da coleção de dados e podem ser mutuamente exclusivos e exaustivos.

Há dois tipos básicos de agrupamento: *hierárquico* e *não hierárquico*. No primeiro, os grupos formam uma hierarquia. Já no segundo tipo, o número de grupos a serem gerados é um parâmetro de entrada do algoritmo. Em geral, a construção de grupos hierárquicos é menos cara computacionalmente do que a de grupos não hierárquicos.

Modelos de agrupamento existem em diferentes formas, mas, usualmente, os padrões existem matematicamente como uma função que associa cada item no conjunto de treinamento a algum grupo. Informações estatísticas sobre as propriedades de cada grupo podem ajudar o especialista de domínio na exploração e sumarização dos dados.

No processo de formação de grupos, são determinados grupos de objetos relacionados na coleção. Após isto, são produzidas descrições destes grupos. Estas descrições podem ser regras descrevendo cada um dos grupos. O processo de decisão acerca de como utilizar os padrões de agrupamento é realizado pelo analista de domínio e pelo analista de dados, que trabalham em conjunto para entender cada grupo. É feita a associação de um nome de rótulo a cada grupo, para ajudar os

usuários a lembrarem das propriedades daquele grupo.

2.4.2 Modelagem de Dependências Causais

Um melhor entendimento dos dados pode ser obtido a partir da derivação de alguma estrutura causal subjacente a estes dados. Modelos de causalidade podem ser probabilísticos (por exemplo, a descoberta de algo sobre a distribuição de probabilidade governando os dados) ou determinísticos (por exemplo, derivando dependências funcionais entre campos de dados). Métodos de estimativa de densidade e Redes Bayesianas [52] são exemplos de métodos utilizados nesta tarefa de MD.

2.4.3 Regressão

Regressão (termo proveniente da área de estatística) faz referência ao descobrimento de padrões onde os atributos sobre os quais se quer fazer predição são valores no domínio real. Na verdade esta é a principal diferença entre esta tarefa e a de classificação, onde os atributos que se quer predizer são de domínio discreto.

Um caso particular de regressão bastante utilizado em Análise Estatística [49] é a *regrassão linear*, onde estes atributos são modelados como uma função linear dos atributos de entrada.

Um modelo de regressão pode ser utilizado, por exemplo, por uma instituição bancária para auxiliar no processo de manutenção e/ou retenção de clientes. Além de saber que clientes são prováveis de cancelar suas contas, seria desejável conhecer o valor de se manter um cliente. Se o valor de se manter um cliente é na verdade negativo então não há nenhum motivo em se fazer esforço para manter este cliente, e de fato este banco se beneficiaria do cancelamento de sua conta.

2.4.4 Análise de Ligações

A tarefa de Análise de Ligações (AL) é identificada na literatura com a extração de *regras de associação*. Regras de associação são padrões descritivos bastante simples da forma $X \rightarrow Y$, onde X e Y são atributos que assumem valores binários.

Assim, pode-se descobrir regras da forma "*chupeta* $\rightarrow$ *cerveja*". Esta regra indica que sempre que alguém compra chupetas em um supermercado, compra cervejas também.

Normalmente, há uma medida de certeza associada a uma regra de associação. A medida mais comum do grau de utilidade das regras é fornecida por um par de grandezas: *suporte* e *confiança*.[8] Suporte é a probabilidade de que uma transação contenha tanto os atributos de X quanto os atributos de Y. Confiança é a probabilidade de que uma transação contenha os atributos de Y, dado que ela contém os atributos de X. Um critério geralmente utilizado é o de que o suporte de uma regra deve estar acima de um determinado limite para esta ser considerada útil. Um típico valor para este limite está no intervalo $[10^{-2}, 10^{-4}]$. Já a confiança pode assumir qualquer valor no intervalo [0, 1].

Há bastante literatura na área de regras de associação. Um algoritmo iterativo para extração de regras de associação foi primeiramente descrito por Agrawal em [3] e posteriormente melhorado em [2] e [7]. Uma versão paralela deste algoritmo é descrita em [9]. Em [53], Han et al. apresentam um algoritmo para extrair regras de associação em múltiplas dimensões. Em [79], o desempenho deste algoritmo é melhorado através da utilização de análise combinatória da informação obtida nas iterações anteriores da execução deste algoritmo.

Um fato interessante é que regras de associação foram primeiramente introduzidas pela comunidade de Bancos de Dados. Este tipo de padrão certamente tem muitas aplicações em potencial que ainda não foram exploradas. No entanto a quantidade de regras possíveis é enorme, e a tarefa de filtragem das regras mais interessantes e informativas de modo eficiente é ainda um problema em aberto.

Outro tipo de padrão que pode ser obtido em (AL) são *padrões seqüenciais*. Muitas lojas de venda a varejo têm programas de distribuição de cartões de identificação a seus clientes. Com esta estratégia, é possível a extração destes padrões seqüenciais [8, 80], que podem ser considerados como regras de associação onde X representa um conjunto de itens comprados durante uma visita à loja e Y representa

[8] Traduções dos termos *support* e *confidence*, respectivamente.

itens comprados em outra visita a mesma loja.

2.4.5 Análise de Desvios

Análise de Desvios (AD) compreende a descoberta de uma grande classe de padrões que exprimem anomalias em relação a uma determinada tendência encontrada nos dados. A maioria dos métodos para a detecção de desvios têm em comum o objetivo de procurar diferenças significantes entre uma observação e uma referência [72]. A observação é usualmente um valor de um atributo ou da combinação de mais de um atributo, e a referência pode ser outra observação.

A tarefa de AD é uma das mais simples encontradas em MD, em virtude de os padrões resultantes não serem preditivos e sim descritivos. Existe razoável literatura nesta área. Em [82], é descrito o sistema *KEFIR* e sua aplicação da detecção de desvios em um banco de dados da área de saúde.

2.5 Métodos de Mineração de Dados

No passo de Extração de Padrões do processo de MD, pode-se utilizar diversos métodos (ou paradigmas) de aprendizado indutivo. Esta Seção descreve as diferenças entre este tipo de aprendizado e o aprendizado dedutivo para dar uma definição do que seja um método de MD. Primeiramente, deve-se fazer uma análise das duas principais técnicas utilizadas para se fazer inferência (ou aprendizado) de informações a partir de uma coleção de dados: *dedução* e *indução*.

Dedução é a técnica utilizada para inferir informações que são uma *conseqüência lógica* da informação armazenada na coleção de dados. Esta técnica é encontrada em SGBDs dedutivos, onde são armazenados fatos (ou relações) e regras, com as quais pode-se derivar informações a partir dos fatos. Note que este tipo de inferência gera informações que são uma conseqüência lógica dos dados utilizados, ou seja, não há geração de informações ou padrões novos. Por este motivo, a técnica de dedução não é muito utilizada em mineração de dados, onde, segundo a Definição 2.1, os padrões interessantes devem ser novos. No entanto, a integração de técnicas

dedutivas e indutivas parece ser um bom caminho para aumentar a qualidade dos padrões extraídos em sistemas de MD. Em [95], por exemplo, é discutido como bancos de dados dedutivos, visualização de dados, e indução de regras (ver abaixo) podem ser utilizados cooperativamente para se fazer MD.

Na indução, há a inferência de informação que constitui uma *generalização* dos dados utilizados. Por exemplo, considere as relações Empregados, Gerentes e Departamentos em um banco de dados relacional. Utilizando-se este paradigma, pode-se *induzir* que cada empregado tem um gerente, o que é uma generalização a partir dos dados existentes naquelas relações. Pode ser que haja pelo menos um empregado que não tenha gerente. Porém, na indução, este fato não é importante, contanto que a generalização produzida se mostre válida na maioria dos casos. Ao se utilizar o paradigma de indução, diz-se estar realizando um *aprendizado indutivo.*[9]

A diferença mais importante entre dedução e indução é que a primeira resulta em declarações verdadeiras (absolutas) sobre a coleção de dados utilizada na inferência, enquanto que a segunda resulta em declarações *provavelmente* corretas sobre esta coleção.

A partir dos conceitos acima definidos, pode-se dar uma definição do termo *método de mineração de dados*:

Definição 2.3 *Métodos de mineração de dados envolvem a utilização de alguma técnica de aprendizado indutivo, objetivando a adequação (inferência) de um modelo ou a determinação de similaridades a partir de um conjunto de dados (ambiente).*

Alguns exemplos de métodos de MD encontrados na literatura são Computação Evolucionária, Redes Neurais, Indução de Regras, Aprendizado Bayesiano, e Aprendizado Baseado em Instâncias. Alguns destes métodos são descritos no Capítulo 3 no contexto da tarefa de classificação, em virtude de esta tarefa constituir assunto da parte principal desta tese.

[9]Segundo Holland em [55], este processo é análogo ao que acontece com seres humanos e outras criaturas inteligentes (denominados *sistemas cognitivos*). Estes tentam entender o ambiente a sua volta através do uso de uma generalização desse ambiente, ou seja, de um *modelo*. Durante a fase de aprendizado, o sistema cognitivo observa o ambiente e reconhece similaridades entre objetos e eventos, similaridades estas que servem para a generalização do modelo.

Capítulo 3

A Tarefa de Classificação

Seja este o vosso modo de falar: Sim, sim; não, não; tudo o que for além disto procede do espírito mal.

Mateus, 5:37

3.1 Introdução

Classificação é uma tarefa que gera padrões de predição semelhantes à tarefa de regressão (Seção 2.4.3), exceto que a primeira prediz o valor de um atributo nominal ou categórico ao invés de um atributo de valor real. O atributo alvo da predição é chamado *classe*. Possíveis aplicações desta área podem ser a predição do comportamento dos clientes de um banco, a sinalização de transações fraudulentas, predição de ações de valores, entre outras.

Este Capítulo faz uma revisão dos principais conceitos encontrados na tarefa de classificação. A Seção 3.2 descreve o objetivo desta tarefa. Na Seção 3.3 são descritas as técnicas utilizadas para se avaliar o modelo gerado. Finalmente, na Seção 3.4 são descritos os principais métodos que podem ser utilizados nesta tarefa.

3.2 Objetivo da Classificação

Nesta tarefa, os atributos da relação mina são particionados em dois grupos. Um dos grupos contém somente um atributo, que corresponde ao *atributo alvo*, ou seja, o atributo do qual se deve fazer a predição da classe. O outro grupo contém os atributos a serem utilizados na predição da classe, denominados *atributos previsores* ou *atributos de predição*. A Tabela 3.1 é uma adaptação de outro exemplo encontrado em [75] (mas, primeiramente apresentado em [85]).[1] O atributo *PlayTenis* é o atributo alvo e os atributos *OutLook*, *Temperature*, *Humidity* e *Wind* são os atributos previsores.

Tabela 3.1: Conjunto de treinamento para o atributo alvo *PlayTenis*.

OutLook	*Temperature*	*Humidity*	*Wind*	*PlayTenis*
sunny	*hot*	*high*	*weak*	*no*
sunny	*hot*	*high*	*strong*	*no*
overcast	*hot*	*high*	*weak*	*yes*
rain	*mild*	*high*	*weak*	*yes*
rain	*cool*	*normal*	*weak*	*yes*
rain	*cool*	*normal*	*strong*	*no*
overcast	*cool*	*Normal*	*strong*	*yes*
sunny	*mild*	*high*	*weak*	*no*
sunny	*cool*	*normal*	*weak*	*yes*
rain	*mild*	*normal*	*weak*	*yes*
sunny	*mild*	*normal*	*strong*	*yes*
overcast	*mild*	*high*	*strong*	*yes*
overcast	*hot*	*normal*	*weak*	*yes*
rain	*mild*	*high*	*strong*	*no*

[1]Note que, na prática, um conjunto de treinamento contém muito mais instâncias do que as deste exemplo, e o número de atributos previsores também é muito maior.

O objetivo da tarefa de classificação é, através da utilização de algum método, gerar um *modelo de classificação* a partir da relação mina (conjunto de treinamento), de tal forma que este modelo permita a classificação de *novas* tuplas, ou seja, de tuplas que não foram utilizadas para a geração do modelo. Por este motivo é que o conjunto de treinamento utilizado deve representar a real distribuição de valores dos atributos. Por exemplo, se o conjunto de treinamento da Tabela 3.1 só possuísse tuplas para as quais o atributo alvo *PlayTenis* tivesse valores *yes*, o modelo assim gerado não teria a capacidade de predizer situações em que este atributo tivesse valor igual a *no*.

3.3 Avaliação do Modelo de Classificação

Uma vez gerado o modelo de classificação, deve-se estimar a *precisão* do mesmo, ou seja, o quão efetivo este é na predição da classe de novas tuplas. A precisão do modelo é medida pela sua *taxa de erro*, que mede o número de predições incorretas feitas. Há dois tipos de taxa de erro [106], a *taxa erro verdadeira* (TEV) e a *taxa de error aparente* (TEA). A TEV é definida como a taxa de erro do modelo quando aplicado a um conjunto assintoticamente grande de tuplas novas, que converge, no limite, para a distribuição de valores real dos atributos. A TEA é a taxa de erro calculada considerando-se as tuplas do conjunto de treinamento.

Se um número ilimitado de tuplas for fornecido como o conjunto de treinamento, a TEA converge para a TEV. No entanto, em situações reais, tanto o conjunto de treinamento quanto o conjunto de teste têm tamanho limitado, fato que torna a TEA uma estimativa pobre da TEV. Usualmente a TEA é otimista em relação à TEV, normalmente pela fato de ter havido uma superespecialização (*overfitting*) sobre os dados na fase de adequação do modelo [90]. Em virtude disto, são utilizadas técnicas diferentes para se estimar o valor da TEV.

Uma técnica bastante utilizada para se fazer uma *estimativa* da TEV é utilizar um *conjunto de dados de teste* (ou, simplesmente, *conjunto de teste*). Este conjunto de teste contém tuplas que não foram utilizadas na fase de geração do modelo de

classificação. O modelo é aplicado a este conjunto de teste para se calcular a taxa de erro sobre este conjunto, denominada *taxa de erro de teste* (TET). Resultados práticos [106] mostram que a TET é uma boa estimativa para a TEV, mesmo quando o número de tuplas no conjunto de teste não é tão grande (da ordem de 10^3).

Outra técnica bastante utilizada para se estimar a TEV é a *validação cruzada* (cross-validation). Nesta técnica, as tuplas são aleatoriamente divididas em k partições de tamanho aproximadamente iguais. As tuplas não presentes em uma dada partição são utilizadas para a geração do modelo de classificação. Este modelo é testado, utilizando-se a partição correspondente. Assim, são gerados k modelos, cada um com sua própria TET. A média destas k TETs constitui a estimativa para a TEV do modelo gerado.

O modelo de classificação gerado pode ser também avaliado com relação a outros critérios, além da precisão preditiva. Critérios comumente usados em MD incluem a compreensibilidade (ou simplicidade) do modelo e o grau de interesse dos padrões descobertos.

3.4 Métodos de Classificação

Na Seção 3.2, é mencionado que um modelo de classificação é gerado através de um *método de classificação*. Esta Seção descreve diversos métodos de classificação existentes na literatura. Note, entretanto, que cada algoritmo que implementa um destes métodos tem suas próprias particularidades. A descrição aqui feita ignora estas particularidades e enfoca o processo geral utilizado em cada método.

Todos os métodos descritos a seguir trabalham sobre uma relação mina, tendo como objetivo a predição do atributo alvo a_c a partir dos atributos previsores a_i, $\{a_i \mid 1 < i < n\}$. Na Seção 3.4.1, o método de Indução de Regras é descrito. Na Seção 3.4.2, o método de Aprendizado Bayesiano Simples é apresentado. A Seção 3.4.3 descreve o método de Aprendizado Baseado em Instâncias. Os três métodos acima são de particular importância para o assunto principal desta tese, discutido no Capítulo 4, visto que estes métodos são adaptáveis às primitivas ali apresentadas.

Nas Seções 3.4.5 e 3.4.4 são descritos os métodos de *Redes Neurais* e *Computação Evolucionária*, respectivamente. Estes dois métodos, embora não fazendo parte do foco desta tese, são aqui descritos para fins de comparação. Para esta Seção, considere as seguintes presuposições:

1. Os termos *exemplo*, *instância*, *caso*, etc., como utilizados pelas comunidades de Aprendizado de Máquina e/ou Estatística, têm aqui o sinônimo de *tupla*.

2. A relação mina é denotada por *Relação* $\mathcal{R}$, ou, simplesmente, $\mathcal{R}$. Considera-se que cada tupla da relação mina tenha a forma $(a_1, a_2,..., a_n, a_c)$, onde $\{a_i \mid 1 < i < n\}$ é o conjunto de *atributos de predição* (alternativamente denominados *atributos previsores*), e a_c é o *atributo alvo*.

3.4.1 Indução de Regras

O método de Indução de Regras (IR) é bastante utilizado para classificação pelo fato de produzir padrões inteligíveis. Os padrões gerados neste método são *regras de produção*. Uma regra de produção é uma declaração da forma $C_1 \rightarrow C_2$, onde C_1 e C_2 são o *antecedente* e o *conseqüente* da regra, respectivamente. Estes, por sua vez, geralmente são compostos de uma conjunção de predicados e de um predicado, respectivamente. Predicados são relações sobre objetos e/ou valores de um banco de dados da forma $a_i \otimes v$ onde a_i faz referência ao i-ésimo atributo da tupla t, v é um valor qualquer pertencente a $\mathrm{Dom}(a_i)$, e $\otimes$ é um operador relacional ($\geq$, $\leq$ $<$, $>$ ou $=$). Um exemplo de regra de produção pode ser $OutLook = rain \wedge Wind = strong \rightarrow PlayTenis = no$. Esta regra denota que sempre que os atributos previsores *OutLook* e *Wind* assumem os valores *rain* e *strong*, respectivamente, é previsto o valor *no* para a classe do atributo alvo *PlayTenis*.

Normalmente, os algoritmos que se encaixam neste método produzem inicialmente uma estrutura de representação de conhecimento denominada *árvore de decisão*, que posteriormente é transformada no conjunto de regras de produção.[2] Na verdade,

[2]Por este motivo, algoritmos para indução de árvores de decisão são também chamados algoritmos de indução de regras [106].

árvores de decisão e regras de produção são padrões equivalentes, no sentido de que pode-se converter uma representação na outra [84].

Uma árvore de decisão é constituída de uma série de nós internos, onde cada um destes nós está associado a um atributo previsor. Partindo de um determinado nó interno, tem-se k arestas (ramos), onde k é o número de possíveis valores do atributo previsor. Cada uma destas arestas termina em outro nó da árvore, que pode ser outro nó interno ou uma folha. As folhas (nós externos) da árvore de decisão correspondem a valores do atributo alvo, ou seja, a uma predição da classe deste atributo.

Na Figura 3.1 (adaptada de [75]) é apresentado um exemplo de árvore de decisão. Nesta árvore, há três nós internos, correspondentes aos atributos previsores *OutLook*, *Humidity* e *Wind*. De cada um destes nós, partem arestas que terminam em outros nós ou em folhas. Considere, por exemplo, o atributo *OutLook*, correspondente à raiz da árvore. Deste atributo, partem três arestas, sendo que o correspondente ao valor *overcast* termina em uma folha que prediz o valor *yes* para o atributo alvo *PlayTenis*. As outras duas arestas saindo deste nó terminam nos nós internos correspondentes aos atributos *Wind* e *Humidity*. Note que estes dois nós internos podem ser considerados como raízes de suas respectivas sub-árvores.

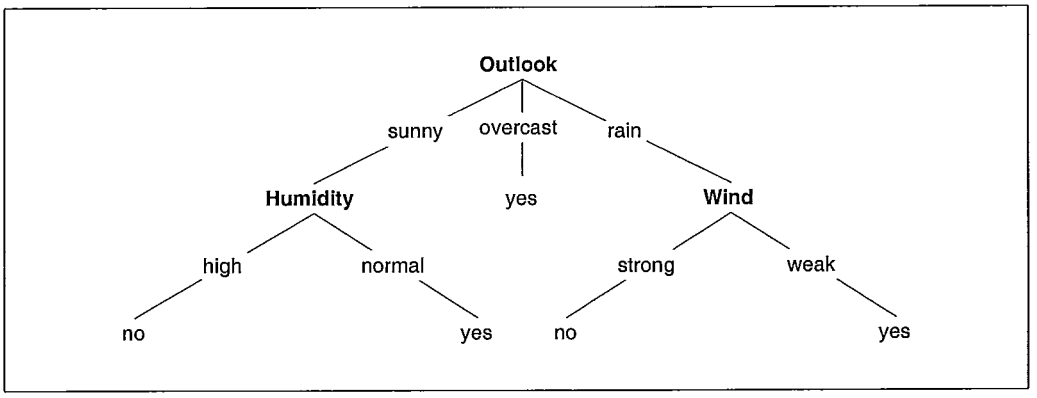

Figura 3.1: Exemplo de árvore de decisão. Os rótulos nos ramos correspondem aos valores do atributo especificado no vértice de onde se originam tais ramos. Os nós folhas correspondem a valores do atributo alvo, *PlayTenis*.

Árvores de decisão, na sua forma normal, são *univariáveis*, significando que cada vértice corresponde a um único atributo previsor. Árvores univariantes, como é o

caso da árvore da Figura 3.1, são o tipo mais estudado na literatura. No entanto, pode-se encontrar trabalhos sobre árvores de decisão *multivariáveis* ou *oblíquas*, nas quais o teste feito em um dos nós da árvore é constituído de uma combinação linear dos atributos previsores [76].

De uma forma genérica, pode-se dizer que um algoritmo de IR possui os seguintes parâmetros de entrada:

1. Uma relação $\mathcal{R}$ correspondente ao *conjunto de treinamento*, onde, em cada uma de suas tuplas, um dos atributos é o *atributo alvo* e os demais são os *atributos previsores*, ou *atributos candidatos* (vide Seção 3.2);

2. Uma *função de avaliação* que determina a qualidade de um determinado atributo com relação ao quão bem este atributo, por si só, classifica o conjunto de treinamento.

3. Um *critério de parada* que determina quando a expansão da árvore deve terminar.

O algoritmo de IR segue iterativamente, a cada passo gerando uma sub-árvore da árvore de decisão final. A árvore inicial é constituída de um único nó, a raiz, ao qual estão associadas todas as tuplas da relação $\mathcal{R}$. Um dos atributos do conjunto de *atributos previsores* é escolhido para formar a raiz da árvore, de acordo com a *função de avaliação* sendo utilizada. Uma vez escolhido este atributo, o conjunto de dados é particionado, segundo os valores deste atributo. Cada uma dessas partições é associada a uma das arestas que saem do nó correspondente ao atributo selecionado. Os demais nós da árvore são produzidos aplicando-se recursivamente o algoritmo a cada uma das partições. A cada iteração, se o *critério de parada* não é satisfeito, a árvore é novamente expandida.

Em relação à *função de avaliação*, esta varia de um algoritmo de IR para outro. A seguir são examinados três tipos de funções de avaliação utilizadas em algoritmos de IR presentes na literatura:

1. **Ganho de Informação (*Information Gain*)**. Esta medida, utilizada nos algoritmos ID3 [83] e C4.5 [85], se baseia no conceito de *entropia*, comumente

utilizada na área de *Teoria da Informação*. O atributo selecionado é aquele que maximiza o ganho de informação. Sejam $\mathcal{C}$ uma coleção de tuplas e c a cardinalidade de $Dom(a_c)$. A entropia de $\mathcal{C}$ com relação aos c valores de a_c é dada por

$$Entropia(\mathcal{C}) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{3.1}$$

onde p_i é a fração da coleção $\mathcal{C}$ cujo atributo alvo é da classe i. O ganho de informação, $G(\mathcal{C}, a_i)$, de um determinado atributo candidato a_i, com relação a uma coleção C, é definido como

$$G(\mathcal{C}, a_i) = Entropia(\mathcal{C}) - \sum_{v=1}^{c_i} \frac{|\mathcal{C}_v|}{|\mathcal{C}|} Entropia(\mathcal{C}_v) \tag{3.2}$$

onde c_i é o número de elementos de $Dom(a_i)$, e $\mathcal{C}_v$ é a subcoleção de $\mathcal{C}$ na qual o atributo candidato a_i tem valor v.

2. **Taxa de Ganho de Informação (*Information Gain Ratio*)**. Esta medida, indicada aqui por G^*, é uma medida definida em termos da medida anterior, ganho de informação (Equação 3.2):

$$G^*(\mathcal{C}, a_i) = \frac{G(\mathcal{C}, a_i)}{SI(\mathcal{C}, a_i)} \tag{3.3}$$

onde $SI(\mathcal{C}, a_i)$ é dada por

$$SI(\mathcal{C}, a_i) = -\sum_{v=1}^{c_i} \frac{|\mathcal{C}_v|}{|\mathcal{C}|} \log_2 \frac{|\mathcal{C}_v|}{|\mathcal{C}|} \tag{3.4}$$

onde C eqüivale a $\Pi(a_i)$, e as coleções de tuplas C_v são resultantes da partição de C pelos v valores do atributo a_i.

3. **Redução do Índice Gini (*Reduction of the Gini Index*)**. Esta medida, primeiramente proposta em [16], é utilizada nos algoritmos *SPRINT* [89] e

CART [16]. Esta medida se baseia na redução do *Índice Gini*, que é definido pela Equação 3.5:

$$Gini(\mathcal{C}) = 1 - \sum_{v=1}^{c_i} \frac{|\mathcal{C}_v|}{|\mathcal{C}|} \tag{3.5}$$

Para um conjunto de dados contendo tuplas de v classes diferentes, a medida de *Redução do Índice Gini*, GR, é definida pela seguinte expressão:

$$GR(\mathcal{C}, a_i) = Gini(\mathcal{C}) - \sum_{v=1}^{c_i} \frac{|\mathcal{C}_v|}{|\mathcal{C}|} Gini(\mathcal{C}_v) \tag{3.6}$$

Para exemplificar como uma função de avaliação é utilizada para seleção de um atributo a partir de uma conjunto de atributos candidatos, considere a medida $G(\mathcal{C}, a_i)$, o ganho de informação. Seja calcular esta medida para os atributos previsores da coleção C correspondente à Tabela 3.1, através da aplicação da Equação 3.2:

G(C, OutLook) = 0.246
G(C, Humidity) = 0.151
G(C, Wind) = 0.048
G(C, Temperature) = 0.029

O atributo que é associado à raiz da árvore é aquele que maximiza o ganho de informação. Examinando-se os valores acima, pode-se concluir que o atributo *OutLook* é o selecionado para o nó raiz, conforme ilustrado pela Figura 3.1. As tuplas de C são, então, particionadas pelos ramos partindo da raiz, correspondentes aos valores de *OutLook* encontrados em C. Esta mesma função de avaliação é aplicada recursivamente aos nós nos quais estes ramos chegam, à exceção do nó terminal (folha) onde o ramo para o valor *overcast* termina.

Exemplos de algoritmos de IR que seguem esta forma geral de geração da árvore de decisão são ID3 [83] e C4.5 [85]. Em [32], é proposto um novo algoritmo para indução de regras, CWS (**C***onquering* **W***ithout* **S***eparating*). O CWS adota uma

abordagem diferente daquela de "dividir e conquistar". Esta abordagem intercala a construção do conjunto de regras com a avaliação de cada regra, no contexto do conjunto de regras geradas até o momento. Outro algoritmo que não utiliza a abordagem de dividir e conquistar é o CN2 [21, 26]. Este algoritmo age de uma forma iterativa, em cada iteração procurando por uma conjunção de predicados (denominada *complexo*) que satisfaça a um grande número de exemplos de uma única classe C e a poucos exemplos das demais classes. Uma vez determinado este complexo, os exemplos que o satisfazem são removidos do conjunto de treinamento e a regra *se complexo então* C é adicionada à lista de regras gerada pelo algoritmo. Para fins de restrição de escopo, note-se que as primitivas propostas nesta tese dão suporte a algoritmos que utilizam a abordagem "dividir e conquistar", diferentemente dos algoritmos CWS e CN2.

Em virtude de uma árvore de decisão ser expandida gradualmente, surge a questão de quando é o melhor momento no qual esta expansão deve ser interrompida. Para resolver este problema, normalmente é utilizada a técnica de *poda* da árvore. A poda de um nó da árvore de decisão consiste (1) da remoção da sub-árvore com raiz neste nó, tornando-o um nó folha, e (2) da atribuição do valor de classificação mais comum dentre as tuplas associadas àquele nó. Esta técnica tem os objetivos de remover as ramificações da árvore associadas a dados espúrios, através da seleção da sub-árvore com a mínima taxa de erro estimada, e melhorar a compreensão das regras geradas a partir da árvore de decisão [12].

Há duas abordagens principais para se fazer a poda de uma árvore de decisão: *pré-poda* e *pós-poda*.

- Pré-poda: Nesta abordagem, o particionamento da árvore de decisão pode chegar ao fim durante a fase de expansão. Usualmente, o critério de parada é calculado para dar uma estimativa do ganho esperado em expansões adicionais da árvore e a expansão é terminada quando um limite mínimo de ganho não é esperado.

- Pós-poda é a abordagem mais utilizada de poda. Uma variante desta aborda-

gem é utilizada pelo algoritmo C4.5 [85]. Esta variante tem os seguinte passos principais [75]:

1. Indução da árvore de decisão, conforme descrito anteriormente;
2. Conversão da árvore de decisão em um conjunto de regras de produção equivalente, através da criação de uma regra para cada caminho da árvore da raiz até um nó terminal (folha);
3. Poda (generalização) de cada regra através da remoção de predicados que resultem em um melhoramento da precisão estimada;
4. Ordenação das regras resultantes da generalização, levando-se em consideração sua precisão estimada

Nesta abordagem, nós internos são removidos somente se a precisão de classificação das regras após a remoção não é pior quando comparada à precisão da árvore original, considerando-se o conjunto de validação separado do conjunto de treinamento.

3.4.2 Aprendizado Bayesiano Simples

A abordagem utilizada no Aprendizado Bayesiano[3] Simples (ABS) para classificar uma nova tupla é associar a seu atributo alvo o valor mais provável [75]. Neste método, é feita uma análise dos relacionamentos existentes entre cada um dos atributos previsores e o atributo alvo, para gerar uma probabilidade condicional para cada um desses relacionamentos. Quando uma nova tupla deve ser classificada, a predição é feita através da combinação dos efeitos de cada atributo previsor sobre a atributo alvo.

Em teoria, o ABS é somente aplicável se todos os atributos previsores são estatisticamente independentes. Por exemplo, dados acerca de um cliente contêm atributos (tais como peso, escolaridade, salário, etc.) que estão relacionados a sua

[3] O termo "bayesiano" provém de Thomas Bayes, nome de um ministro britânico que viveu no século XVIII e que formulou o famoso *Teorema de Bayes.*

idade. Neste caso, a utilização do ABS superestimaria o efeito do atributo idade. Não obstante esta limitação, a prática mostra que o ABS é bastante utilizado em virtude de sua simplicidade e rapidez na investigação de padrões simples.

Durante a fase de geração do modelo, a probabilidade de cada valor $v(a_c)$ do atributo alvo é calculada através da contagem de quantas vezes este ocorre no conjunto de treinamento. Esta probabilidade é denominada *probabilidade anterior* (prior probability). Por exemplo, na Tabela 3.1, a probabilidade anterior associada ao valor *yes* do atributo alvo *PlayTenis* é igual a $\frac{9}{14}$ (note que, do total de 14 tuplas, 9 têm o valor de *PlayTenis* igual a *yes*).

Em adição às probabilidades anteriores, deve-se também calcular o quão freqüentemente cada valor de cada atributo previsor ocorre em combinação com cada um dos valores do atributo alvo. Esta freqüências são, então, utilizadas para o cálculo de *probabilidades condicionais* que, juntamente com as probabilidades anteriores, são utilizadas para se fazer a predição do valor da classe da nova tupla.

Mais formalmente [75], quando uma nova tupla $t = (v(a_1), ..., v(a_n), ?)$ (onde $v(a_i)$ corresponde ao valor do atributo previsor a_i) é apresentada, tem-se que o valor mais provável do atributo alvo para esta tupla, v^*, é obtido pela aplicação da Equação 3.7.

$$v^* = argmax_{v_j \in Dom(a_c)} P(v_j|t) \tag{3.7}$$

Nesta Equação, *argmax* é um operador que retorna o valor do atributo alvo associado ao valor máximo dentre todos os valores de $P(v_j|t)$ calculados para cada valor v_j no domínio do atributo alvo a_c. O Teorema de Bayes pode ser utilizado para escrever a Equação 3.7 de outra forma:

$$v^* = argmax_{v_j \in Dom(a_c)} P(t|v_j)P(v_j) \tag{3.8}$$

Note que os valores $P(t|v_j)$ e $P(v_j)$ correspondem, respectivamente, à probabilidade da tupla t dado v_j e à probabilidade anterior do valor v_j, que podem ser estimadas a partir do conjunto de treinamento. Para descrever estas estimativas,

seja $nt(C)$ o número de tuplas na relação $\mathcal{R}$ que satisfazem à condição C, onde C corresponde a uma conjunção de predicados. Com base na definição de nt, pode-se estimar o valor da probabilidade anterior $P(v_j)$ através da Equação 3.9:

$$P(v_j) = nt(a_c = v_j)/nt(true), \tag{3.9}$$

onde, $nt(a_c = v_j)$ corresponde ao número de tuplas em $\mathcal{R}$ nas quais o atributo alvo a_c tem o valor v_j, e $nt(true)$ corresponde ao número de tuplas em $\mathcal{R}$.

Para a estimativa do valor $P(t|v_j)$, deve-se considerar (1) cada atributo a_i da tupla t e (2) o tipo deste atributo, se numérico ou simbólico. O valor $P(t|v_j)$ é calculado pelo produtório dos valores $P(a_i = v(a_i)|a_c = v_j)$, onde cada um destes fatores é dado pela Equação 3.10:

$$P(a_i = v(a_i)|a_c = v_j) = \begin{cases} g(v(a_i); \mu_{v_j}, \sigma^2_{v_j}) & \text{se } a_i \text{ é numérico} \\ nt(a_i = v(a_i), a_c = v_j)/nt(a_c = v_j) & \text{se } a_i \text{ é simbólico} \end{cases} \tag{3.10}$$

Nesta equação, μ_{v_j} e $\sigma^2_{v_j}$ são a média e a variância de $v(a_i)$ para o valor v_j do atributo alvo a_c, e g é a função de densidade *gaussiana* com média μ_{v_j} e variância $\sigma^2_{v_j}$.[4]

Para ilustrar o mecanismo de classificação de uma nova tupla através do aprendizado bayesiano, considere novamente a Tabela 3.1. Considere, ainda, que a nova tupla a ser classificada é (*sunny, cool, high, strong, ?*). O objetivo é predizer o valor do atributo alvo *PlayTenis* para esta nova tupla. Instanciando a Equação 3.8, o valor v^* é definido a seguir.

$$v^* = argmax_{v_j \in \{yes,no\}} P(sunny|v_j) \times P(cool|v_j) \times P(high|v_j) \times P(strong|v_j) \times P(v_j) \tag{3.11}$$

[4]Na verdade, esta é outra restrição associada ao método de ABS: considera-se que os atributos numéricos estejam distribuídos normalmente (ou seja, tenham uma distribuição normal).

Note que, na expressão final, os valores $v(a_i)$ são instanciados utilizando-se os valores dos atributos previsores da nova tupla. Para calcular v^*, calculam-se todos os fatores da expressão acima. Para as probabilidades anteriores, tem-se que $P(yes) = \frac{9}{14}$ e $P(no) = \frac{5}{14}$. De forma similar, calculam-se as probabilidades condicionais. Por exemplo, para $Wind = strong$, $P(strong|yes) = \frac{3}{9}$ e $P(strong|no) = \frac{3}{5}$. Calculando-se as demais probabilidades condicionais, obtêm-se os seguintes valores:

$$P(yes) \times P(sunny|yes) \times P(cool|yes) \times P(high|yes) \times P(strong|yes) = 0.0053$$
$$P(no) \times P(sunny|no) \times P(cool|no) \times P(high|no) \times P(strong|no) = 0.0206$$

Através da aplicação do operador *argmax* aos dois valores acima, pode-se deduzir que o valor de v^* para este exemplo e igual a *no*, visto que o valor máximo de $P(t|v_j)P(v_j)$ na equação 3.8 ocorre quando o atributo alvo *PlayTenis* assume este valor.

3.4.3 Aprendizado Baseado em Instâncias

Em ambos os métodos de classificação descritos acima, há a indução explícita de um modelo. Ao contrário destes métodos, no *Aprendizado Baseado em Instâncias* (ABI), não há a geração de um modelo explicitamente: os próprios dados da relação mina constituem o modelo. O método de ABI faz parte da família de métodos de *Aprendizado Baseado em Casos* (Case-Based Reasoning). Segundo Aha [1], os métodos desta família compartilham as seguintes etapas:

1. *Pré-processamento* da relação mina e da tupla a ser classificada;

2. *Cálculo de similaridade*, onde são calculadas a similaridades entre a tupla a ser classificada e as tuplas da relação mina;

3. *Predição*, onde, com base nas similaridades calculadas anteriormente, é feita a predição da classe da tupla apresentada;

4. *Atualização de memória*, que corresponde a modificar a relação mina em função da classe prevista. Tais atualizações podem envolver a remoção de

tuplas consideradas espúrias, a atualização do peso de um determinado atributo, etc.

Em algoritmos que implementam o método de ABI, o aprendizado consiste de simplesmente armazenar as tuplas, ou seja, o modelo são as próprias tuplas. Quando uma nova tupla é apresentada, um conjunto de instâncias similares à tupla apresentada é recuperado e utilizado para classificá-la.

O algoritmo mais representativo deste método é o *k-NN* [5]. Este algoritmo assume que todas as tuplas de $\mathcal{R}$ correspondem a pontos no espaço n-dimensional $\Re^n$, onde cada atributo de uma tupla corresponde a uma dimensão deste espaço. Mais formalmente, seja uma tupla t na relação $\mathcal{R}$ formada pelos atributos a_1, a_2, ..., a_n. Então, a seqüência de valores $v(a_1)$, $v(a_2)$, ..., $v(a_n)$, onde $v(a_i)$ é o valor do i-ésimo atributo de t, pode ser considerada como um ponto no espaço $\Re^n$.

Os pontos mais próximos a uma tupla apresentada são definidos em termos de uma *função de distância métrica.* Qualquer métrica utilizada deve resultar em um valor ordinal. As métricas mais utilizadas são as métricas *Euclidiana* e *Absoluta* (ou de *Manhattan*), definidas pelas Equações 3.12 e 3.13, respectivamente. Seja t_q a tupla a ser classificada, t_r uma tupla qualquer de $\mathcal{R}$, t_r^i e t_q^i cada um dos atributos de t_r e t_q, respectivamente. Então:

$$\Delta_e(t_r, t_q) = \sqrt{\sum_{i=1}^{n} w_i \times \delta_i^2} \tag{3.12}$$

$$\Delta_a(t_r, t_q) = \sum_{i=1}^{n} w_i \times \delta_i \tag{3.13}$$

onde δ_i é definida como

$$\delta_i = \begin{cases} |t_r^i - t_q^i| & \text{se } a_i \text{ é numérico} \\ 0 & \text{se } a_i \text{ é discreto e } t_r^i = t_q^i \\ 1 & \text{caso contrário} \end{cases} \tag{3.14}$$

[5] O termo k-NN é a abreviação de *k-Nearest Neighbor.*

Observe que as Equações 3.12 e 3.13 possuem um fator w_i, que corresponde ao valor do peso associado ao atributo a_i. Este peso é normalmente utilizado para reduzir o efeito dos atributos mais irrelevantes na predição de t_q. Quando maior o peso w_i de um atributo, maior a relevância daquele atributo para prever a classe da tupla.

Note que a Equação 3.14 para o cálculo de δ_i leva em consideração o tipo do atributo a_i. Quando este for um atributo discreto, adota-se a convenção de que δ_i é igual a 0 se os valores são iguais, e igual a 1 se são diferentes. Quando este atributo é do tipo numérico (inteiro, real, etc), o cálculo de δ_i é realizado da forma trivial, tomando-se o valor absoluto da diferença entre os dois valores.

Na maioria das implementações deste método, os atributos numéricos são normalizados, para que estes tenham a mesma contribuição que os atributos discretos na predição da classe. Uma possível forma de normalização de um atributo a_i é calcular a média m e o desvio padrão dp de seus valores. O valor normalizado de $v(a_i)$, onde $v(a_i)$ é o valor do atributo a_i em determinada tupla, é calculado através da expressão $\frac{v(a_i)-m}{dp}$ [105].

Quando uma nova tupla t_q é apresentada, o algoritmo percorre as tuplas armazenadas em $\mathcal{R}$ e retorna as k tuplas mais similares à tupla t_q de acordo com a função de distância métrica utilizada (o valor de k é um parâmetro de entrada do algoritmo). Estas tuplas são então utilizadas para se fazer a predição da instância apresentada: a classe de t_q é identificada como sendo a classe predominante dentre as classes das k tuplas mais similares identificadas pelo algoritmo. Por exemplo, para $k = 5$, se 3 das tuplas mais similares têm o valor v para o atributo alvo, então a classe prevista para a tupla t_q é também v, visto que este valor é o predominante dentre as 5 tuplas consideradas na predição.

Qual o valor adequado para k ? O algoritmo 1-NN determina uma tupla que seja mais similar à tupla t_q. Quando esta tupla mais similar é encontrada, assume-se que o valor do atributo alvo é igual ao valor do atributo correspondente na tupla mais similar. Contudo, o algoritmo 1-NN às vezes não tem uma precisão adequada, visto que considera-se somente uma tupla para se fazer a predição. Conforme o

valor de k vai aumentando, a precisão na predição também aumenta, mas é difícil de se determinar o valor k a partir do qual não há mais melhoria apreciável da predição. A determinação do melhor valor de k requer a comparação dos resultados para diferentes valores de k em um conjunto de dados de validação, separado do conjunto de treinamento [75].

3.4.4 Computação Evolucionária

Computação Evolucionária envolve métodos adaptativos e independentes de domínio de aplicação que podem ser utilizados para resolver problemas de busca e otimização. Há duas classes principais de métodos: *Algoritmos Genéticos* e *Programação Genética.*

Algoritmos Genéticos (AGs) são baseados em processos existentes em organismos biológicos. Através de muitas gerações, populações naturais evoluem de acordo com os princípios da seleção natural e sobrevivência do "mais adaptado", primeiramente estudados por Charles Darwin na *Origem das Espécies.* Imitando este processo, algoritmos genéticos podem encontrar soluções para problemas reais, se estes forem adequadamente codificados. As soluções iniciais para um problema são codificadas de alguma forma (por exemplo, em cadeias de *bits*). Após esta codificação, o algoritmo segue iterativamente, selecionando (de acordo com uma medida de adequação) as soluções mais "aptas" a cada iteração. Os indivíduos da população atual são avaliados de acordo com uma função de adequação. Uma nova população é gerada através da seleção aleatória dos indivíduos mais aptos a partir da população atual. Alguns destes indivíduos farão parte da próxima população intactos. Outros indivíduos são utilizados como base para a geração de novos indivíduos através da aplicação de operadores genéticos como mutação e cruzamento. Os princípios básicos deste método foram primeiramente apresentados por Holland [58]. Em [88], é apresentada a implementação de um algoritmo genético hierárquico utilizando a linguagem *RPL2.* Este algoritmo é utilizado para caracterizar o comportamento dos clientes de uma loja de vendas a varejo.

Programação Genética é, na verdade, uma variante do método de AG, onde

os indivíduos da população são programas de computador ao invés de cadeias de *bits*. Estes programas são tipicamente representados por árvores, correspondendo às árvores sintáticas destes programas. Cada chamada de função corresponde a um nó da árvore, e os argumentos da função são dados pelos nós descendentes [75]. Em [45], é discutida a aplicação de programação genética à tarefa de classificação, adaptando-se operadores genéticos para esta tarefa. Em [86] e [87], programação genética é utilizada para, a partir de um conjunto de declarações em uma linguagem de consulta de um SGBD orientado a objetos (*MASSON*), avaliar se tais consultas têm como resultado o mesmo conjunto de objetos especificados.

3.4.5 Redes Neurais

Redes Neurais (RNs) constituem um método proveniente da área de Inteligência Artificial baseado na simulação do funcionamento do cérebro, através de estruturas de dados computacionais [51]. A grande motivação deste método está na nítida superioridade do cérebro quando comparado aos computadores digitais na realização de tarefas que demandem tolerância a falhas, flexibilidade, lógica imprecisa e paralelismo. Para um descrição detalhada das aplicações de redes neurais à mineração de dados, vide [17].

Alguns autores advogam que Redes Neurais não são adequadas para MD. Uma das razões é a de que os padrões extraídos não estão em uma forma inteligível como regras. No entanto, outros trabalhos têm sido propostos no sentido de amenizar ou resolver este problema. Em [27], é apresentada uma introdução ao uso de redes neurais aplicadas à mineração de dados. Em [69], Lu et al. apresentam a utilização de redes neurais para a extração de regras de classificação a partir de uma rede neural treinada.

Capítulo 4

Primitivas para a Tarefa de Classificação

If you wish to make an apple pie from scratch, you must first invent the Universe.

Carl Sagan (1934-1996) Astrônomo norte-americano

4.1 Introdução

Este Capítulo define primitivas de Mineração de Dados a serem utilizadas por aplicações que se destinem a realizar a tarefa de classificação sobre dados armazenados em um SGBD. A motivação para a especificação destas primitivas é que, em grande parte dos algoritmos utilizados para classificação, as informações necessárias para a construção do modelo são resumos estatísticos sobre os dados. Algumas primitivas projetadas com o objetivo de dar suporte à obtenção destes resumos estatísticos têm sido propostas na literatura (vide Capítulo 6 de trabalhos relacionados).

O objetivo deste Capítulo é tentar estender algumas das primitivas definidas em outras trabalhos, de tal forma que as primitivas resultantes desta extensão (1) possam ser utilizadas em um maior número de situações ou (2) sejam mais eficientes que as primitivas anteriores.

A organização deste Capítulo é como segue. Na Seção 4.2, são analisados os requisitos básicos necessários em uma primitiva de MD em geral. Por fim, a Seção 4.3 apresenta a especificação das primitivas de classificação e descreve como estas primitivas se aplicam aos métodos descritos na Seção 3.4.

4.2 Requisitos de uma Primitiva de Mineração de Dados

Em [46], são identificados os requisitos básicos necessários em uma primitiva de Mineração de Dados. São eles:

1. *Uma primitiva deve ter uma especificação precisa.* Isto significa que seus parâmetros de entrada, seu resultado e seu processamento devem ser bem definidos.

2. *Uma primitiva deve ser o mais genérica possível.* Ou seja, a primitiva deve ser construída visando a generalidade, a possibilidade de utilização por um razoável número de algoritmos de MD.

3. *Uma primitiva deve ser computacionalmente significante.* Isto significa que a operação realizada por uma primitiva deve ou ocorrer freqüentemente como parte do algoritmo, ou demandar uma fração considerável do tempo de processamento total. Nas primitivas apresentadas neste Capítulo, um dos objetivos é minimizar o tráfego de dados entre a aplicação de MD e o banco de dados. Este requisito busca melhorar o desempenho destas aplicações, em virtude de as operações de entrada e/ou saída de dados demandarem uma quantidade significativa do tempo de execução, principalmente em arquiteturas do tipo cliente/servidor, uma tendência em sistemas de bancos de dados atuais.

4. *Uma primitiva deve ser orientada a conjunto.* Este requisito diz respeito à necessidade de uma primitiva poder processar uma coleção de tuplas por vez, independentemente da ordem destas tuplas.

Estes requisitos devem ser levados em conta quando da definição de uma primitiva. As primitivas para classificação propostas neste Capítulo satisfazem a este conjunto de requisitos.

4.3 Primitivas para Classificação

Esta Seção descreve a especificação das primitivas de classificação propostas nesta tese. Esta especificação é feita através da definição (1) dos parâmetros de entrada da primitiva, (2) de seu resultado, e (3) da operação realizada pela mesma. Para esta Seção, considere as seguintes presuposições e definições:

- Assume-se que o SGBD utilizado possui uma arquitetura cliente/servidor. Isto significa que a aplicação de MD e o SGBD estão em espaços de endereçamento de memória diferentes (possivelmente, em máquinas diferentes). O servidor é o próprio SGBD e o cliente é o programa onde o algoritmo de mineração de dados está implementado. Este programa é muitas vezes denominado *programa de aplicação*, ou, simplesmente, *aplicação*. Toda a comunicação de dados é feita através de *requisições* pelo cliente ao servidor, como, por exemplo, através de declarações em uma linguagem de consulta.

- Todas as discussões acerca do mapeamento das primitivas para uma linguagem declarativa são feitas levando-se em consideração o padrão SQL atual, conhecido informalmente como *SQL-92* [74]. Em primeiro lugar, porque a linguagem SQL é um padrão indiscutível em SGBDs relacionais. Em segundo lugar, porque, mesmo linguagens de consulta para SGBDs orientados a objetos, como a OQL [20, 22], são bastante semelhantes ao padrão SQL-92.

- Utiliza-se, também, o termo *linguagem SQL-92* para denotar uma linguagem de consulta que esteja em conformidade com o padrão SQL-92.

- $Dom(a_i)$ corresponde ao domínio de valores do atributo previsor a_i (para i no intervalo de inteiros $[1, n]$);

- $Dom(a_c)$ corresponde ao domínio de valores do atributo alvo a_c;
- $Dom(a_i)\chi Dom(a_c)$ corresponde ao produto cartesiano dos conjuntos $Dom(a_i)$ e $Dom(a_c)$;
- Card($\mathcal{C}$) corresponde à cardinalidade da coleção de tuplas $\mathcal{C}$.

4.3.1 A Matriz $\mathcal{M}$

Primeiramente, seja definir a matriz $\mathcal{M}$, cuja estrutura é apresentada na Tabela 4.1. Como mostra esta Tabela, a matriz $\mathcal{M}$ é constituída de n partições, onde cada partição está associada a um atributo previsor a_i. A i-ésima partição de $\mathcal{M}$ é obtida pela aplicação da função $\Pi(a_i)$, assim definida:

Definição 4.1 $\Pi(a_i)$. *A partir do atributo previsor a_i, a função Π constrói uma relação auxiliar na qual cada tupla tem a forma* (id, v_i, v_c, cnt), *onde* (v_i, v_c) ϵ $Dom(a_i)$ χ $Dom(a_c)$, cnt *é o número de co-ocorrências de* (v_i, v_c) *na relação $\mathcal{R}$, e* id *é um identificador único da partição. A relação resultante da aplicação da função Π é obtida retirando-se todas as tuplas da relação auxiliar nas quais o valor de* cnt *seja igual a 0 (zero).*

Tabela 4.1: Estrutura da matriz $\mathcal{M}$.

$\Pi(a_1)$
$\Pi(a_2)$
...
$\Pi(a_i)$
...
$\Pi(a_n)$

Note que toda a informação necessária para a construção da matriz $\mathcal{M}$ através da aplicação da função Π pode ser obtida a partir da relação mina $\mathcal{R}$.

Para exemplificar o processo de construção de $\mathcal{M}$, considere $\mathcal{R}$ como sendo o conjunto de treinamento apresentado na Tabela 3.1. Considere, além disso, que $Dom(OutLook) = \{overcast, rain, sunny\}$ e $Dom(PlayTenis) = \{yes, no\}$. A partição da matriz $\mathcal{M}$ construída a partir da relação $\mathcal{R}$ para o atributo *OutLook* é apresentada na Tabela 4.2. Nesta Tabela, cada linha contém um valor do atributo previsor *OutLook*, um valor do atributo alvo *PlayTenis* e o número de co-ocorrências destes dois valores na relação $\mathcal{R}$. Esta partição contém também uma coluna na qual é armazenado o valor do identificador *id* associado a mesma. De forma análoga, pode-se construir as demais partições de $\mathcal{M}$, correspondentes aos demais atributos previsores.

Tabela 4.2: $\Pi(OutLook)$: Partição Π para o atributo *OutLook*.

id	*OutLook*	*PlayTenis*	*cnt*
0	*overcast*	*yes*	*4*
0	*rain*	*yes*	*3*
0	*rain*	*no*	*2*
0	*sunny*	*yes*	*2*
0	*sunny*	*no*	*3*

Seja, agora, considerar uma maneira de se obter a matriz $\mathcal{M}$ quando $\mathcal{R}$ for uma relação armazenada em um SGBD. Ou seja, uma maneira pela qual um programa de aplicação possa obter $\mathcal{M}$ declarativamente, através de uma requisição enviada ao servidor.

Se $\mathcal{R}$ estiver armazenada em um SGBD, o mesmo resultado obtido pela aplicação da função Π para a construção de uma partição de $\mathcal{M}$, a menos da coluna referente ao identificador, poderia ser obtido através de uma consulta, cuja forma geral é apresentada pela Declaração 4.1.

Para exemplificar, considere, novamente, o atributo *OutLook* e sua partição $\Pi(OutLook)$ exibida na Tabela 4.2. Para a obtenção desta partição, a Declaração 4.2 seria suficiente. Declarações análogas poderiam ser utilizadas para se obter as

```
SELECT a_i AS prev, a_c AS class, COUNT(*)
FROM R
WHERE C
GROUP BY prev, class
```

Declaração 4.1: Mapeamento declarativo da função $\Pi(a_i)$.

partições associadas aos demais atributos previsores, assim obtendo-se a matriz $\mathcal{M}$ por completo.

```
SELECT OutLook AS prev, PlayTenis AS class, COUNT(*)
FROM R
GROUP BY prev, class
```

Declaração 4.2: Obtenção declarativa da partição $\Pi(OutLook)$, sem o identificador.

Na verdade, primitivas para a obtenção do resultado da Declaração 4.1 já foram propostas na literatura (vide Capítulo 6 de trabalhos relacionados). No entanto, uma desvantagem de se obter a matriz $\mathcal{M}$ por esta abordagem é que deve-se enviar n requisições ao servidor, uma para cada atributo previsor. Por exemplo, se a relação mina tem 20 atributos previsores, 20 requisições teriam que ser enviadas ao servidor pelo programa de aplicação, e cada uma delas teria de ser compilada e executada sobre a mesma relação base. Um ganho significativo de desempenho pode ser alcançado se houver a possibilidade de se construir $\mathcal{M}$ a partir do envio de uma única requisição ao servidor. A primitiva α, proposta na próxima Seção, fornece uma solução para este problema.

4.3.2 Obtendo $\mathcal{M}$ diretamente

Esta Seção descreve a Primitiva α, projetada para a obtenção da matriz $\mathcal{M}$ diretamente. Por "obter $\mathcal{M}$ diretamente", denota-se que o objetivo da primitiva aqui especificada é possibilitar a obtenção de $\mathcal{M}$ através de uma única requisição ao servidor.

Considere, primeiramente, uma solução preliminar através da utilização da declaração *UNION ALL*. Em SQL-92, esta declaração tem a seguinte sintaxe [24]:

$< q_1 >$ **UNION** [**ALL**] $< q_2 >$ [**ORDER BY** <sort specification>]

A declaração acima toma duas relações de entrada, correspondentes a q_1 e a q_2, e retorna uma nova relação que corresponde à união de q_1 e q_2, contendo as tuplas de ambas as relação especificadas. As relações especificadas devem ser *compatíveis para união*, o que significa que os atributos correspondentes devem ser de tipos compatíveis. A utilização do modificador *ALL* faz com que tuplas idênticas existentes nas relações de entrada sejam preservadas na relação resultante.

Para construir $\mathcal{M}$ a partir de uma declaração utilizando-se *UNION*, seja a declaração Θ_1 a seguir:

SELECT "1" **AS** id, a_1 **AS** prev, a_c **AS** class, **COUNT**$(*)$
FROM $\mathcal{R}$
WHERE $\mathcal{C}$
GROUP BY id, prev, class

Declaração 4.3: Θ_1 para obtenção declarativa da partição $\Pi(a_1)$ da matriz $\mathcal{M}$.

A Declaração 4.3, correspondente a Θ_1, obtém a partição $\Pi(a_1)$ da matriz $\mathcal{M}$. A primeira expressão da cláusula SELECT, *"1" as id*, significa que a relação resultante desta consulta tem a primeira de suas colunas toda preenchida com o valor 1. Note que o resultado desta declaração é equivalente ao obtido pela Declaração 4.1, a menos do atributo adicional *id*, que em todas as tuplas da relação resultante tem valor igual a 1.

Para a obtenção da matriz $\mathcal{M}$ completa, é necessário que se envie ao servidor uma declaração Θ_u, cuja forma geral é apresentada pela Declaração 4.4. Nesta declaração, n é o número de atributos previsores de $\mathcal{R}$. Note que o resultado produzido em cada declaração Θ_i, $1 < i < n$, corresponde a uma partição da matriz $\mathcal{M}$. O atributo *id* é necessário para se saber os limites de cada partição na relação resultante da

execução de Θ_u, em virtude de as cardinalidades das partições $\Pi(a_j)$ e $\Pi(a_k)$, $j \neq k$, em geral serem diferentes.

Θ_1 **UNION ALL**
Θ_2 **UNION ALL**
...
Θ_{n-1} **UNION ALL** Θ_n

Declaração 4.4: Utilizando UNION ALL para a obtenção de $\mathcal{M}$

Uma restrição associada à solução para a obtenção de $\mathcal{M}$ descrita acima é que o domínio do atributo previsor a_i pode variar de uma partição para outra, o que invalida tal declaração se os atributos previsores não possuírem domínios compatíveis (todos discretos ou todos contínuos). Alguns SGBDs (por exemplo, o *Microsoft Access* [29]) permitem declarações do tipo da Declaração 4.4, mesmo que os atributos previsores tenham domínios diferentes, embora este tipo de construção não faça parte do padrão SQL-92. Uma possível solução para este problema é a discretização prévia dos atributos contínuos de $\mathcal{R}$.

O problema mais sério encontrado na solução acima diz respeito à execução propriamente dita da declaração Θ_u. Na maioria dos SGBDs, a ação que se toma na execução de uma união é a de se executar cada consulta Θ_i separadamente e aglutinar os resultados parciais para formar Θ_u [24]. Isto é indesejável do ponto de vista de tempo de execução. No entanto, o *formato* do resultado de cada consulta Θ_i é muito semelhante e seria desejável que o otimizador de consultas fosse capaz de detectar esta similaridade e de tirar proveito dela para diminuir o tempo de execução de Θ_u. Pode-se, por exemplo, aproveitar o fato de que cada Θ_i é independente um do outro para se construir Θ_u em paralelo, a medida que a relação mina $\mathcal{R}$ é percorrida. Desta forma, $\mathcal{M}$, o resultado de Θ_u, poderia ser construída com uma única passada pelas tuplas de $\mathcal{R}$.

Os problemas levantados acima servem de motivação para a definição da primitiva α. Para isto, propõe-se uma extensão à cláusula *GROUP BY* de uma consulta de seleção. A forma geral da cláusula *GROUP BY* é a seguinte:

GROUP BY a_1, a_2, ..., a_n,

onde a_1, a_2, ..., a_n é uma lista de atributos da relação base. A cláusula *GROUP BY* particiona a relação base em conjuntos disjuntos de tuplas e, posteriormente, faz agregação de valores em cada um destes conjuntos, de acordo com a função de agregação especificada na cláusula *SELECT*. A Declaração 4.2 serve como exemplo de utilização desta cláusula juntamente com a função de agregação *COUNT*.

Note que cada elemento desta lista corresponde a *um e somente um* atributo da relação base. A extensão à cláusula *GROUP BY* aqui proposta relaxa esta restrição, permitindo que um elemento da lista possa corresponder a *mais de um* atributo da relação. Nesta extensão, cada elemento a_i da lista associada à cláusula *GROUP BY* pode ter a sintaxe a seguir:

av **IN** $\{a_1, a_2, ..., a_n\}$

A semântica desta extensão consiste em considerar os atributos em $\{a_1, a_2, ..., a_n\}$ como um só atributo virtual *av*. Este atributo virtual tem como domínio o conjunto união dos conjuntos $Dom(a_1)$, $Dom(a_2)$, ..., $Dom(a_n)$. De posse desta extensão à cláusula *GROUP BY*, pode-se agora apresentar a primitiva α. A forma geral desta primitiva é exibida na Declaração 4.5.

SELECT av, a_c **AS** class, **COUNT**(*)
FROM $\mathcal{R}$
WHERE $\mathcal{C}$
GROUP BY av **IN** $\{a_1, a_2, ..., a_n\}$, class

Declaração 4.5: Mapeamento Declarativo da Primitiva α

A primitiva α realiza a seguinte operação: São selecionadas, a partir de $\mathcal{R}$, todas as tuplas que satisfaçam às condições expressas em $\mathcal{C}$, formando-se, assim uma relação temporária $\mathcal{R}_t$. Considerando-se $\mathcal{R}_t$, para cada atributo a_i especificado na lista a_1, a_2, ..., a_n, a partição $\Pi(a_i)$ é gerada; a relação $\mathcal{R}_\alpha$ resultante é constituída da união de todas as partições $\Pi(a_i)$ assim formadas. A cada tupla (v_{ij}, v_c, *cnt*)

de $\Pi(a_i)$ é adicionado um novo atributo *id*, onde $Dom(id) = \mathcal{Z}^*$. Os valores deste atributo, constantes dentro de cada partição e únicos para cada uma delas, são gerados automaticamente.

A função do atributo *id* é a de determinar unicamente cada partição $\Pi(a_i)$ da matriz $\mathcal{M}$, em virtude de que esta última é retornada como uma relação $\mathcal{R}_\alpha$, na qual o domínio de v_{ij} é o conjunto união dos domínios de todos os atributos previsores. Como a primitiva α associa um único valor de *id* a cada partição $\Pi(a_i)$, este valor pode ser utilizado para identificar a qual atributo previsor tal partição está associada.

Note que a sintaxe geral da Primitiva α é bem menos complexa do que a de Θ_u (vide Declaração 4.4), embora ambas produzam resultados equivalentes. Esta vantagem é particularmente relevante quando o número de atributos previsores é bastante grande.

Outra vantagem da Primitiva α em comparação com a Declaração Θ_u é que, utilizando-se a Primitiva α, pode-se construir a matriz $\mathcal{M}$ com uma única passada pela relação $\mathcal{R}$. Além disso, a execução da primitiva α possui um potencial grande para paralelização, visto que as partições $\Pi(a_i)$ podem ser construídas independentemente (vide Capítulo 5).

Uma restrição da Primitiva α é a de que, assim como em Θ_u, esta somente pode ser usada quando os atributos previsores possuírem domínios compatíveis, o que parece inviabilizar seu uso prático. No entanto, pode-se contornar este problema de duas maneiras diferentes:

1. A propriedade da matriz $\mathcal{M}$ de ser constituída de partições independentes pode ajudar a resolver esta restrição. Em uma tarefa de classificação que contenha tanto atributos discretos como atributos contínuos, pode-se executar a Declaração 4.5 duas vezes, uma para obter as partições referentes aos atributos contínuos e outra para obter as partições para atributos discretos. Note que, no caso de atributos contínuos, é mais provável que o resultado da Primitiva α tenha uma cardinalidade maior, podendo, no pior caso, ter cardinalidade igual a da relação $\mathcal{R}$.

2. Alternativamente, como uma das ações da fase de pré-processamento do processo de MD, pode-se realizar uma discretização sobre os atributos contínuos da relação $\mathcal{R}$. Isto tornaria todos os atributos discretos, o que permitiria a obtenção de $\mathcal{M}$ com uma única requisição ao servidor através da Primitiva α. Esta solução pode parecer imprópria, em virtude de que há perda de informação no conjunto de dados utilizado (na discretização, um número de valores contínuos é associado a um número bem menor de valores discretos). No entanto, Cattlet [19] discute diversos métodos de discretização e conclui que, em alguns casos, uma redução substancial no tempo de aprendizado é alcançada sem perda na precisão do modelo gerado.

Esta Seção apresentou a Primitiva α sem nenhuma menção a como esta primitiva pode ser aplicada na implementação de algum método de classificação. As Seções 4.3.3 e 4.3.4 descrevem como esta primitiva pode dar suporte aos métodos de Indução de Regras e de Aprendizado Bayesiano Simples.

4.3.3 Primitiva α: Indução de Regras

Existem diversos algoritmos descritos na literatura para Indução de Regras. No entanto, uma característica que é comum à grande maioria destes algoritmos é a de que eles podem gerar uma coleção de regras ou uma árvore de decisão única e exclusivamente a partir de informações estatísticas sobre a distribuição dos valores dos atributos nas tuplas de $\mathcal{R}$.

Suponha que todas estas informações estatísticas sejam calculáveis a partir da matriz $\mathcal{M}$. Neste caso, ao invés de percorrer todas as tuplas da relação original $\mathcal{R}$, algoritmos de IR podem ser modificados de tal forma a utilizarem a Primitiva α para a obtenção de $\mathcal{M}$ e, a partir desta, decidirem como gerar partições a partir das tuplas de $\mathcal{R}$ para a formação da árvore de decisão e posterior geração de regras. Com o objetivo de atestar esta suposição, esta Seção demonstra como algumas das operações para a IR podem ser realizadas através da Primitiva α.

Considere, primeiramente, a questão da seleção do atributo previsor para a

criação de um dos nós da árvore de decisão. Como visto na Seção 3.4.1, na criação de um nó da árvore, deve-se selecionar o atributo mais adequado, escolhido como aquele que melhor divide o conjunto de treinamento, segundo a *função de avaliação* utilizada. Nesta mesma Seção são descritos três exemplos de funções de avaliação: *ganho de informação* (Equação 3.2), *taxa de ganho de informação* (Equação 3.3) e *índice gini* (Equação 3.6). A seguir é descrito como o cálculo dessas funções de avaliação pode ser feito utilizando-se somente informações contidas em $\mathcal{M}$.

Examinando-se as equações 3.2, 3.3 e 3.6, pode-se notar que os valores necessários para o cálculo destas Equações são p_i, $|\mathcal{C}|$, $|\mathcal{C}_v|$. Portanto, uma vez calculados estes valores, pode-se calcular qualquer uma das funções acima.

- p_i: para o cálculo de p_i, basta escolher qualquer uma (a primeira, por exemplo) das partições de $\mathcal{M}$ e percorrê-la, acumulando o valor do atributo *cnt* cada vez que o atributo alvo pertença à classe i. Seja c_i este valor acumulado. Então $p_i = \frac{c_i}{|\mathcal{C}|}$.

- $|\mathcal{C}|$: para o cálculo de $|\mathcal{C}|$, basta acumular o valor do atributo *cnt* para todas as tuplas da partição correspondente ao atributo previsor sendo analisado.

- $|\mathcal{C}_v|$: para calcular este valor, basta, dentro da partição do atributo previsor sendo analisado, acumular o valor do atributo *cnt* para todas as tuplas nas quais o atributo previsor assume o valor v.

Note que, na construção da declaração da primitiva α para a obtenção dos dados necessários à determinação do atributo previsor mais adequado a constituir o teste de um nó da árvore, deve-se considerar o nível da árvore gerada até o momento. Por exemplo, considere que já foram escolhidos m atributos em um ramo da árvore. Neste caso, no parâmetro $\mathcal{C}$ da cláusula WHERE da primitiva α (vide Declaração 4.5) deve ser especificada uma conjunção de m predicados, sendo que cada um destes predicados corresponde ao teste feito em um dos nós ancestrais do nó a ser criado. Isto garante que somente os dados referentes àquela sub-árvore estão sendo considerados na seleção do atributo previsor mais adequado.

A análise acima mostra como a Primitiva α pode ser utilizada para dar suporte à seleção de atributos em algoritmos de IR. Na verdade, várias outras funções de avaliação existentes na literatura podem ser calculadas através do uso da Primitiva α. A Tabela 4.3 é uma lista de outras medidas que podem ser calculadas pela Primitiva α. Esta tabela é uma adaptação de duas outras tabelas, encontradas uma em [46], outra em [65]. Como descrito no Capítulo 6, a Primitiva α aqui proposta é uma extensão das primitivas já propostas em outros trabalhos. Portanto, todos os algoritmos implementáveis através das primitivas de [46] e [65] podem ser implementados através da Primitiva α.

Tabela 4.3: Algumas medidas às quais a Primitiva α provê suporte para o cálculo.

Coeficiente de Cramer
Erro de ressubstituição
Ganho de Informação
Índice Gini
Medida de Associação Tau
Medida J
Ortogonalidade entre vetores de classe
Chi-quadrado
Taxa de Ganho de Informação

Além da seleção de atributos, outra questão que diz respeito à utilização da primitiva α em algoritmos de IR é a de poda da árvore (vide Seção 3.4.1). Em [44], Freitas descreve como o método de pós-poda pode ser implementado através da utilização de uma variante da primitiva *Count By Group* (vide Seção 6.3.3 para maiores detalhes acerca dessa primitiva), através da seguinte declaração:

A Declaração 4.6 fornece informação suficiente para a determinação de se a condição $a_i = v_{ij}$ deve ser removida de uma determinada regra. (O leitor que desejar maiores detalhes deve se direcionar à referência [44].) Embora a primitiva α seja uma extensão à primitiva *Count By Group*, sua utilização aqui não é tão vantajosa

SELECT a_c **AS** class, **COUNT**(*)
FROM $\mathcal{R}$
WHERE $\mathcal{C}$ **AND** (a_i <> v_{ij})
GROUP BY class

Declaração 4.6: Variante da primitiva *Count By Group* para remoção de condições em regras de produção.

quanto na fase expansão da árvore. Isto porque, na fase de poda, a conjunção de atributos especificada na cláusula WHERE varia de atributo para atributo (repare o predicado adicional a_i <> v_{ij} na Declaração 4.6).

4.3.4 Primitiva α: Aprendizado Bayesiano Simples

Esta Seção demonstra como a Primitiva α também pode ser utilizada para dar suporte à construção de algoritmos que implementam o método de Aprendizado Bayesiano Simples (ABS). Conforme descrito na Seção 3.4.2, o método de ABS prediz a classe da tupla apresentada a partir de probabilidades anteriores e condicionais calculadas a partir do conjunto de treinamento. Ou seja, para cada atributo previsor a_i, deve-se calcular a probabilidade condicional $P(a_i = v(a_i)|a_c = v_j)$, onde $v_j \in Dom(a_c)$. Adicionalmente, as probabilidades $P(v_j)$ devem também ser calculadas para cada valor v_j do atributo alvo. De posse destes valores, é possível determinar o valor v^*, correspondente à classe da tupla apresentada.

Para ilustrar como as probabilidades anteriores e condicionais podem ser calculadas a partir da relação $\mathcal{R}_\alpha$, considere novamente a Tabela 4.2 correspondente à partição Π(OutLook). Considere, ainda, que a tupla cuja classe deve ser prevista é a mesma apresentada no exemplo da Seção 3.4.2, (*sunny, cool, high, strong, ?*).

Para calcular as probabilidades $P(v_j)$, $v_j \in Dom(PlayTenis)$, defina $n(v_j)$ como o somatório dos valores do atributo *cnt* de $\Pi(a_i)$ considerando-se as tuplas onde o valor do atributo alvo *PlayTenis* é igual a v_j. Desta forma, obtêm-se $n(yes) = 9$ e $n(no) = 5$ para a Tabela 4.2.[1] Com estes valores pode-se calcular $P(v_j)$ como segue:

[1]Observe (1) que $n(v_j)$, $\forall v_j$, pode ser obtido percorrendo-se a partição $\Pi(a_i)$ somente uma vez

$$P(v_j) = \frac{n(v_j)}{\sum_k n(v_k)} \tag{4.1}$$

Para calcular as probabilidades $P(sunny|v_j)$, defina, $n(v(a_i), v_j)$ como a soma dos valores de *cnt* em $\Pi(a_i)$ considerando-se a tuplas desta partição nas quais a_i tem valor $v(a_i)$ e a_c tem valor v_j. A partir desta definição, obtêm-se $n(sunny, yes) = 2$ e $n(sunny, no) = 3$. De posse destes valores, pode-se calcular os valores $P(sunny|v_j)$ através da Equação a seguir:

$$P(sunny|v_j) = \frac{n(sunny, v_j)}{n(v_j)} \tag{4.2}$$

Analogamente, pode-se calcular as probabilidades para os demais atributos previsores, desta forma obtendo-se todos os valores necessários à aplicação da Equação 3.8, que prediz a classe da tupla apresentada. Este exemplo demostra que a Primitiva α pode dar suporte ao cálculo de todas a probabilidades utilizadas no método de ABS, quando os atributos previsores são todos discretos.

No caso de haver atributos previsores contínuos em $\mathcal{R}$, uma solução seria a discretização destes atributos. Esta solução tem a vantagem de eliminar a restrição de distribuição normal sobre atributos contínuos no método de ABS (vide Seção 3.4.2).

Outra solução é calcular a *média* e a *variância* (necessárias para o cálculo da probabilidade condicional) dos valores de um atributo contínuo de uma forma declarativa. Celko [24] descreve várias maneiras de se calcular estes valores através de declarações em SQL-92.

4.3.5 Primitiva β: Aprendizado Baseado em Instâncias

Esta Seção define outra primitiva, denominada *Primitiva* β, que tem o objetivo de dar suporte ao terceiro método de classificação considerado neste Capítulo, o de Aprendizado Baseado em Instâncias (Seção 3.4.3). Para simplificar a apresen-

e (2) que este valor é constante para todas a partições da matriz $\mathcal{M}$.

tação, considera-se que os pesos, se existirem, são calculados em uma fase de pré-processamento da relação $\mathcal{R}$ e armazenados como atributos desta relação.

Computacionalmente, o método de ABI não envolve qualquer esforço de aprendizagem a partir de exemplos em virtude de o modelo ser a própria relação $\mathcal{R}$ (vide Seção 3.4.3). No entanto a fase de predição da classe de uma nova tupla é bastante cara do ponto de vista computacional, visto que a nova tupla deve ser comparada a todas as outras tuplas de $\mathcal{R}$ [1].[2] Em virtude disto é que a primitiva aqui apresentada se aplica à fase de predição.

Como motivação para a definição da Primitiva β, considere o algoritmo k-NN para $k = k_v$. Nesta situação, as k_v tuplas mais similares à tupla t_q devem ser obtidas a partir da relação $\mathcal{R}$. Seja $\Delta(t_r, t_q)$ a distância (segundo alguma métrica) entre as tuplas t_q e t_r. Para resolver este problema de uma forma declarativa, propõe-se a primitiva β a seguir:

SELECT $\Delta(t_r, t_q)$ **AS** dist, [**COUNT**(*) **AS** cnt,] a_c **AS** class
FROM $\mathcal{R}$
[**GROUP BY** dist, class]
ORDER BY dist

Declaração 4.7: Mapeamento Declarativo da Primitiva β

Esta primitiva retorna uma relação $\mathcal{R}_\beta$ de tuplas da forma (*dist*, *cnt*, *class*), onde, em cada tupla t desta relação, *cnt* corresponde ao número de tuplas em $\mathcal{R}$ cuja classe é *class* que estão à distância *dist* de t_q. A tuplas de $\mathcal{R}_\beta$ são ordenadas pelo valor de *dist*.

Já que, em última análise, as tuplas da relação retornada são percorridas uma a uma pela aplicação, esta poderia considerar somente as k_v primeiras tuplas para fazer a predição da classe de t_q, onde k_v é o número de tuplas a serem consideradas pelo algoritmo para a determinação da classe de t_q.

[2]Isto é exatamente o inverso do que acontece com os outros métodos de classificação, nos quais a fase de aprendizado demanda um tempo de computação maior e a fase de predição geralmente envolve uma simples comparação [106].

Note que as construções associadas à função de agregação COUNT da cláusula SELECT e à cláusula GROUP BY da Declaração 4.7 estão entre colchetes, o que indica sua opcionalidade. De fato, se a função de distância for muito complexa e/ou houver diversos atributos contínuos em $\mathcal{R}$, estas construções não são de muita valia visto que os valores de *cnt* serão predominantemente iguais a 1. (Estas construções podem até ser indesejáveis do ponto de visto computacional, pelo fato de implicarem na contagem e agrupamento das tuplas.) Além disso, assumindo que k_v é um número pequeno, o cliente (em vez do servidor) pode ser usado para determinar o valor de *cnt* associado a um grupo de tuplas. Isto tende a ser mais barato computacionalmente do que usar o servidor para agrupar (GROUP BY) e agregar (COUNT) todas as tuplas de R_β. No caso da não utilização da cláusula GROUP BY e da função de agregação COUNT na Declaração 4.7, as tuplas de $\mathcal{R}_\beta$ tomam a forma (*dist*, *class*).

Note também, que, ao contrário da Primitiva α, a Primitiva β não constitui uma extensão sintática à linguagem SQL-92. Seu mapeamento para esta linguagem pode ser inteiramente feito com recursos de sintaxe existentes atualmente. No entanto, um complicador que pode existir neste mapeamento diz respeito à expressão δ_i (vide Equação 3.14), necessária para o cálculo de $\Delta(t_r, t_q)$, em virtude de esta expressão envolver uma condição. Para contornar este problema, adota-se aqui a mesma solução utilizada por Freitas em [44]: Assume-se que os valores dos atributos previsores estejam armazenados no banco de dados como números inteiros. Isto pode ser feito na fase de pré-processamento dos dados. Por exemplo, para o atributo *Sex* da Tabela 4.4, os valores *female* e *male* são convertidos para os valores numéricos 1 e 2, respectivamente. Assim, a expressão δ_i poderia ser calculada da seguinte forma:

$$\delta_i = \mathbf{CEILING}(\mathbf{ABS}(t_r^i - t_q^i)/c) \tag{4.3}$$

Na Equação 4.3, c é uma constante igual ou maior que a cardinalidade do domínio do i-ésimo atributo de $\mathcal{R}$. A função *CEILING(x)* retorna o menor número inteiro que é igual ou maior que o parâmetro x. Sendo assim, conforme os valores t_r^i e t_q^i sejam iguais ou diferentes, *CEILING(ABS(t_r^i - t_q^i)/c)* retorna ou 0 ou 1.

Para ilustrar a utilização da Primitiva β, considere o conjunto de treinamento

Tabela 4.4: Fragmento do conjunto de treinamento *Adults* do repositório UCI.

Age	*Sex*	*Education*	*Income*
53	*female*	*preschool*	$L50K$
17	*female*	*10th*	$G50K$
40	*male*	*7th-8th*	$G50K$
54	*male*	*7th-8th*	$G50K$
28	*male*	*7th-8th*	$L50K$
45	*female*	*7th-8th*	$L50K$
60	*male*	*10th*	$G50K$
60	*female*	*7th-8th*	$L50K$
44	*male*	*7th-8th*	$L50K$
51	*female*	*7th-8th*	$L50K$
52	*male*	*10th*	$G50K$
60	*female*	*7th-8th*	$L50K$
17	*female*	*10th*	$L50K$
17	*female*	*10th*	$L50K$

da Tabela 4.4.[3] Considere, ainda, que a tupla t_q cuja classe deve ser determinada é (*17, female, 10th, ?*). Como uma fase de pré-processamento, os domínios {*female, male*} e {*preschool, 7th-8th, 10th*} são convertidos para {*1,2*} e {*1,2,3*}, respectivamente. O domínio do atributo *Age* não sofre modificação em virtude de ser um domínio numérico. A Declaração 4.8 apresenta uma instanciação da Primitiva β para este caso, onde é utilizada a métrica absoluta (vide Seção 3.4.3) para o cálculo de $\Delta(t_r, t_q)$, e um valor para a constante c igual a 10.

A Tabela 4.5 exibe o resultado da Declaração 4.8. De posse desta Tabela, o

[3]Esta tabela corresponde a um fragmento do conjunto de treinamento *Adults*, do repositório UCI [78]. O conjunto original tem mais atributos e muitos mais tuplas do que o apresentado no fragmento da Tabela 4.4. No entanto, o fragmento aqui apresentado é suficiente para a explanação do mecanismo de funcionamento da primitiva.

```
SELECT (ABS(age - 17) +
        CEILING(ABS(sex - 1) / 10) +
        CEILING(ABS(education - 3) / 10)) AS dist,
        COUNT(*) AS cnt,
        income AS class
FROM R
GROUP BY dist, class
ORDER BY dist
```

Declaração 4.8: Exemplo de utilização da Primitiva β.

programa de aplicação pode determinar a classe da tupla t_q. Considere, por exemplo, que a aplicação esteja implementando o método k-NN para $k = 3$. Como o resultado está ordenado pelos valores de distância até a tupla t_q, a aplicação considera as duas primeiras tuplas da Tabela 4.5. Estas duas tuplas identificam a classe de t_q como sendo igual a $L50K$, em virtude de esta ser a classe predominante dentre as tuplas relevantes para $k = 3$ (2 tuplas com classe igual a $L50K$ e uma tupla com classe igual a $G50K$).

Tabela 4.5: Exemplo da relação $\mathcal{R}_\beta$, resultante da Primitiva β.

dist	*cnt*	*class*
0	*2*	$L50K$
0	*1*	$G50K$
13	*1*	$L50K$
25	*1*	$G50K$
29	*2*	$L50K$
35	*1*	$L50K$
36	*1*	$G50K$
37	*1*	$L50K$
39	*1*	$G50K$
44	*2*	$L50K$
44	*1*	$G50K$

Capítulo 5

Aspectos de Implementação e Validação das Primitivas

Eu medi os céus, agora as sombras eu meço. Para o firmamento viaja a mente, na terra descansa o corpo.

Epitáfio (auto-escrito) de Johannes Kepler (1571-1630)

5.1 Introdução

Este Capítulo apresenta aspectos de implementação e validação das primitivas definidas no Capítulo 4, através da avaliação de seu desempenho em conjuntos de dados sintéticos e naturais, e da descrição de sua construção utilizando-se os sistemas GOA++ e PVM. Na Seção 5.2, é descrito o sistema GOA++. Na Seção 5.3, é feita a descrição do sistema PVM. Finalmente, na Seção 5.4, é feita a descrição dos experimentos realizados para a validação de ambas as primitivas propostas nesta tese, e são feitas comparações com primitivas propostas em outros trabalhos da literatura.

5.2 O Sistema GOA++

O Sistema GOA++ [30, 71, 73, 77] é um protótipo de SGBD orientado a objetos em desenvolvimento pela equipe de pesquisa em Bancos de Dados do COPPE/UFRJ. Este sistema ainda não possui todas as características de um SGBD completo (não possui mecanismos de controle de transações e recuperação de falhas, por exemplo). No entanto, o fato da disponibilidade do código fonte (em linguagem C++) viabilizaram a implementação e validação das primitivas de MD propostas neste trabalho de tese.

O primeiro passo na implementação das primitivas no GOA++ foi a adequação deste sistema ao padrão ODMG. Para isso, o autor desta tese implementou, juntamente com outros integrantes da equipe de desenvolvimento do GOA++, um compilador para a linguagem de definição de objetos deste padrão, ODL, e um compilador para a sua linguagem de consulta, OQL. Basicamente, para a construção destas funcionalidades, foi implementada uma camada sobre as operações básicas de armazenamento e processamento de objetos já existentes na versão anterior do GOA++. Atualmente, o GOA++ possui um modelo de dados em conformidade com o padrão ODMG e as linguagens ODL e OQL já implementadas, além de uma interface para a linguagem C++.

O padrão ODMG seguido pelo GOA++ foi definido por uma organização de mesmo nome com o objetivo de assegurar a portabilidade de aplicações através de diferentes SGBDs orientados a objetos, reduzindo a dependência destas aplicações a um simples produto. Os componentes funcionais do padrão ODMG incluem um *modelo de objetos*, uma linguagem de definição de objetos, uma linguagem de consulta a objetos e mecanismos de acoplamento a linguagens de programação.

A Figura 5.1 ilustra a arquitetura comum a ser utilizada no desenvolvimento de aplicações de bancos de dados, segundo o padrão ODMG. O desenvolvedor da aplicação escreve declarações para o esquema da aplicação (dados e interface) além de um programa fonte para a implementação da aplicação. Este programa fonte é escrito em uma linguagem de programação como C++ e pode conter chamadas às

funções da interface de programação da aplicação, que contém funções para manipulação do bancos de dados, incluindo funcionalidade de consulta declarativa a objetos. As declarações do esquema são escritas em linguagem ODL. As declarações e o programa fonte são então compilados e ligados às funções da biblioteca de interface para produzir o executável da aplicação.

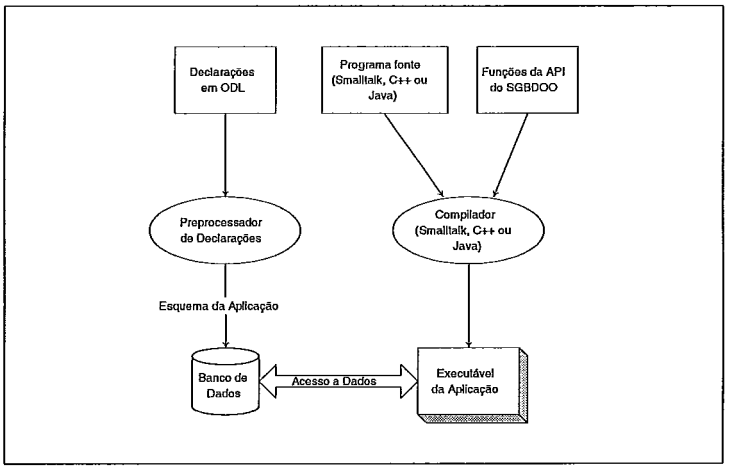

Figura 5.1: Arquitetura do padrão ODMG, adotada pelo sistema GOA++.

5.2.1 O Modelo de Dados

No modelo de dados do GOA++ cada objeto tem um identificador único no banco de dados. Objetos podem ser categorizados em *tipos*. Tipos podem ser atômicos (inteiro, real, caracter, etc.) ou estruturados (classes, estruturas, etc.). O modelo permite a declaração de extensões para uma classe de objetos. Extensões, também denominadas de objetos compostos, são coleções que armazenam objetos de uma mesma classe (ou de subclasses desta). Diversos tipos de coleções são definidos no modelo: conjuntos (*set*), bolsas (*bag*), listas e cadeias (*array*). Instâncias nomeadas destas coleções podem ser usadas para agrupar objetos.

Para os objetivos desta tese, é necessário que se possa modelar a relação mina em termos dos tipos disponíveis no GOA++. Para isso, faz-se a seguinte definição:

Definição 5.1 $\mathcal{R}$-coleção. *Seja t um tipo estruturado da forma* (a_1: t_1, a_2: t_2,..., a_n: t_n), *onde* $t_1, t_2, ..., t_n$ *são tipos atômicos. Uma* $\mathcal{R}$*-coleção é um objeto composto do tipo* set, *onde cada item desta coleção é um objeto do tipo t.*

Repare que uma $\mathcal{R}$-coleção, como definido acima, é equivalente à relação $\mathcal{R}$ definida no Capítulo 3. Os dados utilizados para a validação das primitivas α e β no GOA++ estão armazenados em $\mathcal{R}$-coleções.

5.2.2 Linguagem de Definição de Objetos

A linguagem de definição de dados do GOA++ é a ODL (Object Definition Language). A linguagem ODL fornece a possibilidade de definição do esquema de um banco de dados. A ODL dá suporte aos elementos básicos do modelo de objetos tais como abstração, encapsulamento, modularidade, hierarquia, tipagem e persistência [15].

Uma definição de esquema em ODL é uma camada de abstração independente tanto da linguagem de programação da aplicação a ser utilizada pela aplicação quanto dos SGBDs compatíveis com o padrão ODMG. Através da ODL, as interfaces das classes que modelam o domínio da aplicação são especificadas.

Considere, para fins de melhor entendimento dos conceitos, o exemplo de definição de classe em linguagem ODL a seguir:

```
class S15
extent Tuples
{
        attribute short  a0;
        attribute short  a1;
        attribute short  a2;
        attribute short  a3;
        attribute short  a4;
        attribute short  a5;
        attribute short  a6;
        attribute short  a7;
        attribute short  a8;
        attribute short  a9;
        attribute short  a10;
        attribute short  a11;
        attribute short  a12;
        attribute short  a13;
        attribute short  ac;
};
```

Figura 5.2: Exemplo de definição de classe em ODL.

A Figura 5.2 é um exemplo de $\mathcal{R}$-coleção, no qual a classe *S15* possui 15 atributos do tipo *short* (tipo predefinido do modelo de dados ODMG). Esta classe possui a extensão *Tuples*, coleção onde são armazenados os objetos criados desta classe.

5.2.3 Linguagem de Consulta a Objetos

A linguagem declarativa de consulta a objetos construída para dar suporte ao modelo de dados do GOA++ é a *OQL* (Object Query Language). A linguagem OQL é muito semelhante à linguagem SQL 92. Na verdade, como mencionado por Cattell em [20], a linguagem OQL pode ser vista como o resultado de extensões a esta última. Estas extensões envolvem conceitos de orientação a objetos, como objetos complexos, identidade de objetos, expressões de caminho, polimorfismo, invocação de uma operação, etc. Qualquer expressão de consulta em linguagem OQL tem um resultado cujo tipo pertence ao conjunto de tipos do modelo ODMG.

O mapeamento, em linguagem OQL, das primitivas definidas no Capítulo 4 é apresentado pelas Declarações 5.1 e 5.2 a seguir (Compare com Declarações 4.5 e 4.7, respectivamente.):

SELECT alpha(prev, class: u.a_c, **COUNT**(*): cnt)
FROM u **IN** $\mathcal{R}$
WHERE $\mathcal{C}$
GROUP BY prev **IN** {u.a_1, u.a_2, ..., u.a_n}, class

Declaração 5.1: Mapeamento da Primitiva α para OQL

SELECT beta($\Delta(t_r, t_q)$: dist, [**COUNT**(*): cnt,] class: u.a_c)
FROM u **IN** $\mathcal{R}$
[**GROUP BY** dist, class]
ORDER BY dist

Declaração 5.2: Mapeamento da Primitiva β em OQL

5.3 O Sistema PVM

O sistema PVM (acrônimo para *Parallel Virtual Machine*) [47] fornece uma maneira através da qual programas paralelos podem ser desenvolvidos de uma maneira

eficiente e simples utilizando-se de *hardware* não especializado, ou seja, de equipamentos que não sejam especializados para a execução de programas paralelos. O PVM possibilita que uma coleção de máquinas heterogêneas seja vista como uma única máquina paralela virtual, manipulando de modo transparente ao programa de aplicação todas as mensagens de roteamento, conversão de dados e escalonamento de tarefas através de uma rede de diferentes arquiteturas de computadores.

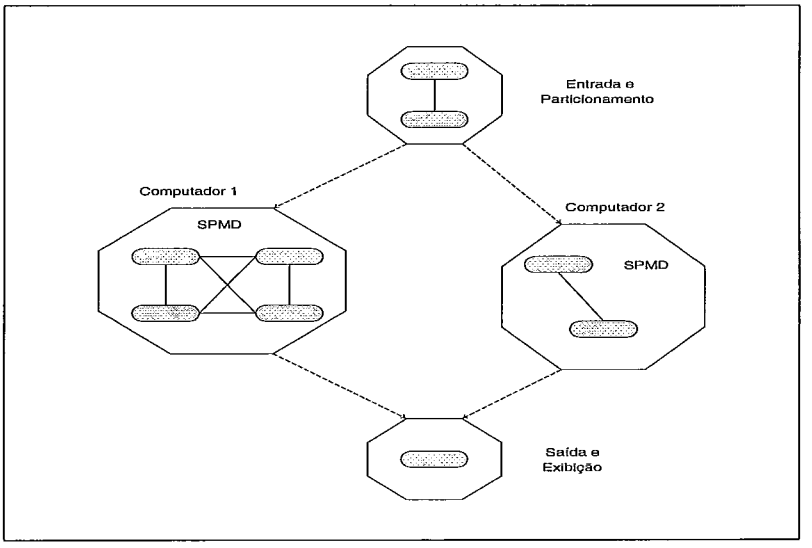

Figura 5.3: Modelo computacional do PVM.

O modelo computacional do sistema PVM (vide Figura 5.3, adaptada de [47]) se baseia na noção de que um programa de aplicação é constituído de várias tarefas, cada uma sendo responsável por uma parte do trabalho computacional a ser realizado pela aplicação. Tarefas são a unidade de paralelismo no PVM e estão freqüentemente associadas a um processo do sistema operacional. Uma tarefa é uma seqüência independente de controle que alterna entre computação e comunicação com as outras tarefas da máquina virtual. As tarefas são executadas em um conjunto de máquinas selecionadas pelo usuário para uma determinada rodada do sistema PVM. Para implementar um programa que se utilize da máquina virtual paralela, o desenvolvedor escreve seu programa de aplicação como uma coleção de tarefas cooperativas.

As tarefas de um programa de aplicação obtêm acesso aos recursos através de uma biblioteca de rotinas de interfaceamento. Estas rotinas também permitem a

inicialização e terminação de tarefas através da rede de computadores componentes da máquina virtual, assim como a sincronização e comunicação entre tarefas. As rotinas de troca de mensagens do PVM são orientadas para operações heterogêneas, envolvendo funcionalidades para *buferização* e transmissão.

Há dois principais níveis de paralelismo que podem ser utilizados em aplicações [60]. No primeiro, uma aplicação é paralelizada ao longo de suas funções; isto é, cada tarefa realiza uma função diferente, por exemplo, entrada, inicialização, solução do problema, saída e exibição. Este nível é freqüentemente chamado de *paralelismo funcional.* Um nível de paralelismo mais comumente utilizado na prática é denominado *paralelismo de dados*, no qual o mesmo conjunto de operações (tarefas) manipula e processa uma parte dos dados. Este nível é o utilizado nos experimentos descritos neste Capítulo, embora o sistema PVM dê suporte aos dois níveis de paralelismo.

5.4 Descrição dos Experimentos

Esta Seção é dividida em duas partes. Na primeira parte, são descritos os experimentos realizados para a validação da primitiva α. São também descritos detalhes de implementação desta primitiva, visto que seu mapeamento declarativo se trata de uma extensão à linguagem de consulta do GOA++.

A segunda parte descreve os experimentos realizados com a primitiva β. Em cada uma destas partes, são feitas comparações das primitivas α e β em relação a outras primitivas definidas em outros trabalhos. Todos os resultados reportados nos experimentos descritos a seguir foram obtidos retirando-se a média sobre (20) vinte repetições da mesma execução.

5.4.1 Validação da Primitiva α

Como a primitiva α tem um potencial grande para ser executada em paralelo, os experimentos realizados procuraram validar esta característica. Com este objetivo, foi utilizado o PVM para a simulação de um ambiente de processamento paralelo de consultas. As máquinas utilizadas como nós da máquina virtual paralela foram

estações de trabalho *SUN* rodando o sistema operacional *Solaris* versão 4.X.

Antes de passar à descrição dos experimentos realizados, seja descrever o mecanismo de distribuição da carga de processamento da primitiva α por entre os nós (unidades de processamento).

Quando o servidor GOA++ recebe uma requisição para a execução de uma declaração da primitiva α, o número de partições a ser gerado é determinado a partir da expressão de consulta recebida. Este número de partições eqüivale ao número de atributos previsores especificado na cláusula *GROUP BY ... IN ...* (vide Seção 4.3.2). De posse deste valor, e do número de nós disponíveis na máquina virtual, determina-se a quantidade de partições que cada nó deve gerar. As requisições para geração de partições são então enviadas a cada nó.

A geração das partições $\Pi(a_i)$ é distribuída por entre os nós da máquina virtual paralela utilizando-se a seguinte estratégia (considera-se que o número de partições a serem geradas é sempre maior que o número de nós disponíveis na máquina virtual paralela): se o número de partições n_Π a serem geradas é divisível pelo número de nós n_p da máquina virtual paralela, cada nó recebe uma requisição para gerar n_Π/n_p partições. Caso contrário, $n_p - r$ nós recebem a requisição para gerarem um número de partições igual a $\lfloor\, n_\Pi/n_p \,\rfloor$, e r nós recebem requisição para gerarem $\lfloor\, n_\Pi/n_p \,\rfloor + 1$ partições, onde r é o resto da divisão de n_Π por n_p. Após o envio das requisições, o servidor entra em estado de espera, aguardando pelos resultados gerados em cada nó da máquina virtual.

Após ter recebido o resultado do último nó, é feita a "montagem" das partições recebidas para a criação da relação $\mathcal{R}_\alpha$ resultante da primitiva α. Note que, em cada nó da máquina virtual permanente, existe uma réplica da $\mathcal{R}$-coleção. Note também que, quando os nós da máquina virtual paralela têm velocidades diferentes, a obtenção de $\mathcal{R}_\alpha$ se torna dependente do tempo de resposta do nó mais lento.

Todo este processo de requisição de geração de partições aos nós para a construção de $\mathcal{R}_\alpha$ é ilustrado pelo algoritmo a seguir, escrito em pseudocódigo, uma variação do algoritmo *Round Robin*.

Algoritmo *ObterMatrizM*

Begin

minimum $\leftarrow \lfloor n_{\Pi} \ / \ n_p \rfloor$;

req_restantes $\leftarrow n_{\Pi}$ **mod** n_p;

If (*req_restantes* = *0*) **Then**

envie minimum *requisições para cada unidade*;

Else

envie minimum *requisições para* (n_p - req_restantes) *unidades*;

envie (minimum + 1) *requisições para* req_restantes *unidades*;

espere pelo resultado de cada unidade;

construa $\mathcal{R}_{\alpha}$ *a partir da partições recebidas*;

Return $\mathcal{R}_{\alpha}$

End.

Para a validação da primitiva α foi gerado um conjunto de dados sintético com 15 atributos. As cardinalidades dos domínios dos atributos previsores são as seguintes: 4 atributos de cardinalidade 2; 3 atributos de cardinalidade 4; 4 atributos de cardinalidade 6; 3 atributos de cardinalidade 10. O atributo alvo tem cardinalidade 2. A distribuição destas cardinalidades é melhor ilustrada pela Tabela 5.1. Os valores de todos os atributos seguem uma distribuição uniforme. Este conjunto de dados é chamado simplesmente de *S15* na descrição dos experimentos a seguir. (A Figura 5.2 corresponde à classe GOA++ definida para o armazenamento deste conjunto de dados.)

Experimento I

O primeiro experimento avaliou as execuções serial (utilizando-se um único nó da máquina virtual) e em paralelo da primitiva α sobre o banco de dados *S15*. Para a execução em paralelo, a máquina virtual paralela foi configurada com 4 nós. Foram geradas 80000 tuplas para este experimento. Pela aplicação do algoritmo descrito anteriormente, as 14 partições $\Pi(a_i)$ (referentes aos 14 atributos previsores de *S15*) foram geradas da seguinte forma: 2 nós geraram quatro partições cada, e

Tabela 5.1: Cardinalidades dos domínios dos atributos do conjunto de dados sintético *S15*.

Cardinalidade	*Atributo*
2	a_0, a_1, a_2, a_3, a_{14}
4	a_4, a_5, a_6
6	a_7, a_8, a_9, a_{10}
10	a_{11}, a_{12}, a_{13}

2 nó geraram 3 partições cada. Em cada bateria de execução da primitiva α, foram especificados 0, 2, 4, 6 e 8 predicados na cláusula WHERE [1]. Os resultados obtidos são exibidos na Figura 5.4. Nesta Figura, as curvas de rótulos *Alpha(1)* e *Alpha(4)* apresentam, respectivamente, os tempos de execução das versões serial (utilizando-se um único nó) e paralela (utilizando-se quatro nós) da primitiva α. O eixo das abscissas apresenta o número de predicados utilizados na cláusula WHERE da Declaração 5.1 e o eixo das ordenadas apresenta o tempo de execução medido em segundos.

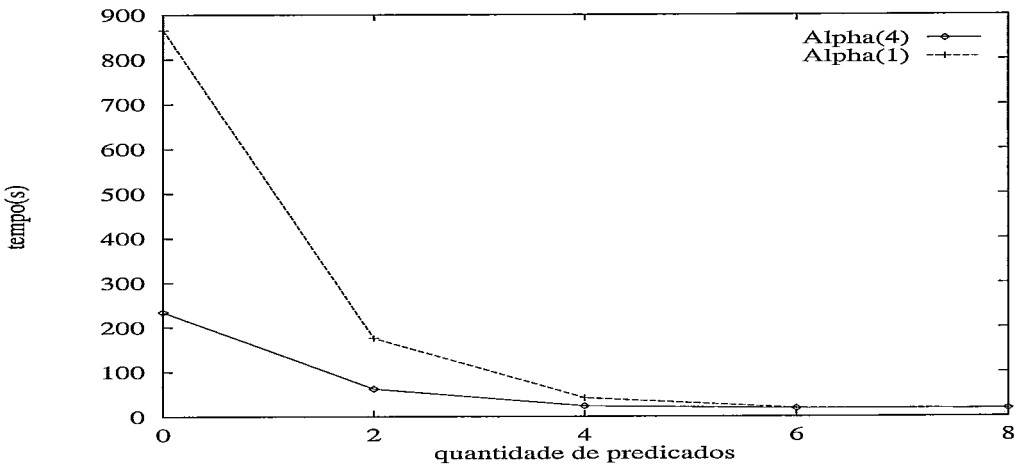

Figura 5.4: Resultados obtidos pela execução da primitiva α sobre *S15* (80.000 tuplas) com 0, 2, 4, 6 e 8 predicados na cláusula WHERE. São exibidos tanto os resultados da execução serial (em 1 único nó) quanto da execução paralela em 4 nós.

[1] Na implementação de uma algoritmo de indução de regras, isto eqüivaleria a especificar o ramo da árvore de decisão em que se está atualmente (vide Seção 4.3.3).

Pode-se notar pela Figura 5.4 que a versão paralela apresenta um ganho significativo no tempo de execução em relação à versão seqüencial quando nenhum predicado é especificado na cláusula WHERE. No entanto, à medida que predicados vão sendo acrescentados na cláusula WHERE, o ganho da versão paralela em relação à versão serial vai diminuindo até que se chega a um "empate" quando o número de atributos na cláusula WHERE é igual a 6. Isto se deve ao fato de que quanto menor o número de atributos utilizados na cláusula WHERE, menor a quantidade de tuplas a ser agrupada.[2]

A Figura 5.4 também permite notar que para os casos em que foram especificados 6 e 8 predicados na cláusula WHERE, os tempos de execução foram praticamente os mesmos tanto para a versão paralela quanto para a versão serial. Isto porque os atributos utilizados nestes casos foram atributos de cardinalidades grandes e, consequentemente, pouca seletividade, o que fez com que a quantidade de tuplas a serem ordenadas nos casos para 6 e 8 predicados fosse praticamente a mesma.

Experimento II

O segundo experimento realizado com a primitiva α foi o de testar sua execução em um número variável de nós na máquina virtual. Este experimento realizou a execução da primitiva α sem predicados na cláusula WHERE sobre configurações da máquina virtual paralela com 2, 4, 6 e 8 nós. Os resultados são ilustrados na Figura 5.5.

Estes resultados mostram que a primitiva α tem um grande potencial para paralelismo, muito embora não se tenha conseguido um ganho linear no tempo de execução neste experimento. Deve-se levar em conta que este experimento, assim como todos os outros aqui descritos, foram realizados sobre uma rede de estações de trabalho não isolada. Acrescente-se a isso o fato de que os nós da rede utilizados nas diversas configurações da máquina virtual paralela são de velocidades diferentes, e o tempo de execução da primitiva α depende do tempo de resposta do nó mais lento

[2]Note que, em uma declaração da forma *SELECT...FROM...WHERE...GROUP BY...*, as tuplas que não satisfazem à condição especificada na cláusula WHERE são retiradas antes de se realizar o agrupamento da tuplas pelos atributos especificados na cláusula *GROUP BY* [24].

da configuração.

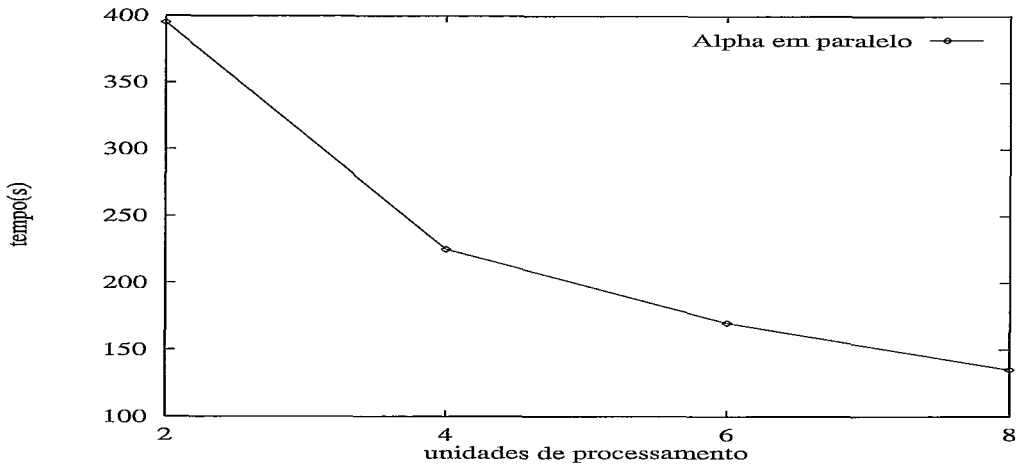

Figura 5.5: Resultados obtidos pela execução da primitiva α com 2, 4, 6 e 8 nós na máquina virtual paralela sobre o conjunto de dados *S15* com 80.000 tuplas.

Experimento III

Este experimento comparou o tempo de execução da versão serial da primitiva α com o tempo de execução total de diversas chamadas à Declaração 4.1 necessárias para se obter um resultado equivalente. Esta declaração é equivalente à primitiva *Count By Group*, proposta por Freitas em [44], e à *consulta pura*, proposta por John e Lent em [65] (vide Capítulo 6 de trabalhos relacionados). Os resultados são exibidos na Figura 5.6. Nesta figura, a plotagem de rótulo *Alpha Serial* exibe os tempos de execução da primitiva α, e a plotagem de rótulo *CBG/CP* exibe os tempos totais das várias execuções da Declaração 4.1.

Como visto no Capítulo 4, a Declaração 4.1 pode ser utilizada para se construir a matriz $\mathcal{M}$ através de n (número de atributos previsores) execuções, uma para cada atributo previsor. Deste ponto de vista, a extensão apresentada pela Primitiva α parece não ser tão relevante. No entanto, conforme ilustrado pela Figura 5.6, o ganho em termos de desempenho da primitiva α em relação à Declaração 4.1 é significativo.

Para entender os resultados apresentados na Figura 5.6, considere as Figuras 5.7 e 5.8 a seguir. Em cada uma destas figuras, é exibido o tempo total de execução (o

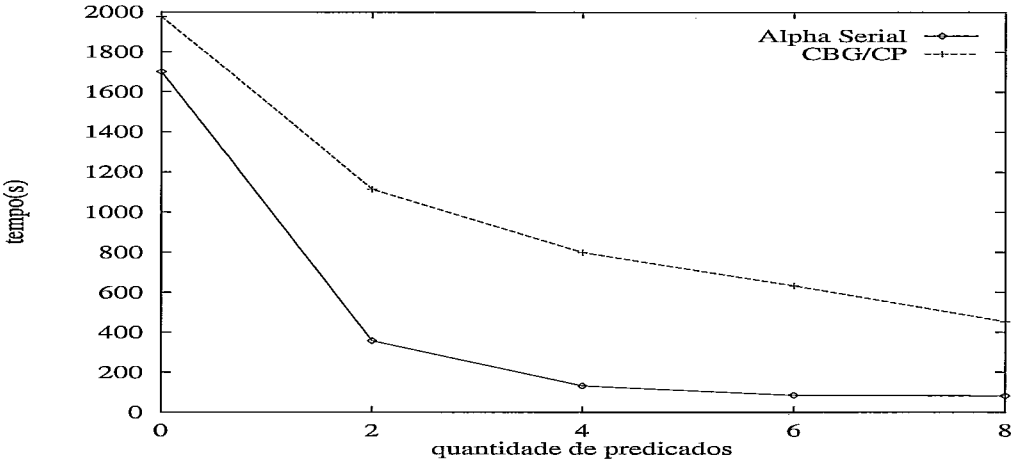

Figura 5.6: Resultados obtidos pela execução da versão serial da primitiva α e da Declaração 4.1 sobre *S15* (160.000 tuplas) com 0, 2, 4, 6 e 8 predicados na cláusula WHERE.

mesmo da Figura 5.6) e a fração deste tempo de execução que se levou para se fazer a ordenação e agrupamento das tuplas. (Estes tempos estão associados às plotagens de rótulos *Tempo total* e *Tempo de agrupamento*, respectivamente.)

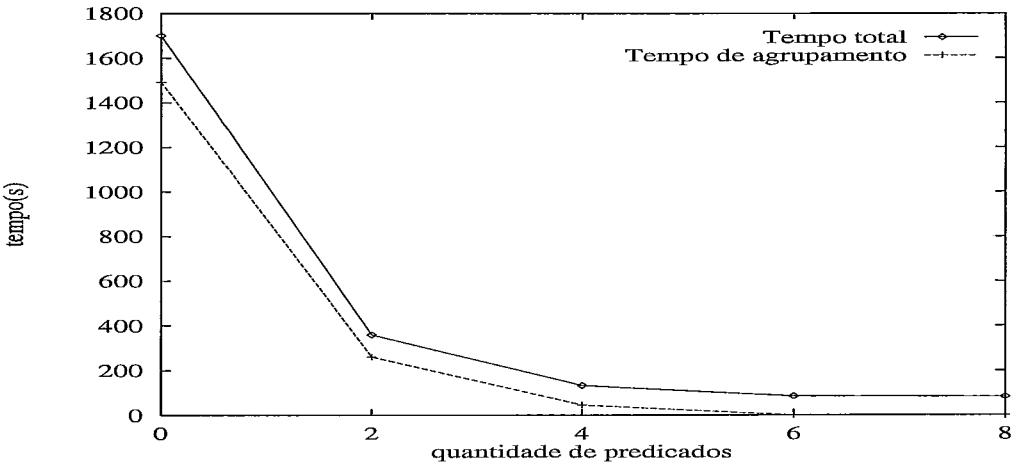

Figura 5.7: Tempos total e de agrupamento para a versão serial da primitiva α.

Analisando a Figura 5.7, pode-se notar que a fração do tempo de agrupamento é predominante no tempo total de execução da Primitiva α. Note que, neste caso, a relação $\mathcal{R}$ só precisa ser percorrida uma vez para a seleção das tuplas que satisfazem à conjunção de predicados especificados na cláusula WHERE.

Já na Figura 5.8, que mostra valores análogos aos da Figura 5.7, mas para execuções das primitivas *Count By Group* e *Consulta Pura*, o tempo de agrupamento

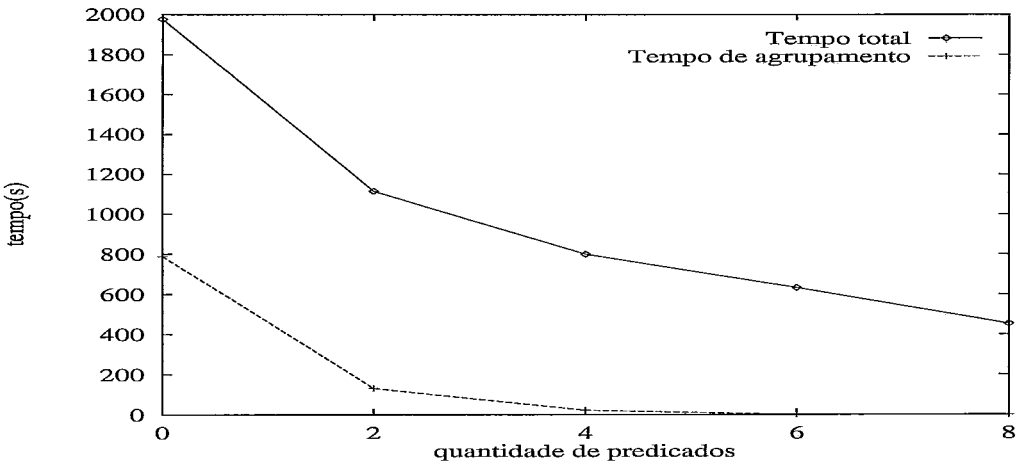

Figura 5.8: Tempos total e de agrupamento para as primitivas *Count By Group* e *Consulta Pura.*

é pequeno em relação ao tempo total de execução. Isto ocorre porque, neste caso, a relação $\mathcal{R}$ tem que ser percorrida várias vezes (uma para cada atributo previsor) em busca das tuplas que satisfazem à condição da cláusula WHERE.

5.4.2 Validação da Primitiva β

Os experimentos para validação da primitiva β comparam os desempenhos desta primitiva e da primitiva *Compute Tuple Distance* (descrita na Seção 6.3.3). Nestes experimentos, foram utilizados dois conjuntos de dados: o conjunto de dados *S15* descrito anteriormente neste Capítulo, e um dos conjuntos de dados do repositório UCI [78], *Letter Recognition.* Este conjunto de dados contém 20.000 tuplas, 1 classe (tipo de letra) e 16 atributos previsores numéricos. Os experimentos descritos nesta Seção foram realizados em uma máquina de processador Pentium 166MHz, 32Mb de memória RAM, com sistema operacional *Linux* versão 2.0.0.

Na Figura 5.9, são exibidos os tempos de execução das primitivas β e *Compute Tuple Distances* sobre o conjunto de dados *Letter Recognition* armazenado como uma $\mathcal{R}$-coleção no GOA++. Estes tempos correspondem às plotagens de rótulos *Beta* e *CTD*, respectivamente. O eixo das abscissas corresponde ao número de atributos utilizados na função de distância métrica (FDM).

Como se pode notar pela Figura 5.9, o tempo de execução de ambas as primitivas

aumenta em função da complexidade da FDM utilizada, ou seja, do número de atributos (dimensões) utilizados para se calcular a distância de uma tupla apresentada em relação as outras tuplas na $\mathcal{R}$-coleção.

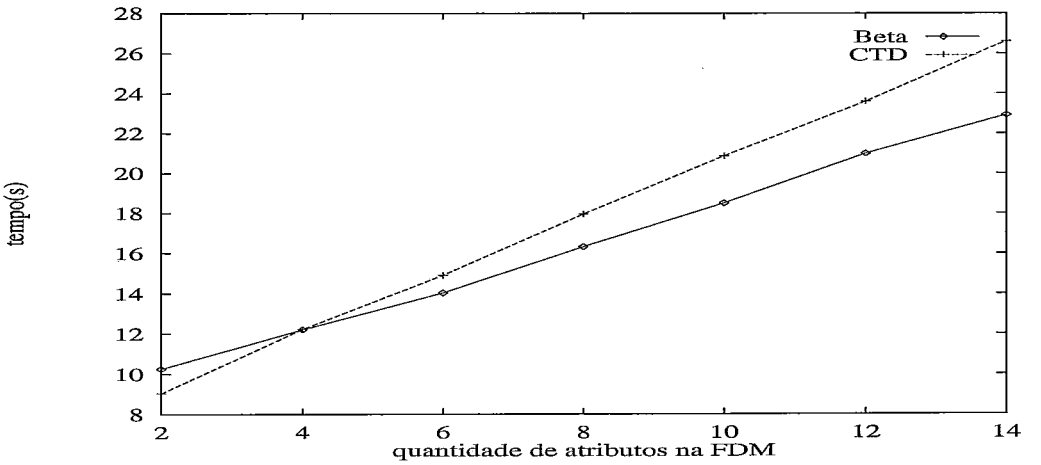

Figura 5.9: Resultados da execução das primitivas *Compute Tuple Distances* e β sobre o conjunto de dados *Letter Recognition* armazenado como uma $\mathcal{R}$-coleção no GOA++.

No entanto, neste experimento a primitiva *Compute Tuple Distances* se mostra mais sensível a este crescimento de complexidade do que a primitiva β. De fato, na primitiva *Compute Tuple Distances*, há que se calcular a função de distância o dobro de vezes do necessário na primitiva β. Na primitiva β, esta função é calculada uma vez para cada tupla do conjunto de dados (quando da formação da relação temporária a ser passada para a fase de ordenação/agrupamento).

O mesmo não acontece com a primitiva *Compute Tuple Distances*, na qual a função de distância deve ser calculada duas vezes para cada tupla, uma para a determinação do valor mínimo da distância (fase correspondente ao SELECT interno)[3], e outra para a determinação de que tuplas estão a tal distância mínima da tupla apresentada (fase correspondente ao SELECT externo).

Por outro lado, a primitiva *Compute Tuple Distances* não requer o uso de uma cláusula ORDER BY, a qual é necessária na primitiva β. Cabe lembrar que orde-

[3]A primitiva *Compute Tuple Distances* possui um *laço procedural*, ou seja, um aninhamento de comandos de seleção em uma declaração. A abordagem usual para a execução de uma declaração contendo um laço procedural é percorrer ambas as relações referenciadas [24].

nação é uma operaçào que requer uma complexidade de tempo $O(n \log n)$, onde n é o número de tuplas da relação $\mathcal{R}$. Neste caso, conforme a quantidade de tuplas de relação mina $\mathcal{R}$ cresce, se espera que o tempo de processamento da primitiva β fique cada vez maior em relação ao tempo de processamento da primitiva *Compute Tuple Distances.*

Note que a primitiva *Compute Tuple Distances* leva vantagem em relação à primitiva β para uma FDM simples (até quatro atributos, o tempo de execução da primeira é menor). Entretanto, conforme o número de atributos (dimensões) na FDM vai aumentando, a diferença entre os tempos de execução das duas primitivas diminui, até que, a partir de quatro atributos sendo utilizados na função de distância, o tempo de execução da primitiva β passa a ser menor do que a da primitiva *Compute Tuple Distances.*

As Figuras 5.10 e 5.11 apresentam os mesmos tempos totais exibidos na Figura 5.9, com adição dos tempos parciais de ordenação/agrupamento para a primitiva β e da sub-consulta (SELECT interno) para a primitiva *Compute Tuple Distances.* Note que o tempo parcial de ordenação/agrupamento da primitiva β é independente da complexidade da FDM. Para uma mesma quantidade de tuplas, o tempo de agrupamento permanece constante, não obstante a variação na complexidade da FDM.

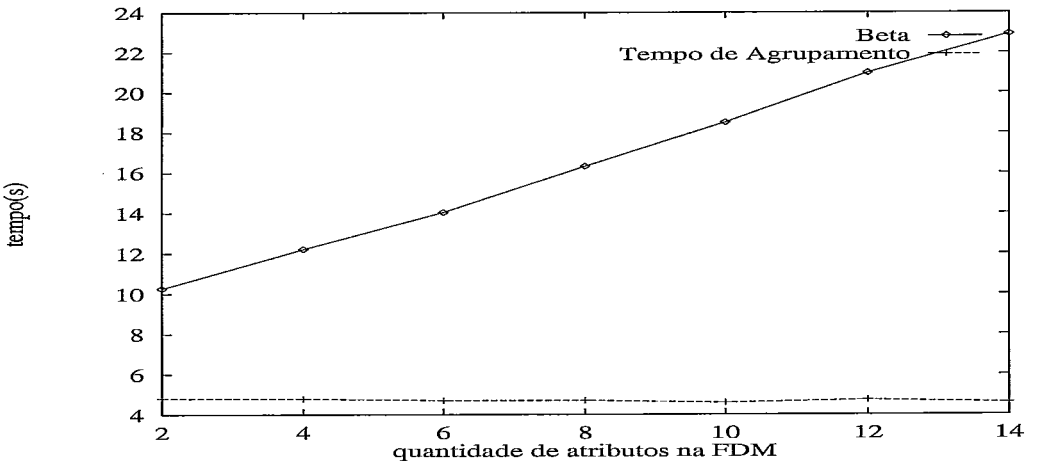

Figura 5.10: Tempo de execução total da primitiva β e tempo parcial da fase de ordenação/agrupamento das tuplas.

Já na primitiva *Compute Tuple Distances*, o tempo parcial relativo à execução da sub-consulta presente na primitiva *Compute Tuple Distances* aumenta proporcionalmente à complexidade da FDM. Isto explica porque a primitiva β passa a ter um tempo de execução relativamente menor conforme a complexidade da FDM aumenta.

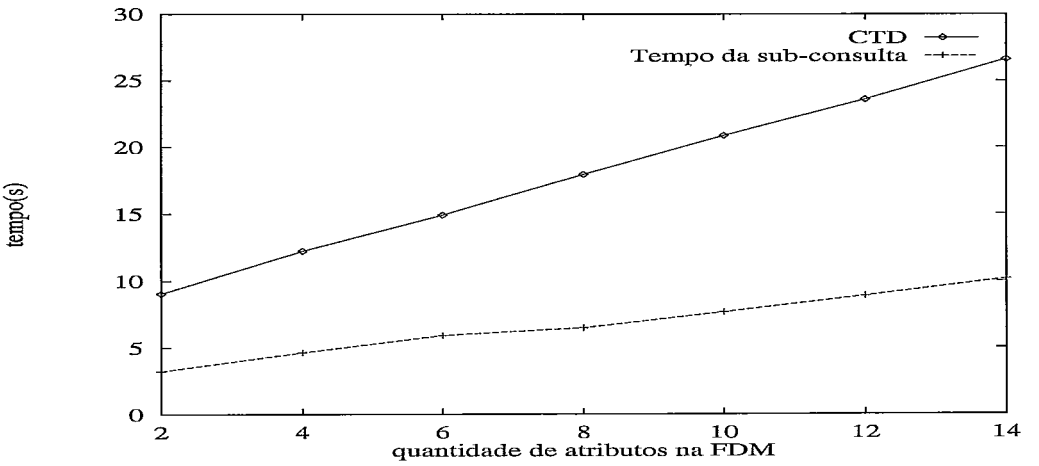

Figura 5.11: Tempo de execução total da primitiva *Compute Tuple Distances* e tempo parcial da fase de determinação do valor mínimo da FDM.

Capítulo 6

Trabalhos Relacionados

[A]ny great [...] invention is the summing up of the labour, the experience, the reason, and even the blunders of numerous workmen ...

Charles Darwin, em The Origin of Species

6.1 Introdução

Em [13], Baião, Mattoso e Zaverucha definem três níveis de integração entre sistemas de MD e SGBDs. Estes três níveis são descritos a seguir.

- No primeiro nível, a integração se realiza através de conversão de dados. O banco de dados serve simplesmente como um repositório de dados, e os dados devem ser convertidos antes de serem processados pelo sistema de MD, que não precisa ser modificado.

- O segundo nível, integração em relação ao acesso aos dados, se caracteriza por uma forte ligação entre o SGBD e o sistema de MD. Não somente os dados, mas também parte do conhecimento do sistema é armazenado no banco de dados. Nesta abordagem, o sistema precisa ser modificado para que as funcionalidades de acesso aos dados sejam direcionadas diretamente para o banco de dados. Embora isto possa parecer uma desvantagem (necessidade de modificação dos

algoritmos), note que estas funcionalidades de acesso aos dados podem ser definidas de modo a capturar as operações mais básicas e que mais demandam tempo de processamento, o que justifica e valida este tipo de integração.

- O terceiro nível, integração de programas, define a ligação mais forte dos três níveis entre o sistema de MD e o SGBD. Neste nível o sistema de MD é encapsulado como um módulo no interior do SGBD.

O primeiro nível obviamente não oferece vantagens, em virtude de os dados terem que se adaptar ao formato de entrada do algoritmo em questão. Um descrição mais detalhada desta questão se encontra na Seção 1.2.

Este Capítulo está organizado em duas partes principais, representadas pelas Seções 6.2 e 6.3. A primeira descreve os trabalhos da literatura que preconizam uma integração mais forte das aplicações de mineração de dados com o SGBD. Estes trabalhos estão associados ao segundo e ao terceiro níveis descritos acima. A segunda parte deste Capítulo revisa, dentre estes trabalhos, os que, assim como o trabalho apresentado nesta tese, propõem primitivas para mineração de dados.

6.2 Integração Forte com SGBDs

Esta Seção descreve os trabalhos da literatura que preconizam uma integração forte com o SGBD. A Seção 6.2.1 descreve propostas que utilizam o conceito de linguagem de mineração de dados (LMD). A Seção 6.2.2 descreve outros trabalhos neste nível de integração, mas que não utilizam LMDs.

6.2.1 Linguagens de Mineração de Dados

Diversos trabalhos na literatura têm proposto a utilização de uma *Linguagem de Mineração de Dados* para extração de padrões a partir de uma relação mina $\mathcal{R}$. Uma LMD é uma linguagem declarativa que permite a definição do padrão a ser extraído a partir de $\mathcal{R}$, da mesma forma que as linguagens OQL e SQL permitem definições declarativas do que se quer obter como resultado. Nestes trabalhos, o

foco/objetivo está em aumentar a produtividade de desenvolvimento de aplicações de MD através da criação de funcionalidades específicas à MD e disponibilização destas funcionalidades de uma forma declarativa.

De uma forma geral, estas linguagens são implementadas sobre um sistema de MD (vide Seção 2.2.1), que pode utilizar interfaces gráficas para a apresentação dos resultados. Tal abordagem é diferente da adotada nesta tese, em virtude de LMDs estarem em um nível mais alto de abstração que as primitivas aqui propostas.

O restante desta Seção revisa três LMDs existentes na literatura. São elas: *LDL++*, *DMQL* e *M-SQL*, respectivamente.

A Linguagem LDL++

Em [94], Shen et al. apresentam a linguagem de programação lógica *LDL++* como uma linguagem de *meta-consultas* do sistema de MD denominado *Knowledge Miner* (KM). Meta-consultas constituem um formato geral através do qual se pode que especificar o tipo de padrão a ser extraído. Por exemplo, sejam P, Q e R variáveis para predicados. Então, a meta-consulta $P(X,Y) \wedge Q(Y,Z) \rightarrow R(X,Z)$ especifica que os padrões a serem descobertos são relações transitivas da forma $p(X,Y) \wedge q(Y,Z) \rightarrow r(X,Z)$, onde p, q e r são predicados específicos. Um possível resultado desta meta-consulta é o padrão $cidadão(X,Y) \wedge linguagemOficial(Y,Z) \rightarrow fala(X,Z)$ [0.93], onde *cidadão*, *linguagemOficial* e *fala* são relações que são instanciadas a partir de P, Q e R, respectivamente.

Meta-consultas podem ser geradas por um especialista ou pelo próprio sistema KM. Neste último caso, as meta-consultas podem ser analisadas e refinadas pelo especialista. Por outro lado, não há suporte da linguagem para quantificação estatística das regras imprecisas por meios de conceitos como suporte e confiança.

Um banco de dados dedutivo é utilizado para coletar os dados que são agrupados através da utilização de um método de agrupamento *bayesiano*. Regras são extraídas a partir dos agrupamentos e armazenadas em uma base de conhecimento que um usuário pode utilizar para formar suas próprias meta-consultas.

A Linguagem DMQL

Em [54], é descrita a linguagem *DMQL*, que está embutida na arquitetura do sistema *DBMiner*. Esta LMD permite a especificação declarativa de quatro componentes de mineração de dados: (1) a relação mina, (2) o tipo de conhecimento a ser extraído, (3) o conhecimento de domínio existente, e (4) os limites de interesse do conhecimento a ser extraído (*tresholds*).

Esta LMD propõe-se a gerar diversos tipos de regras a partir de uma relação armazenada em um banco de dados relacional, incluindo regras de associação e regras de classificação. Para o primeiro tipo a utilização de uma LMD parece factível, em virtude de a extração de regras de associação ser uma tarefa determinística (vide Seção 2.4.4). No entanto, não fica claro como a linguagem DMQL possa ser utilizada para gerar regras de classificação, padrões associados a uma tarefa de mineração de dados bem mais complexa.

A Linguagem M-SQL

Outro preconizador da abordagem declarativa de alto nível para realização de mineração de dados é Imielinski que, em [64], descreve o sistema *DataMine*, constituído de uma interface para programação de aplicações de mineração de dados e da LMD *M-SQL* (*Mining SQL*). Esta LMD estende a linguagem SQL através do operador *MINE* para a geração de regras com suporte e confiança especificados. O formato geral desta linguagem é exibido abaixo.

```
SELECT
FROM MINE(T)
WHERE
        condition
        MinSupp < SUPPORT < MaxSupp
        MinConf < CONFIDENCE < MaxConf
```

Declaração 6.1: Sintaxe geral da linguagem *M-SQL*.

O esquema proposto por Imielinski é o de utilizar a linguagem M-SQL embutida

em uma linguagem de programação (como, por exemplo, C++). Uma declaração especificada nesta LMD seria, então, passada a um otimizador de consultas M-SQL, que traduziria a expressão em um plano de execução.

Em outro trabalho [61], Imielinski define o conceito de Mineração de Dados como sendo equivalente a um processo de consulta a um banco de dados. Com base neste conceito, é feita a descrição de um *Sistema de Gerência de Extração de Dados e Conhecimento* (SGEDC), que é definido como um sistema para dar suporte a aplicações de MD da mesma forma que SGBDs dão suporte a aplicações de Sistemas de Informação. Neste sistema, objetos de mineração de dados seriam tratados da mesma forma que objetos convencionais do SGBD. Imielinski descreve alguns exemplos de consultas possíveis em um SGEDC:

- Gerar um classificador utilizando como conjunto de treinamento um conjunto de dados definido pelo usuário, e utilizando atributos e categorias de classificação especificados pelo usuário.

- Gerar regras com atributos do antecedente e do conseqüente especificados pelo usuário. Encontrar todos os objetos do banco de dados que não se enquadram nestas regras.

- Gerar regras utilizando valores de determinados atributos gerados por uma consulta convencional.

- Encontrar objetos do banco de dados que pertençam ao maior grupo de um agrupamento construído de acordo com uma determinada métrica de distância definida pelo usuário.

Realmente, a construção de um SGEDC com todas estas funcionalidades implementadas seria um trabalho revolucionário. O próprio autor desta dissertação chegou a propor um protótipo de um SGEDC em [103]. Neste trabalho, é descrita uma abordagem declarativa para extração de regras a partir de um SGBD, onde o usuário poderia especificar o tipo de regra a ser extraída através de uma linguagem

declarativa resultante da extensão da linguagem declarativa do SGBD. Os algoritmos para extração de regras estariam implementados no próprio SGBD. No entanto, a implementação para a extração de diferentes tipos de regras se torna impraticável quando se considera que tal declaração resulta na execução de um determinado algoritmo e que não existe um algoritmo que se possa considerar como o melhor em todos os casos [91]. Esta abordagem, assim como todas as abordagens utilizando-se LMDs, esbarra na dificuldade imposta pelas grandes diversidade e complexidade dos algoritmos de mineração de dados existentes.

6.2.2 Outros Trabalhos

Em [97], Sousa descreve a implementação de um algoritmo de indução de regras em um sistema de gerenciamento de banco de dados paralelo. A versão paralela do banco de dados comercial *Oracle* foi utilizada nos experimentos. Para a implementação do algoritmo, foram utilizados procedimentos armazenados (*stored procedures*) para a construção de partes do mesmo que fossem caras dos pontos de vista computacional e de operações de entrada/saída. Desta forma, boa parte do algoritmo é executada no próprio SGBD, fato que aumenta de sobremaneira o desempenho e diminui seu tempo de execução.

Em [4], Agrawal et al. o sistema *Quest* é descrito. Neste sistema os algoritmos de mineração são executados no servidor e o cliente interage com o sistema através de uma IGU (interface gráfica com o usuário). Os algoritmos implementados incluem algoritmos para as tarefas de regras de associação, padrões seqüenciais, agrupamento baseado em séries temporais e classificação.

Em [98], várias alternativas são consideradas e comparadas para a extração de regras de associação a partir de bancos de dados relacionais. Estas alternativas incluem: acoplamento fraco através de cursores SQL; encapsulamento do algoritmo em um procedimento armazenado; utilização de *cache* temporário de dados em um sistema de arquivos; acoplamento forte através da utilização de funções definidas pelo usuário. Um ponto em comum do trabalho apresentado em [98] e o desta tese é que os autores estudam formas de expressar as operações mais caras do algoritmo

através de consultas declarativas.

Em [56], o projeto e a implementação de uma arquitetura de dois níveis para um sistema de MD são descritos. Esta arquitetura consiste de uma ferramenta de mineração de dados e de um SGBD paralelo denominado *Monet*. A ferramenta de mineração de dados organiza e controla o processo de procura enquanto que o SGBD fornece as funcionalidades necessárias para acesso aos dados. Este trabalho explora o paralelismo na execução das consultas, onde o resultado de uma pode ser utilizado para a obtenção do resultado de outra.

6.3 Primitivas para Mineração de Dados

Esta Seção descreve os trabalhos da literatura onde são propostas primitivas de MD semelhantes às apresentadas no Capítulo 4. Estes trabalhos se encaixam no segundo nível de integração descrito na Seção 6.1.

6.3.1 O Sistema MKS

Em [10], Anand et al. descrevem o sistema *MKS* (*Mining Kernel System*). Este sistema é constituído de um conjunto de bibliotecas de funções [1]. Cada uma destas bibliotecas tem o objetivo de fornecer "blocos de montagem"para a construção de aplicações de mineração de dados. As bibliotecas do MKS podem ser agrupadas em dois módulos principais: O *Módulo de Interface* e o *Módulo de Mineração*. O primeiro tem a função de fornecer meios de interação entre a aplicação sendo construída e o sistema *MKS*. Já o segundo módulo é formado pelas bibliotecas de funções contendo funcionalidades genéricas para a implementação da aplicação.

Em relação ao trabalho aqui apresentado, o sistema *MKS* está em um nível mais alto de abstração do ponto de vista do fornecimento de funcionalidades genéricas para a construção de aplicações. Se por um lado, esta característica é vantajosa, por

[1] Biblioteca de Dados Virtuais, Biblioteca de Teoria da Informação, Biblioteca de Estatística, Biblioteca de Teoria de Evidências, Biblioteca de Representação do Conhecimento, Biblioteca de Entrada/Saída de Conhecimento, e Biblioteca de Manipulação de Conjuntos.

outro, pode ser um ponto negativo em virtude de que o *MKS* não está diretamente integrado a um banco de dados. Por exemplo, a *Biblioteca de Dados Virtuais*, responsável pela interface com sistemas de bancos de dados, trabalha com um arquivo denominado *DSM* (Data Source Mapping File) onde o usuário especifica os atributos a serem utilizados na tarefa de mineração. A especificação neste arquivo é posteriormente traduzida para uma expressão em SQL para a obtenção dos dados. No entanto, na sintaxe de especificação, não é permitida a existência de funções de agregação, o que torna a relação mina resultante ineficiente. Ou seja, embora fornecendo uma funcionalidade de alto nível para a especificação da relação mina (o arquivo *DSM*), tal funcionalidade proíbe implementações mais eficientes de um algoritmo de classificação, por exemplo. Neste sentido, o trabalho apresentado nesta tese pode ser considerado como sendo de granulosidade mais fina quanto às funcionalidades de acesso aos dados.

6.3.2 O Protocolo SIP

Em [65], John e Lent descrevem o *Protocolo SIP* (*SQL Interface Protocol*).[2] Este trabalho é bastante relacionado ao desta tese, em virtude de preconizar a mesma estratégia para acesso aos dados: ao invés de se obter acesso a todo o conteúdo da relação mina, pode-se utilizar funcionalidades do próprio SGBD para transformá-la e obter as informações estatísticas necessárias. O Protocolo SIP é um arquétipo para a comunicação de dados entre um algoritmo de mineração de dados e um SGBD que tem este objetivo. Este protocolo é constituído de dois tipos básicos de primitivas: *Consultas Puras* e *Consultas Impuras*. Nas Declarações 6.2 e 6.3, f é uma função de agregação, ϕ é uma lista de nomes de atributos e Ψ é uma lista de condições sobre atributos. A lista ϕ contém exatamente k atributos.

Ainda em [65], John e Lent descrevem como o algoritmo C4.5 e o classificador bayesiano simples podem ser implementados utilizando-se as primitivas constituintes do protocolo SIP.

[2]Uma versão expandida deste artigo pode ser encontrada no Capítulo 2 da tese de doutorado de G. John [66].

```
SELECT φ, f(*)
FROM D
WHERE Ψ
GROUP BY φ
```

Declaração 6.2: Consultas Puras

```
SELECT φ
FROM D
WHERE Ψ
```

Declaração 6.3: Consultas Impuras de Grau k

A Primitiva α estende o trabalho apresentado em [65] em virtude de ela ser mais eficiente com respeito ao tráfego de dados entre cliente e servidor. Por exemplo, em [65], o algoritmo *C4.5* é implementado utilizando-se consultas puras, onde, para cada atributo previsor candidato a participar do teste de um determinado nó da árvore de decisão, uma consulta é enviada ao SGBD. Este mesmo algoritmo pode ser implementado através da primitiva α, sendo que, neste caso, somente uma consulta é suficiente para a obtenção das informações relativas aos atributos. Além disso, a mesma matriz $\mathcal{M}$ pode ser utilizada para implementação de um classificador bayesiano simples (vide Seção 4.3.4).

6.3.3 Count By Group e Compute Tuple Distances

Outro trabalho relacionado que influenciou bastante o desta tese é o de Freitas [44, 46, 38], onde são apresentadas diversas primitivas, dentre a quais destacam-se as primitivas *Count By Group* e *Compute Tuple Distances.* Resultados práticos da utilização destas primitivas em um SGBD com capacidade para processamento paralelo de dados são apresentados.

A primitiva *Count By Group* é uma primitiva para ser utilizada por algoritmos de indução de regras (vide Seção 3.4.1) e tem os seguintes parâmetros de entrada:

1. Um atributo candidato.

2. Um atributo alvo.

3. Um *descritor de conjunto de tuplas*, que é uma conjunção lógica de pares da forma atributo/valor descrevendo as tuplas cobertas pela regra candidata atual.

A saída desta primitiva é uma matriz $m \times n$ com totais por linhas e colunas, onde m é o total de valores distintos do atributo candidato e n de valores distintos do atributo alvo. Cada célula *(i, j)* desta matriz ($1 < i < m$, $1 < j < n$) contém o número de tuplas satisfazendo o descritor de conjunto de tuplas com o valor do atributo candidato a_i e valor do atributo alvo a_j. Freitas propõe o mapeamento desta primitiva para SQL da seguinte forma:

```
SELECT AtributoCandidato, AtributoAlvo, COUNT(*)
FROM R
WHERE DescritorDeConjuntoDeTuplas
GROUP BY AtributoCandidato, AtributoAlvo
```

Declaração 6.4: Primitiva *Count By Group*.

Pode-se notar que a Declaração 6.4 produz exatamente uma partição $\Pi(a_i)$ da matriz $\mathcal{M}$ para um dos atributos previsores (vide Capítulo 4). Na verdade, a primitiva *Count By Group* serviu de ponto de partida para a definição da primitiva α. A diferença entre estas duas primitivas é que a primitiva α pode produzir as informações estatísticas para *todos* os atributos candidatos de uma só vez.

O próprio Freitas já havia sugerido em um trabalho anterior [37] que seria vantajosa a implementação de uma extensão à linguagem SQL para prover esta funcionalidade. Neste mesmo trabalho, ele propõe uma sintaxe para esta extensão, através da cláusula denominada *FOREACH* que produz um resultado equivalente ao produzido pela extensão da cláusula *GROUP BY* utilizada pela Primitiva α. Esta extensão, denominada *Multiple Count By Group*, é apresentada na Declaração 6.5.

Outra primitiva apresentada por Freitas é a denominada *Compute Tuple Distances*. De forma semelhante à primitiva β, *Compute Tuple Distances* tem o objetivo

```
SELECT AttrID, Class, COUNT(*)
FROM R
WHERE condition
GROUP BY AttrID, Class
FOREACH AttrID IN CandidateAttributeList
```

Declaração 6.5: Primitiva *Multiple Count By Group.*

de dar suporte ao cálculo de uma métrica de distância entre uma nova tupla e todas as tuplas armazenadas na relação mina. Freitas apresenta o mapeamento em SQL desta primitiva da seguinte forma:

```
SELECT classe
FROM R
WHERE Dist(X, Y) IN (SELECT MIN(Dist(X, Y)) FROM R)
```

Declaração 6.6: Primitiva *Compute Tuple Distances.*

Na forma geral da primitiva *Compute Tuple Distances* exibida pela Declaração 6.6, $Dist(X, Y)$ denota a distância entre a tupla apresentada X e a tupla Y armazenada na relação $\mathcal{R}$. Note que a primitiva *Compute Tuple Distances* retorna um conjunto de valores do atributo alvo. No entanto, esta primitiva só retorna valores do atributo alvo associados a tuplas cuja distância seja mínima em relação à tupla X. Como não há como se saber, a *priori*, o número de tuplas retornadas por esta primitiva, sua utilização para a implementação do algoritmo k-NN para um valor arbitrário de k não é possível. Já a primitiva β (Seção 4.3.5) retorna uma lista ordenada por valores de distância entre a tupla X e as tuplas Y da relação $\mathcal{R}$. O resultado desta primitiva permite que se implemente o algoritmo k-NN para qualquer valor possível de k. Deste ponto de vista, a primitiva β é mais genérica que a primitiva *Compute Tuple Distances.*

Outra comparação que pode ser feita entre estas duas primitivas é com respeito à cardinalidade da relação resultante. Na primitiva β, a cardinalidade da relação resultante é bem maior, podendo se igualar a da relação $\mathcal{R}$ no caso em que não

haja duas tuplas em $\mathcal{R}$ com a mesma distância à tupla apresentada. Mas, para problemas de classificação típicos, o número de valores distintos de um atributo não é tão grande, o que contribui para diminuir a cardinalidade do resultado da primitiva β. Além disso, as tuplas resultantes são lidas pela aplicação uma por vez, o que permite que esta leia somente a quantidade de tuplas necessária para a determinação das k tuplas mais próximas.

6.3.4 O Operador *UNPIVOT*

Outro trabalho relacionado ao desta tese é o de Graeffe, Fayyad e Chaudhuri em [48]. Neste artigo, é proposta outra extensão à linguagem SQL, através do operador *UNPIVOT*. Este operador consome cada tupla da relação $\mathcal{R}$ e produz n tuplas no formato (*AttrID, AttrValue, Class*), onde n é o número de atributos previsores de uma tupla em $\mathcal{R}$. Através deste operador, pode-se construir uma estrutura equivalente a da matriz $\mathcal{M}$, com a seguinte declaração:

```
SELECT AttrID, AttrValue, Class, COUNT(*)
FROM R.UNPIVOT(AttrValue FOR AttrID IN (a1, ..., an))
GROUP BY AttrID, AttrValue, Class
```

Declaração 6.7: Operador *UNPIVOT*.

Uma propriedade interessante que o operador *UNPIVOT* possui é a do *princípio de fechamento*, segundo o qual o resultado de uma consulta pode ser objeto de nova consulta [28]. Esta característica é também encontrada na primitiva α, embora, nesta última, não se utilize um operador.

Outra comparação que pode ser feita é com relação à sintaxe das duas alternativas para a obtenção da matriz $\mathcal{M}$. Embora a matriz $\mathcal{M}$ possa ser obtida tanto pela utilização da primitiva α, quanto pela utilização de uma declaração na forma da Declaração 6.7, a sintaxe da primitiva α é mais intuitiva e simples, sendo constituída de uma extensão a uma cláusula preexistente e bem conhecida da linguagem de consulta, a cláusula GROUP BY.

Capítulo 7

Conclusões

> A conclusion is simply the place where someone got tired of [or doesn't have more time to] thinking.
>
> *Fortune* do UNIX

7.1 Resumo

Esta dissertação foi dividida em sete capítulos, entre eles um de introdução no qual se apresentou a motivação para o trabalho aqui apresentado através da análise das desvantagens da abordagem atual de acesso aos dados utilizada por grande parte das aplicações de mineração de dados. Também foi definida uma estratégia para abordar o problema.

O segundo capítulo apresentou uma revisão dos conceitos mais utilizados na área de pesquisa em mineração de dados, descrevendo algumas definições deste termo, áreas de pesquisa relacionadas, as diversas tarefas e métodos utilizados.

O terceiro capítulo apresentou a tarefa de classificação em maiores detalhes, já que a proposta desta tese enfoca métodos de mineração de dados associados a esta tarefa em particular. Foram descritos diversos métodos de classificação. Os métodos de *Indução de Regras*, *Aprendizado Bayesiano Simples*, e *Aprendizado Baseado em Instâncias* mereceram uma descrição mais detalhada pelo fato de que as implemen-

tações de tais métodos podem ser feitas utilizando-se às primitivas propostas nesta tese.

No quarto capítulo, foram propostas duas novas primitivas de mineração em bancos de dados para classificação. A primitiva α, que pode ser utilizada na implementação de algoritmos de indução de regras e de aprendizado bayesiano, e a primitiva β que pode ser utilizada para se implementar algoritmos do método de aprendizado baseado em instâncias. Para cada um destes métodos, foi descrito de que modo a respectiva primitiva poderia ser utilizada para a implementação de um algoritmo.

Os aspectos computacionais e de implementação das primitivas α e β são descritos no quinto capítulo. Neste capítulo, é dada uma breve descrição dos sistemas GOA++ e PVM, utilizados na implementação das primitivas. A primitiva α, sendo uma extensão à linguagem de consulta, mereceu uma descrição mais detalhada de sua implementação. Para ambas as primitivas foram feitos experimentos comparativos a primitivas propostas em outros trabalhos.

No sexto capítulo são descritos alguns trabalhos relacionados aos desta tese. São descritos diversos trabalhos da literatura que preconizam uma integração mais forte com o SGBD onde estão armazenados os dados a serem utilizados para mineração. Foi dada uma especial atenção aos trabalhos da literatura que, assim como o desta tese, propuseram primitivas de mineração de dados.

Neste presente capítulo, são apresentadas as considerações finais do trabalho de tese, suas contribuições para o estado da arte, e sugestões para trabalhos futuros.

7.2 Contribuições desta tese

A abordagem de aumentar o grau de integração entre aplicações de mineração de dados e o SGBD com o objetivo de possibilitar uma maior padronização e produtividade na construção destas aplicações não é inédita, tendo alguns trabalhos propostos na literatura (alguns deles descritos no Capítulo 6). No entanto, as propostas abordando este problema de integração através de primitivas de mineração de dados são

relativamente recentes, o que pode ser notado pelo períodos de publicação dos trabalhos descritos na Seção 6.3. Esta tese contribuiu para esta área de estudo através da definição de duas novas primitivas de mineração de dados a serem utilizadas para a tarefa de classificação.

Em relação à primitiva α, esta resultou de uma extensão natural de primitivas propostas nos trabalhos de Freitas [44], através da primitiva *Count By Group*, e de John [66], através de *consultas puras*. Outros trabalhos [37, 48] já haviam proposto primitivas semelhantes à primitiva α, mas, até onde vai o conhecimento do autor desta tese, este trabalho é o único até o momento a definir, implementar, e validar uma primitiva desta espécie.

Os resultados das avaliações apresentadas no Capítulo 5 mostram que através da primitiva α os resumos estatísticos dos dados necessários em um algoritmo de indução de regras ou de aprendizado bayesiano simples podem ser recuperados mais eficientemente, em virtude do alto potencial para paralelismo desta primitiva.

Com respeito à primitiva β, esta surgiu como uma alternativa à primitiva *Compute Tuple Distances* proposta por Freitas para dar suporte à implementação de algoritmos de aprendizado baseado em instâncias, particularmente o algoritmo k-NN. Os experimentos realizados com estas primitivas no sistema GOA++, e descritos no Capítulo 5, mostram que a primitiva β é menos sensível ao aumento da complexidade da função de distância métrica utilizada, fato que levou a resultados melhores no tempo de execução desta primitiva em relação à primitiva *Compute Tuple Distances* no GOA++.

7.3 Direções futuras

Este trabalho de tese levantou diversas outras questões que necessitam de maiores estudos. Esta Seção discute algumas destas questões que serão alvo de pesquisa em trabalhos posteriores.

Embora, neste trabalho de tese, tenha sido descrito como as primitivas aqui propostas poderiam ser utilizadas na implementação dos métodos de Indução de

Regras, Aprendizado Bayesiano Simples, e Aprendizado Baseado em Instâncias, não foi apresentada uma implementação prática destes algoritmos. Um trabalho que já está em andamento é a implementação destes algoritmos, com o objetivo de atestar de modo prático as primitivas α e β.

O desempenho da primitiva β em um ambiente paralelo também é outro tópico a ser pesquisado. Intuitivamente, esta primitiva tem um certo potencial para ser paralelizada, em virtude de que se pode utilizar as unidades de processamento (nós) disponíveis para se fazer a ordenação de partições da relação mina que seriam posteriormente reaglutinadas para produzir o resultado final. Um estudo possível de ser feito é a utilização dos sistemas GOA++ e PVM, assim como foi feito em relação à primitiva α.

Um fator que de certo modo atrapalhou a execução dos experimentos de validação da primitiva α foi o de utilização de uma rede de estações não isoladas. Neste ambiente, o tempo de execução das consultas variou bastante em função da carga na rede e das estações componentes da máquina virtual paralela. Um trabalho que já está sendo estudado é a utilização de uma máquina paralela SP2 da IBM^{TM} para realização daqueles e de outros experimentos.

Nos experimentos relativos à primitiva α, o conjunto de dados utilizados foi replicado pelos diversos nós da máquina virtual paralela. Uma outra abordagem que certamente pode melhorar o desempenho desta primitiva é a fragmentação (distribuição) horizontal da relação mina por entre estes nós. Isto acarretaria em um menor tempo para se percorrer cada fragmento, o que poderia resultar em uma diminuição do tempo total de execução da primitiva.

Uma outra questão que pode influenciar bastante no desempenho das primitivas α e β é a utilização de índices para os atributos referenciados nas declarações destas primitivas. Esta questão não foi tratada neste trabalho em virtude de que a versão atual do GOA++ não dá suporte à especificação de índices para objetos de um banco de dados. No entanto, existem trabalhos em andamento na linha de Bancos de Dados do COPPE para incorporação de índices a versões futuras do GOA++ [81]. Uma vez que estas funcionalidades estejam disponíveis no GOA++, pode-se

fazer experimentos para avaliar o quão efetivamente a utilização de índices pode influir no desempenho das primitivas.

Seguindo a abordagem de integração de aplicações de mineração de dados a SGBDs, outra questão que merece estudos adicionais é o aproveitamento de funcionalidades específicas de SGBDs orientados a objetos para melhorar o processo de mineração de dados. Aproveitando a infra-estrutura fornecida pelo GOA++, estudos poderiam ser realizados para se avaliar a utilização de métodos na implementação de algoritmos de mineração de dados. Intuitivamente, esta abordagem de implementação parece ter um grande potencial para melhorar o desempenho de determinados algoritmos, em virtude de que parte da lógica destes passaria a ser executada no servidor. Uma abordagem semelhante, mas para um SGBD relacional, foi utilizada por Sousa [98, 97], que implementou um algoritmo para indução de regras através de procedimentos armazenados no SGBD Oracle^{TM} (vide Seção 6.2.2).

A criação de primitivas para outras tarefas de mineração de dados também parece ser uma área com muito potencial para ser pesquisada. Dos relativamente poucos trabalhos da literatura que tratam do problema de integração de aplicações de mineração de dados com SGBDs, a grande maioria enfoca a tarefa de classificação. (Esta tarefa é com certeza a mais estudada e a mais utilizada na prática.) Um exemplo de trabalho existente na literatura que aborda este problema de integração, mas enfocando outra tarefa, é o de Sarawagi [99], onde são apresentadas diversas abordagens para a extração de regras de associação de SGBD relacionais.

Em [31], é apresentado o algoritmo *RISE*, que combina aspectos de aprendizado baseado em instâncias e indução de regras. Experimentos sobre coleções de dados reais descritos neste artigo apresentam significante melhoria na precisão dos modelos gerados, mostrando que a sinergia entre diferentes paradigmas de MD é um caminho de pesquisa proeminente.

Referências Bibliográficas

[1] D. W. Aha. "Case-Based Learning Algorithms". In: *Proceedings of the DARPA Case-Based Reasoning Workshop*, pp. 147–158, Washington, D. C., 1991. Morgan Kaufmann Publishers, San Mateo, California.

[2] R. Agrawal, T. Imielinski e Arun Swami. "Mining association rules between sets of items in large databases". *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(2):207–216, jun 1993.

[3] R. Agrawal, T. Imielinski e Arun N. Swami. "Mining Association Rules between Sets of Items in Large Databases". Peter Buneman e Sushil Jajodia editores, In: *1993 ACM SIGMOD International Conference on Management of Data*, pp. 26–28, Washington, D.C., may 1993.
`http://www.almaden.ibm.com/cs/people/ragrawal/papers/sigmod93.ps`

[4] R. Agrawal, Manish Mehta, J. Shafer, Ramakrishnan Srikant, Andreas Arning e Toni Bollinger. "The Quest Data Mining System". In Simoudis et al. [93], pp. 244.

[5] R. Agrawal e Giuseppe Psaila. "Active Data Mining". In: *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, 1995.

[6] R. Agrawal, Giuseppe Psaila e Edward L. Wimmers ans Mohamed Zaït. "Querying Shapes of Histories". In: *Proceedings of the XXI VLDB Conference*, Zürich, Switzerland, 1995.

[7] R. Agrawal e R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". Jorgeesh Bocca, Matthias Jarke e Carlo Zaniolo editores, In: *Proceedings XX International Conference on Very Large DataBases*, pp. 487–499, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers, San Mateo, California.

[8] R. Agrawal e R. Srikant. "Mining Sequential Patterns". In: *Proceedings of the International Conference on Database Engineering*, pp. 3–14. ieee, 1995.

[9] R. Agrawal e J. C. Shafer. "Parallel mining of association rules". *Ieee Trans. On Knowledge And Data Engineering*, 8:962–969, 1996.

[10] S. S. Anand, B. W. Scotney, M. G. Tan, S. I. McClean, D. A. Bell e J. G. Hughes. "Designig a Kernel for Data Mining". *IEEE Expert*, 1996.

[11] R. J. Brachman e T. Anand. "The Process of Knowledge Discovery in Databases". U. Fayyad, G. Piatetsky-Shapiro, P Smyth e R. Uthurusamy editores, In: *Advances in Knowledge Discovery and Data Mining*, pp. 37–57. AAAI Press/The MIT Press, 1996.

[12] L. A. Breslow e D. W Aha. *Simplifying decision trees: A survey.* Technical Report AIC-96-14, NCARAI, 1997.

[13] F. Baião, M. Mattoso e G. Zaverucha. "Issues in Knowledge Discovery in Databases". *World Multiconference on Systemics, Cybernetics and Informatics*, 1998.

[14] F. Baião, M. Mattoso e G. Zaverucha. "A Knowledge-Based Perspective of the Distributed Design of Object Oriented Databases". In Ebecken [33], pp. 383–399.

[15] F. Bancilhon. *Object Database Systems: the ODMG Model.* Technical Report 12, O_2 Technology, 2685 Marine Way - Suite 1220 - Mountain View - CA 94043 - USA, 1994.

[16] L. Breiman, R. A. Friedman, J. H. Olshen e Stone C. J. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[17] J. P. Bigus. *Data Mining with Neural Networks: Solving Business Problems from Application Development to Decision Support.* McGraw-Hill, Inc., New York, 1996.

[18] J. Baranauskas e M. Monard. "Experimental Feature Subset Selection using the Wrapper Approach". In Ebecken [33], pp. 161–170.

[19] J. Catlett. *Megainduction: machine learning on very large databases.* Ph.D. thesis, University of Sidney, jun 1991.

[20] R. G. G. Cattell. *Object data management: object-oriented and extended relational database systems.* Addison-Wesley Publishing Company, Inc., 1994.

[21] P. Clark e R. Boswell. "Rule Induction with CN2: Some Recent Improvements.". Y. Kodratoff editor, In: *Machine Learning - EWSL-91*, pp. 151–163, Berlin, 1991. Springer-Verlag.

[22] R. G. G. Cattell, K. B. Barry e et al. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[23] S. Chaudhuri e U. Dayal. *An Overview of Data Warehousing and OLAP Technology.* Technical Report MSR-TR-97-14, Microsoft Research, Advanced Technology Division, mar 1997.

[24] Joe Celko. *Joe Celko's SQL for Smarties: Advanced SQL Programming.* Morgan Kaufmann Publishers, San Mateo, California, 1995.

[25] Chris Clifton e Don Marks. "Security and Privacy Implications of Data Mining". In: *Proceedings of the Workshop on Data Mining and Knowledge Discovery*, pp. 15–19, ACM SIGMOD, Montreal, Canada, jun 1996. University of British Columbia Department of Computer Science.

[26] P. Clark e T. Niblett. "The CN2 Induction Algorithm". *Machine Learning*, 3(4):261–283, 1989.

[27] M. Craven e J. Shavlik. "Using Neural Networks for Data Mining". *Future Generation Computer Systems special issue on Data Mining*, 1996.

[28] C. J. Date. *An Introduction to Database Sytems.* Addison-Wesley Publishing Company, Inc., 1995.

[29] Microsoft Group. *Microsoft Access Relational Database Menegment System for Windows, Language Reference - Functions, Statements, Methods, Properties, and Actions*, 1994.

[30] Linha de Bancos de Dados do COPPE. *GOA++'s Home Page.* 1999. `http://www.cos.ufrj.br/~goa`

[31] P. Domingos. "Rule Induction and Instance-Based Learning: A Unified Approach". In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1226–1232, Montreal, Canada, 1995. Morgan Kaufmann Publishers, San Mateo, California.

[32] P. Domingos. "Linear-Time Rule Induction". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 96–101, Portland, 1996. AAAI Press/The MIT Press.

[33] N. Ebecken editor,. *Proceedings of the First International Conference on Data Mining (ICDM-98).* WIT Press, 1998.

[34] U. Fayyad. "Editorial". U. Fayyad editor, In: *Data Mining And Knowledge Discovery*, volume 1, pp. 5–10. Kluwer Academic Publishers, Boston, 1997.

[35] U. Fayyad, David Hausler e Paul Stolorz. "KDD for Science Data Analysis: Issues and Examples". In Simoudis et al. [93].

[36] U. Fayyad, David Haussler e Paul Stolorz. "Mining Scientific Data". *Communications of the ACM*, 39(11):51–57, 1996.

[37] A. A. Freitas e S. H. Lavington. *A data-parallel primitive for high-performance knowledge discovery in large databases.* Technical Report CSM-242, University of Essex, UK, may 1995.

`ftp://ftp.essex.ac.uk/pub/csc/technical-reports/CSM-242.ps.Z`

[38] A. A. Freitas e S. H. Lavington. *Mining Very Large Databases with Parallel Processing.* Kluwer Academic Publishers, 1998.

[39] W. J. Frawley, G. Piatetsky-Shapiro e C. J. Matheus. "Knowledge discovery in databases: an overview". G. Piatetsky-Shapiro e W. J. Frawley editores, In: *Knowledge Discovery in Databases*, pp. 1–27, Menlo Park, CA/Cambridge, MA, 1991. AAAI Press/The MIT Press.

[40] U. Fayyad, G. Piatetski-Shapiro e P. Smyth. "From Data Mining to Knowledge Discovery: An Overview". U. Fayyad, G. Piatetsky-Shapiro, P Smyth e R. Uthurusamy editores, In: *Advances in Knowledge Discovery in Databases and Data Mining*, pp. 1–34. AAAI Press/The MIT Press, 1996.

[41] U. Fayyad, G. Piatetsky-Shapiro e P. Smyth. "From data mining to knowledge discovery in databases". *AI Magazine*, 17:37–54, 1996.

[42] U. Fayyad, G. Piatetsky-Shapiro e Padhraic Smyth. "The KDD Process for Extracting Useful Knowledge from Volumes of Data". *Communications of the ACM*, 39(11):27–34, 1996.

[43] U. Fayyad, G. Piatetsky-Shapiro e Padhraic Smyth. "Knowledge Discovery and Data Mining: Towards a Unifying Framework". In Simoudis et al. [93], pp. 82.

[44] A. A. Freitas. *Generic, Set-Oriented Primitives to Support Data-Parallel Knowledge Discovery in Relational Databases.* Ph.D. thesis, Essex University, Inglaterra, jul 1997.

[45] A. A. Freitas. "A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalized Rule Induction". *Genetic Programming 1997: Proceedings of the 2nd Annual Conference*, pp. 96–101, July 1997.

[46] A. A. Freitas. "Towards Large-Scale Knowledge Discovery in Databases (KDD) by Exploiting Parallelism in Generic KDD primitives". In: *Proceedings of the III International Workshop on Next-Generation Information Technologies and Systems*, pp. 33–43, Neve Ilan, Israel, jul 1997.

[47] A. GEIST e et al. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, Tennessee, EUA, 1994.

[48] G. Graefe, U. Fayyad e S. Chaudhuri. "On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases". R. Agrawal, P. Stolorz e G. Piatetsky-Shapiro editores, In: *Fourth International Conference on Knowledge Discovery and Data Mining (KDD98)*, pp. 204–208, Menlo Park, CA, 1998. AAAI Press/The MIT Press.

[49] Clark Glymour, David Madigan, Daryl Pregibon e Padhraic Smyth. "Statistical Inference and Data Mining". *Communications of the ACM*, 39(11):35–41, 1996.

[50] Clark Glymour, David Madigan, Daryl Pregibon e Padhraic Smyth. "Statistical Themes and Lessons for Data Mining". U. Fayyad editor, In: *Data Mining And Knowledge Discovery*, volume 1, pp. 11–28. Kluwer Academic Publishers, Boston, 1997.

[51] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillian College Publishing Company, New York, 1994.

[52] D. Heckerman. *A Tutorial on Learning With Bayesian Networks.* Technical Report MSR-TR-95-06, Microsoft Research, nov 1996.

[53] J. W. Han e Yongjian Fu. *Discovery of Multiple-Level Association Rules from Large Databases.* Technical Report TR 95-05, School of Computing Science, Simon Fraser University, March 1995.
`ftp://ftp.fas.sfu.ca/pub/cs/techreports/1995/CMPT95-05.ps.Z`

[54] J. W. Han, Y. Fu, W. Wang, K. Koperski e O. Zaiane. "DMQL: A Data Mining Query Language for Relational Databases". In: *Proceedings of the*

SIGMOD'96 Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada, jun 1996.

[55] J. H. Holland, Keith J. Holyoak, Richard E. Nisbett e Paul R. Thagard. *Induction: processes of inference, learning and discovery.* Computational models of cognition and perception. MIT Press, Cambridge, 1986.

[56] M. Holsheimer e M. Kersten. *Architectural Support for Data Mining.* Technical report, CWI Amsterdam, PO Box 94079, 1090 GB, Amsterdam, The Netherlands, 1994.

`ftp://ftp.cwi.nl/pub/CWIreports/AA/CS-R9429.ps.Z`

[57] C. Hsu e C. Knoblock. "Using Inductive Learning To Generate Rules for Semantic Query Optimization". U. Fayyad, G. Piatetsky-Shapiro, P Smyth e R. Uthurusamy editores, In: *Advances in Knowledge Discovery and Data Mining*, pp. 425–445. AAAI Press/The MIT Press, 1996.

[58] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, 1975.

[59] Peter J. Huber. "From Large to Huge: A Statistician's Reactions to KDD & DM". In: *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pp. 304, 1997.

[60] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* International Editions, 1993.

[61] T. Imielinski e H. Mannila. "A database perspective on knowledge discovery". *Communications of the ACM*, 39:58–64, 1996.

[62] W. H. Inmon. "The Data Warehouse and Data Mining". *Communications of the ACM*, 39(11):49–50, 1996.

[63] John Elder IV e Daryl Pregibon. "A Statistical Perpective on Knowledge Discovery in Databases". U. Fayyad, G. Piatetsky-Shapiro, P. Smyth e R. Uthu-

rusamy editores, In: *Advances in Knowledge Discovery and Data Mining*, pp. 83–113. AAAI Press/The MIT Press, 1996.

[64] T. Imielinski, Aashu Virmani e Amin Abdulghani. "DataMine: Application Programming Interface and Query Language for Database Mining". In Simoudis et al. [93], pp. 256–261.

[65] G. H. John e B. Lent. "SIPping from the Data Firehose". In: *Proceedings of the III International Conference on Knowledge Discovery and Data Mining*, pp. 199–202. AAAI Press/The MIT Press, 1997.

[66] G. H. John. *Enhancements to the Data Mining Process.* Ph.D. thesis, Department of Computer Science of Stanford University, mar 1997.

[67] Daniel A. Keim e Hans-Peter Kriegel. "Visualization techniques for Mining Large Databases: A Comparison". *IEEE Transactions on Knowledge and Data Engineering*, 8(6):923–938, December 1996.

[68] Jyrki Kivinen e H. Mannila. "The Power of Sampling in Knowledge Discovery". In: *Proceedings of the ACM SIGACT-SIGMOD-SIGACT Symposium on Principles of Database Theory (PODS'94)*, pp. 77–85, Minneapolis, MN, Mai 1994.

[69] H. Lu, R. Setiono e H. Liu. "NeuroRule: A Connectionist Approach to Data Mining". In: *Proceedings of the Proceedings of XXI VLDB Conference*, Zurique, Suiça, 1995.

[70] H. Mannila. "Data mining: machine learning, statistics and databases". In: *Proceedings of the 8th International Conference on Scientific and Statistical Database Management*, pp. 1–6, Stocolmo, 1996.

[71] R. C. Mauro. *Aspectos de Gerência de Objetos Persistentes.* M.Sc. thesis, COPPE - Universidade Federal do Rio de Janeiro, dez 1998.

[72] C. J. Matheus, P. K. Chan e G. Piatetsky-Shapiro. "Systems for Knowledge Discovery in Databases". *IEEE Trans. On Knowledge And Data Engineering*, 5:903–913, December 1993.

[73] R. C. Mauro e E. B. Silva et al. "GOA++: Tecnologia, Implementação e Extensões aos Serviços de Gerência de Objetos". In: *Proceedings of the XIII Brazilian Simposium on Databases*, pp. 272–286, Fortaleza, Ceará, oct 1997.

[74] *Understanding the new SQL - a complete guide.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[75] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., 1997.

[76] S. K. Murthy, S. Kasif e S Salzberg. "System for Induction of Oblique Decision Trees". *Journal of Artificial Inteligence Research*, 2:1–32, 1994.

[77] R. C. Mauro e M. L. Q. Mattoso. "Integração de LPOO e BDOO: Uma experiência com Java^{TM} e GOA++". In: *Proceedings of the XIII Brazilian Simposium on Databases (SBBD98)*, pp. 169–184, Maringá, Paraná, oct 1998.

[78] C. J. Merz e P. M. Murphy. *UCI Repository of Machine Learning.* 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[79] H. Mannila, Hannu Toivonen e A. Inkeri Verkamo. "Efficient algorithms for discovering association rules". U. Fayyad e Ramasamy Uthurusamy editores, In: *AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, pp. 181–192, Seattle, Washington, July 1994. AAAI Press/The MIT Press.

[80] H. Mannila, H. Toivonen e A. I. Verkamo. "Discovering Frequent Episodes in Sequences". U. Fayyad e R. Uthurusamy editores, In: *First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Canada, August 1995. AAAI Press/The MIT Press.

[81] E. S. Ogasawara e M. L. Q. Mattoso. *Análise de índices para bancos de dados orientados a objetos.* Technical Report ES-497/99, Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ, abr 1999.

[82] M. Pazzani. "When Prior Knowledge Hinders Learning". In: *Proceedings of the AAAI Workshop on Constraining Learning with Prior Knowledge*, San Jose, CA, 1992.

[83] G. Piatetsky-Shapiro e C. J. Matheus. "The Interestingness of Deviations". U. Fayyad e Ramasamy Uthurusamy editores, In: *AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, Seattle, Washington, July 1994. AAAI Press/The MIT Press.

[84] J. R. Quinlan. "Induction of decision trees". *Machine Learning*, 1(1):81–106, 1986.

[85] J. R. Quinlan. "Generating Production Rules from Decision Trees". In: *Proceedings of the X International Joint Conference on Artificial Inteligence*, pp. 304–307, San Mateo, CA, oct 1987.

[86] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[87] Tae-Wan Ryu e Christoph F. Eick. "Deriving Queries From Examples Using Genetic Programming". In Simoudis et al. [93], pp. 303–306.

[88] Tae-Wan Ryu e Christoph F. Eick. "MASSON: Discovering Commonalities in Collection of Objects using Genetic Programming". John R. Koza, David E. Goldberg, David B. Fogel e Rick L. Riolo editores, In: *Genetic Programming 1996: First Annual Conference*, pp. 200–208, Stanford University, CA, USA, jul 1996. MIT Press.
`http://www.cs.uh.edu/~twryu/papers/gp96.ps`

[89] N. J. Radcliffe e P. D. Surry. *Cooperation through Hierarchical Competition in Genetic Data Mining.* Parallel Computing Centre, Edinburgh, 1994

[90] J. Shafer, R. Agrawal e Manish Mehta. "SPRINT: A Scalable Parallel Classifier for Data Mining". In: *Proceedings of the XXII VLDB Conference*, pp. 544–555, Mombaim, India, oct 1996.

[91] C. Schaffer. "When Does Overfitting Decrease Prediction Accuracy in Induced Decision Trees and Rule Sets?". *EWSL*, 1991.

[92] C. Schaffer. "A Conservation Law for Generalization Performance". In: *Proceedings of the 11th Int. Conf. on Machine Learning*, pp. 259–265, 1994.

[93] E. Simoudis, J. W. Han e U. Fayyad editores,. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96).* AAAI Press/The MIT Press, 1996.

[94] E. Simoudis. "Reality check for data mining". *IEEE Expert-Intelligent Systems & Their Applications*, 11:26–33, 1996.

[95] W. M. Shen e B. Leng. "A metapattern-based automated discovery loop for integrated data mining - unsupervised learning of relational patterns". *IEEE Trans. On Knowledge And Data Engineering*, 8:898–910, December 1996.

[96] E. Simoudis, Brian Livezey e Randy Kerber. "Integrating Inductive and Deductive Reasoning for Data Mining". U. Fayyad, G. Piatetsky-Shapiro, P. Smyth e R. Uthurusamy editores, In: *Advances in Knowledge Discovery and Data Mining*, pp. 353–373. AAAI Press/The MIT Press, 1996.

[97] M. Sousa, M. Mattoso e N. Ebecken. "Data Mining: A Database Perspective". In Ebecken [33], pp. 413–431.

[98] M. Sousa. *Mineração de Dados: Uma Implementação Fortemente Acoplada a um Sistema Gerenciador de Banco de Dados Paralelo.* M.Sc. thesis, COPPE - Universidade Federal do Rio de Janeiro, ago 1998.

[99] Sarawagi S., Thomas S e Agrawal R. *Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications.* Technical report, IBM Research Division, 1998.

[100] A. Silberschatz, M. Stonebreaker e J. D. Ullman. "Database Research: Achievements and opportunities into the 21st Century". *NSF Workshop on the Future of Database Systems*, 1995.

[101] A. Silberschatz e A. Tuzhilin. "On subjective measures of interestingness in knowledge discovery". U. Fayyad e Ramasamy Uthurusamy editores, In: *First Int. Conf. on Knowledge Discovery and Data Mining (KDD-95)*, aug 1995.

[102] A. Silberschatz e A. Tuzhilin. "User-Assisted Knowledge Discovery: How Much Should the User Be Involved". *ACM-SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery*, jun 1996.

[103] A. Silberschatz e A. Tuzhilin. "What makes patterns interesting in knowledge discovery systems". *IEEE Transactions On Knowledge And Data Engineering*, 8:970–974, 1996.

[104] E. B. Silva e G. B. Xexéo. "Constructing Data Mining Functionalities in a DBMS". In Ebecken [33], pp. 367–381.

[105] J. D. Ullman. *Principles of Database and Knowledge-Base Systems. Volume 1: Classical Database Systems.* Addison-Wesley Publishing Company, Inc., New York, 1988.

[106] D. Wettschereck e D. W. Aha. "Weighting Features". In: *Proceedings of the I International Conference on Case-Based Reasoning*, Lisboa, Portugal, 1995. Springer-Verlag.

[107] S. M. Weiss e C. A. Kulikovski. *Computer Systems that Learn.* Morgan Kaufmann Publishers, San Mateo, California, 1991.

[108] S. Wrobel, Dietrich Wettschereck, A. Inkeri Verkamo, Arno Siebes, H. Mannila, Fred Kwakkel e Willi Klösgen. "User Interactivity in Very Large Scale Data Mining". W. Dilger, M. Schlosser, J. Zeidler e A. Ittner editores, In: *Proc. FGML-96 (Annual Meeting of the GI Special Interest Group Machine Learning)*, pp. 125–130, 09111 Chemnitz, August 1996. TU Chemnitz-Zwickau.

[109] J. P. Yoon e L. Kerschberg. *A Framework for Knowledge Discovery and Evolution in Databases.* Technical report, George Mason University, ISSE, jul 1994.
`ftp://isse.gmu.edu/pub/techrep/by_index/ISSE-TR-93-109.ps.Z`