UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Um algoritmo para indução de árvores e regras de decisão

por

CARINE HALMENSCHLAGER

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do
grau de Mestre em Ciência da Computação

Prof. Luis Otavio Campos Alvares
Orientador

Porto Alegre, abril de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Halmenschlager, Carine
Um algoritmo para indução de árvores e regras de decisão / por Carine Halmenschlager. – Porto Alegre : PPGC da UFRGS, 2002.
112 p. : il.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2002. Orientador: Alvares, Luis Otavio Campos.

1. Inteligência Artificial. 2. Descoberta de Conhecimento em Bases de Dados. 3. Mineração de Dados. 4. Árvores de Decisão. 5. Regras de Classificação. I. Alvares, Luis Otavio Campos. II. Título.

A ti, RØSP!,
com amor.

Agradecimentos

Aos meus pais, Alcides e Leonata Halmenschlager, pela confiança e amor que sempre depositaram em mim, pelo exemplo de honestidade, pelo apoio e carinho que sempre prestaram e pelos cuidados com o Scooby :).

Aos meus adoráveis sobrinhos, Maiara, Maurício, Tainara e Mayle, dos quais fiquei muitas vezes afastada e distante, para dedicar-me a este trabalho.

Ao meu namorado, Rosp!, pelo intenso amor, carinho, alegrias e sonhos compartilhados, bem como pela paciência e motivação de seus gestos sempre amparadores e encorajadores. Já te disse?!? Ah, e ao Billy, pelos momentos de descontração!

À minha "irmanzona", Fabi, pela nossa grande amizade e por tornar nossa convivência algo tão sereno e agradável, bem como por todo o apoio nos momentos mais difíceis.

À minha nova família, em especial à Lô e ao Larry, por terem me acolhido com tanto carinho e afeição.

Ao meu orientador, Prof. Luis Otavio, pelo auxílio, dedicação, amizade e pelos imprescindíveis caminhos apontados, sem os quais este trabalho não se completaria. Também ao Prof. Rodrigo Martínez-Bejár, da Universidad de Murcia, pela amizade e permanente incentivo.

Às professoras e amigas, Rejane e Duda, pela motivação ao ingresso no mestrado e pelo precioso auxílio nas primeiras semanas em Porto Alegre.

Aos colegas do Instituto de Informática, Filipe, João, Dani e João Paulo, pelas experiências e ajudas compartilhadas, bem como pela amizade iniciada. Em especial ao meu bolsista, Luis Henrique, pela inestimável ajuda profissional.

A todos meus amigos, que souberam compreender o motivo de minha ausência e distância nestes últimos tempos.

Ao CNPq, pelo suporte financeiro que permitiu a minha dedicação integral ao mestrado, e ao Instituto de Informática e seus profissionais, pelo apoio e suporte prestados.

A todos que, de uma forma ou outra, contribuíram para o sucesso deste trabalho.

Sumário

Lista de Abreviaturas

AD	Árvore de Decisão
AIH	Autorização de Internação Hospitalar
CART	*Classification And Regression Trees*
CENEPI	Centro Nacional de Epidemiologia
DCBD	Descoberta de Conhecimento em Bases de Dados
FDN	Forma Disjuntiva Normal
IAD	Inteligência Artificial Distribuída
JAM	*Java Agents for Meta-Learning*
PADMA	*Parallel Data Mining Agents*
RDP	Resolução Distribuída de Problemas
SES	Secretaria da Saúde do Estado do Rio Grande do Sul
SGBD	Sistema de Gerenciamento de Bases de Dados
SIM	Sistema de Informações sobre Mortalidade
SMA	Sistemas Multiagentes
SQL	*Structured Query Language*
SUS	Sistema Único de Saúde
TDIDT	*Top Down Induction of Decision Trees*

Lista de Figuras

Lista de Tabelas

Resumo

A classificação é uma das tarefas da Mineração de Dados. Esta consiste na aplicação de algoritmos específicos para produzir uma enumeração particular de padrões. Já a classificação é o processo de gerar uma descrição, ou um modelo, para cada classe a partir de um conjunto de exemplos dados. Os métodos adequados e mais utilizados para induzir estes modelos, ou classificadores, são as árvores de decisão e as regras de classificação.

As regras e árvores de decisão são populares, principalmente, por sua simplicidade, flexibilidade e interpretabilidade. Entretanto, como a maioria dos algoritmos de indução particionam recursivamente os dados, o processamento pode tornar-se demorado, e a árvore construída pode ser muito grande e complexa, propensa ao *overfitting* dos dados, que ocorre quando o modelo aprende detalhadamente ao invés de generalizar. Os conjuntos de dados reais para aplicação em Mineração de Dados são, atualmente, muito grandes, e envolvem vários milhares de registros, sendo necessária, também, uma forma de generalizar estes dados.

Este trabalho apresenta um novo modelo de indução de classificadores, em que o principal diferencial do algoritmo proposto é a única passada pelo conjunto de treinamento durante o processo de indução, bem como a sua inspiração proveniente de um Sistema Multiagente. Foi desenvolvido um protótipo, o Midas, que foi validado e avaliado com dados de repositórios. O protótipo também foi aplicado em bases de dados reais, com o objetivo de generalizar as mesmas.

Inicialmente, foi estudado e revisado o tema de Descoberta de Conhecimento em Bases de Dados, com ênfase nas técnicas e métodos de Mineração de Dados. Neste trabalho, também são apresentadas, com detalhes, as árvores e regras de decisão, com suas técnicas e algoritmos mais conhecidos. Finalizando, o algoritmo proposto e o protótipo desenvolvido são apresentados, bem como os resultados provenientes da validação e aplicação do mesmo.

Palavras-chaves: Inteligência Artificial, Descoberta de Conhecimento, Mineração de Dados, Árvores de Decisão, Regras de Classificação

TITLE: "AN ALGORITHM FOR INDUCTION OF DECISION TREE AND CLASSIFICATION RULES"

Abstract

Data Mining consists of the application of specific algorithms to produce pattern enumeration. One of the main tasks in Data Mining is classification, which can be defined as the process of generating a description or a model for each class of a group of given examples. Decision trees and classification rules can be said to be the most appropriate and used methods to induce these models, also called classifiers.

Decision trees and classification rules have become popular among researchers in Artificial Intelligence due mainly to their all speed, flexibility and interpretation characteristics. However, most of the induction algorithms partition the data recursively, so that the process can result in a very slow one, while the built tree can be very big, complex or overfitting-biased. This happens when the model learns in full detail instead of generalizing. Currently, there are numerous data sets for which data mining techniques are applicable. These data involve millions of records, so that a form of generalizing them has arisen as an strategic endeavor for many organizations from all around the world.

This work presents a new classifier induction model, which the following main characteristics: (1) it only crosses once the training group during the induction process, and (2) it is Multiagent System philosophy-inspired. Based on this model, a software prototype Midas has been developed and validated with repository data. Moreover, the prototype has also been applied in a real database.

Keywords: Artificial Intelligence, Knowledge Discovery, Data Mining, Decision Trees, Classification Rules

1 Introdução

A cada ano, as empresas acumulam centenas de informações em suas bases de dados, o que aumenta expressivamente o volume dos dados e a riqueza de suas informações. Como resultado deste aumento efetivo, o processamento destas informações tornou-se cada vez mais complexo e difícil, e, normalmente, os dados ficam armazenados nas bases de dados sem que sejam utilizados de uma forma mais eficiente [AVI 98].

Com o avanço da tecnologia e a necessidade de métodos mais poderosos para a recuperação e utilização da informação, surgiu a área de Descoberta de Conhecimento em Bases de Dados, que revela a informação estratégica oculta nos grandes bancos de dados [FAY 96].

A Descoberta de Conhecimento em Bases de Dados consiste basicamente em descobrir conhecimento útil nos dados armazenados. Isso ocorre a partir da aplicação de técnicas modernas de Mineração de Dados, da avaliação dos padrões obtidos e da interpretação dos resultados [HAL 2001].

As técnicas de Mineração de Dados operam sobre grandes volumes de dados extraindo informações implícitas e padrões nos dados, que não podem ser descobertos utilizando as técnicas convencionais de inferência em banco de dados [FEL 97]. Entre as técnicas de Mineração de Dados mais conhecidas e utilizadas, pode-se citar as Árvores de Decisão, as Regras de Classificação, as Regras de Associação e as Redes Neurais.

As árvores de decisão e as regras de classificação são consideradas métodos simbólicos que representam, através de expressões, o que é aprendido sobre os atributos dados. Já nas redes neurais, que são métodos conexionistas, o aprendizado consiste em ajustar pesos em uma rede [QUI 94]. Devido ao alto grau de legibilidade dos métodos simbólicos, as árvores de decisão e as regras de classificação são muito utilizadas para a geração de classificadores, que, por sua vez, são utilizados para atribuir um rótulo a um novo objeto cuja classe é desconhecida.

Uma árvore de decisão classifica exemplos de uma base de dados em um número finito de classes, possuindo uma forma simples de representação [BAR 2000a]. Em uma árvore de decisão, os nodos (nós[1]) da árvore representam os nomes de atributos, as ligações (arcos ou ramos) representam possíveis valores para o atributo e as folhas representam as diferentes classes. Um objeto é classificado seguindo o caminho da raiz da árvore até uma folha, enquanto as suas características satisfizerem os nodos e suas ligações.

[1] Termo também equivalente e bastante freqüente na literatura. Porém, neste trabalho será adotado apenas o termo *nodo,* por motivos de padronização do texto.

As árvores de decisão podem ser muito bem aplicadas a grandes conjuntos de dados e são adequadas para qualquer tipo de dados (discretos ou contínuos). Outra vantagem é que elas são fáceis de serem entendidas, e o resultado do algoritmo pode ser usado diretamente pelo usuário [ADR 97].

Os algoritmos de indução de árvores de decisão mais conhecidos e utilizados são o CART, de Breiman apud [FON 94], o ID3 [QUI 88] e seu derivado, o C4.5 [QUI 93], ambos de Quinlan – ainda que existam vários outros, como o HistClass, de Fonseca [FON 94] e OC1, de Murthy [MUR 97]. Embora a maioria destes algoritmos se baseie na estratégia de "dividir para conquistar", existem diferentes técnicas utilizadas na construção dos mesmos, que são determinantes para a sua eficiência. Entre eles, pode-se destacar os métodos usados para: (I) a escolha do atributo a utilizar em cada nodo; (II) a definição da partição de cada nodo; (III) a decisão de quando o nodo é uma folha e de qual classe atribuir a esta folha; e (IV) a definição de técnicas de poda [FON 94].

Segundo Fonseca [FON 94] e Baranauskas [BAR 2000a], entre os critérios disponíveis para a escolha do atributo a utilizar como nodo estão o método da entropia, o critério de Gini, o método da paridade, a técnica de Laplace e também a escolha randômica.

Já as técnicas para particionamento das ligações de cada nodo variam conforme o tipo do atributo do nodo em questão [FON 94]. Para definir o particionamento de atributos discretos, pode-se utilizar critérios como: uma ligação para cada valor do atributo, criação de nodos binários, ou agrupamento de valores do atributo em vários grupos. Já para ligações dos nodos de atributos contínuos, pode-se utilizar métodos como: testes simples (em que é escolhido o ponto de cisão considerado de maior valor pelo critério de avaliação adotado), testes múltiplos (em que definem-se múltiplos intervalos de partição dos valores do atributo), e combinação linear de características.

Ainda segundo Fonseca [FON 94], a definição de qual classe atribuir a uma folha pode ser feita pela atribuição da classe de maior probabilidade e pela determinação baseada na noção de custos.

A limitação e redução do tamanho da árvore pode ser feita, respectivamente, através das abordagens de pré-poda e pós-poda [ESP 97]. A pré-poda pode ser feita com a parada da geração da árvore baseada na informação mútua, no teste de independência, entre outras. Já a pós-poda, realizada depois de induzida a árvore, pode ser realizada, entre outros, com base no algoritmo Laplaciano, no erro, na minimização do custo/complexidade, ou segundo o modelo de Guo e Gelfand.

Todos esses aspectos devem ser cuidadosamente analisados no momento de desenvolver um sistema deste tipo, de modo a maximizar a eficácia e a qualidade da árvore, e de otimizar a performance durante a sua construção.

Os algoritmos de indução de árvores de decisão são relativamente simples. Porém, em casos nos quais o volume da base de dados é muito grande, o processamento torna-se demorado, e o tamanho da árvore construída pode ser muito grande, dificultando a representação gráfica e o entendimento de toda a sua estrutura. Uma forma de resolver isso é utilizando regras de classificação.

As regras de classificação, ou regras de decisão, são estruturas do tipo *"Se <condição> então <conclusão>"*, em que <condição> é um conjunto de atributos e seus valores e <conclusão> é uma classe do conjunto de dados.

As regras de decisão podem ser geradas diretamente a partir de um conjunto de dados, utilizando um algoritmo de indução de regras, ou a partir de uma árvore de decisão, traduzindo os conceitos aprendidos pela árvore de uma forma mais compreensível e equivalente. Assim, as regras resolvem a maior desvantagem de árvores de decisão, que é a dificuldade de compreensão quando aplicadas em grandes bases de dados, que tendem a aumentar as árvores exageradamente.

Os algoritmos de indução de regras mais conhecidos e utilizados são o AQ1, de Michalski [MIC 69], o CN2, de Clark e Niblett [CLA 89], e o PRISM, de Centrowska [CEN 88].

Dentro desse contexto, o objetivo principal deste trabalho é apresentar um novo modelo para a indução de árvores e regras de decisão para uso na Descoberta de Conhecimento em Bases de Dados, inspirado no paradigma de agentes. O principal diferencial do algoritmo proposto é a única passada pelo conjunto de treinamento durante a indução do classificador, com o objetivo de generalizar rapidamente os dados. Este modelo foi implementado, validando o protótipo desenvolvido com dados de repositórios e aplicando-o em bases de dados reais.

Os fundamentos da área de Descoberta de Conhecimento em Bases de Dados e de Mineração de Dados são apresentados no segundo capítulo, bem como as etapas e funcionamento do processo, os desafios e as aplicações da área.

No terceiro capítulo, são apresentados os conceitos e as características principais das árvores de decisão e regras de classificação, seguido dos métodos e técnicas de construção e avaliação de classificadores. Ainda, são apresentados os principais algoritmos de indução de classificadores (CART, C4.5, CN2 e PRISM) e as principais formas de comparação de algoritmos.

No quarto capítulo, é apresentado o Sistema Multiagente que serviu de inspiração para a abordagem utilizada neste trabalho, bem como o modelo proposto para a indução de classificadores. Também o algoritmo e o protótipo desenvolvido são apresentados detalhadamente neste capítulo. É apresentado, ainda, a validação do protótipo desenvolvido com comparações do desempenho e tamanho da árvore gerada, assim como das regras induzidas – utilizando, para isso, dados do repositório UCI [UCI 2000] e a ferramenta See5 [DAT 2001].

Neste capítulo também é apresentada a aplicação do protótipo em bases de dados da Secretaria da Saúde do Estado do Rio Grande do Sul, descrevendo com detalhes os dados utilizados na mineração, algumas regras descobertas e uma avaliação preliminar dos resultados.

Finalmente, são apresentadas as contribuições, considerações finais e trabalhos futuros no sexto capítulo.

2 Descoberta de Conhecimento

A área de Descoberta de Conhecimento surgiu em torno de 1981, quando a Technical University Berlin, a partir da idéia de adquirir regras de inferência para uma linguagem natural através da modelagem baseada na aprendizagem, desenvolveu o sistema METAXA, que fazia parte do projeto LERNER, relacionado ao conceito de Aprendizado de Máquina. Nesse projeto foi desenvolvida uma nova metodologia, a *Balanced Cooperative Modeling* (Modelagem de Balanceamento Cooperativo), que formou a base para a construção de um novo tipo de sistema, integrando a aquisição de conhecimento a diferentes algoritmos de aprendizagem [MOR 93].

Inicialmente, foram determinados vários nomes ao processo de achar padrões úteis em bases de dados, como Extração de Conhecimento, Descoberta de Informação, Mineração de Dados e Processamento do Padrão de Dados. Apenas em 1989, durante o primeiro *workshop* de Descoberta de Conhecimento, Piatetsky-Shapiro utilizou o termo *Descoberta de Conhecimento em Bases de Dados* para se referir ao processo total de procura do conhecimento em dados, enfatizando a aplicação de alto-nível de técnicas de Mineração de Dados [FAY 96].

A Descoberta de Conhecimento em Bases de Dados (DCBD) é "um processo não trivial de identificar padrões válidos, não conhecidos, potencialmente úteis e interpretáveis" [FAY 96], consistindo, basicamente, em descobrir o conhecimento útil nos dados armazenados, a partir da aplicação de técnicas modernas de Mineração de Dados, da avaliação dos padrões obtidos e da interpretação dos resultados.

Mineração de Dados, ou *Data Mining,* é uma das etapas do processo total de Descoberta de Conhecimento, que compreende todo o processo de descoberta de dados, desde o desenvolvimento da ferramenta até a representação do conhecimento adquirido para o usuário. Enquanto isso, a Mineração de Dados refere-se somente à aplicação de algoritmos, que operam sobre grandes volumes de dados, extraindo informações implícitas destes dados [HAL 2000].

As ferramentas convencionais utilizadas em bases de dados têm a capacidade de retornar respostas sobre consultas, realizadas através de uma linguagem como SQL[2]. Entretanto, essa informação resultante não possui uma importância expressiva e dificilmente poderia ser fundamental na tomada de importantes decisões de uma empresa [AVI 98]. Em contrapartida, a Descoberta de Conhecimento é uma das áreas que investe no desenvolvimento de ferramentas mais modernas para a recuperação e interpretação das informações, através de sistemas capazes de extrair o conhecimento das bases de dados [FEL 97].

Os sistemas de Descoberta de Conhecimento se relacionam às áreas de Aprendizado de Máquina, Reconhecimento de Padrões, Inteligência Artificial, Banco de Dados, Visualização de Conhecimento e Estatística, utilizando técnicas, métodos e algoritmos dessas áreas para extrair o conhecimento das bases de dados.

[2] Linguagem estruturada utilizada para realizar consultas em bases de dados.

Alguns parâmetros que são utilizados para representar os elementos do processo de DCBD estão descritos a seguir [FAY 96]:

a) *dados:* é o conjunto de fatos, casos ou exemplos da base de dados;

b) *padrão*: é uma expressão *E* em uma linguagem *L*, que descreve casos em um subconjunto *FE* do conjunto de fatos *F*. A expressão *E* é chamada padrão se ela é mais simples do que a enumeração de todos os fatos em *FE;*

c) *validade:* é o grau de certeza dos padrões descobertos. Uma medida de certeza é uma função *C* que mapeia expressões em uma linguagem *L*. Uma expressão *E* em *L*, sobre um subconjunto $FE \subset F$, pode ter uma medida dada por $c = C(E, F)$;

d) *novidade:* são os novos padrões. A novidade pode ser medida por uma função $n = N(E, F)$, que pode ser uma função *booleana* ou uma medida que retorne algum grau de novidade, através da comparação dos valores obtidos com os valores esperados, ou com os valores obtidos anteriormente;

e) *utilidade:* são os padrões potencialmente úteis para alguma ação e que podem ser medidos por uma função de utilidade do tipo $u = U(E, F)$;

f) *interpretabilidade:* é tornar os padrões compreensíveis para o usuário final, facilitando o entendimento dos dados. Essa característica é difícil de medir precisamente, mas, geralmente, a simplicidade é expressa através da função $s = S(E, F)$;

g) *informação:* é o dado que possui um significado;

h) *conhecimento*: é a informação obtida através de análise dos diversos parâmetros identificados durante o processo de Descoberta de Conhecimento, ou seja, conhecimento é um padrão $E \in L$, se $I(E, F, C, N, U, S) > i$, onde *i* é a informação anterior.

2.1 Processo de Descoberta de Conhecimento

O processo de Descoberta de Conhecimento envolve várias etapas complexas, como a preparação dos dados, a busca por padrões, a avaliação e o refinamento do conhecimento. Todas as etapas devem ser executadas cuidadosamente, pois cada etapa é fundamental para que os objetivos estabelecidos e o sucesso completo da aplicação sejam alcançados.

Segundo John [JOH 97], existem duas partes envolvidas neste processo: o analista de Mineração de Dados e o especialista do domínio. Geralmente, o problema a ser resolvido é muito bem entendido pelo especialista, assim, o processo de DCBD inicia com este explicando o problema ao analista. Isto permite a troca de conhecimentos entre ambos, que é essencial ao bom andamento do processo, pois os especialistas, em geral, não possuem conhecimento das técnicas ou métodos de Descoberta de Conhecimento, mas são grandes conhecedores do domínio.

O processo de Descoberta de Conhecimento é *interativo*, com muitas decisões a serem tomadas, mas que devem depender o mínimo possível da intervenção do usuário, executando da forma mais automatizada possível. O processo é também *iterativo*, podendo possuir laços entre quaisquer das etapas, não existindo uma ordem ou seqüência única durante o andamento do processo [HAL 2000]. O processo de Descoberta de Conhecimento é ilustrado na Figura 2.1.

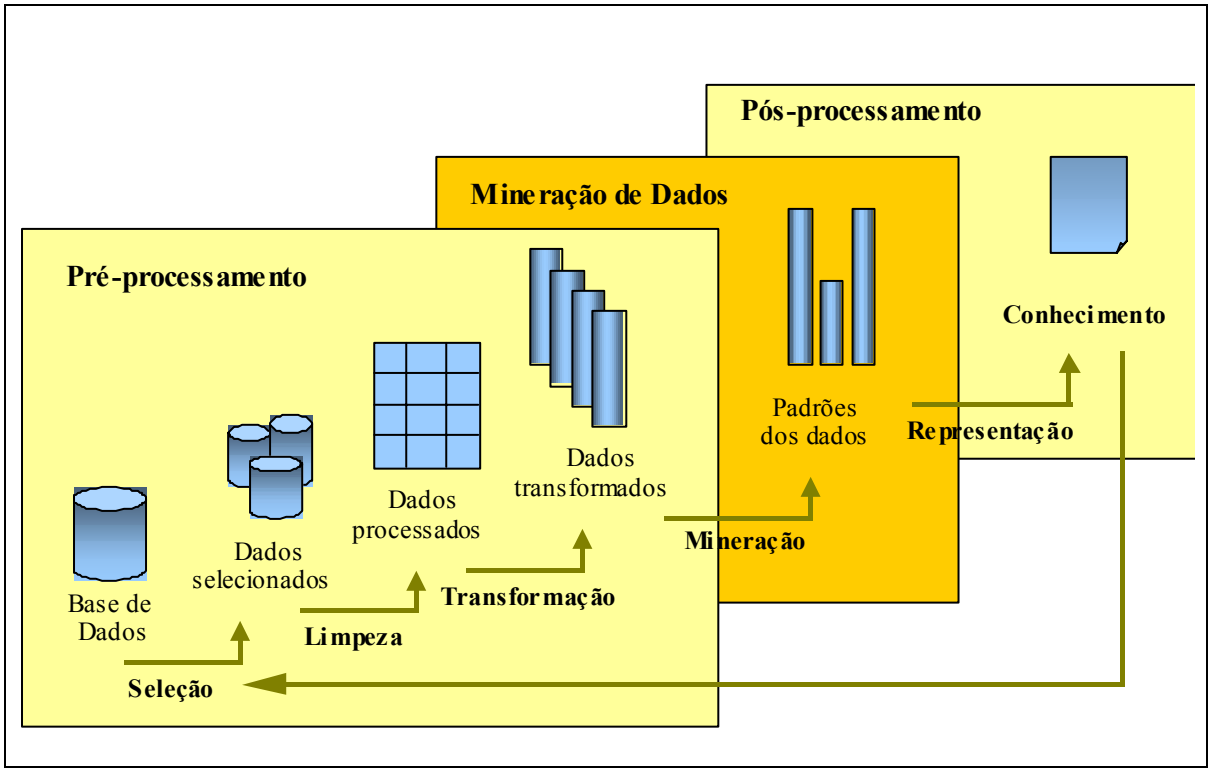

FIGURA 2.1 – Etapas do processo de DCBD
Fonte: FAYYAD, 1996. p.10

Pode-se dividir os passos do processo de Descoberta de Conhecimento (vistos na Figura 2.1) em três etapas principais: Pré-processamento, Mineração de Dados e Pós-processamento. A seguir, é apresentada uma descrição de cada uma das etapas, segundo Fayyad [FAY 96] e John [JOH 97].

2.1.1 Pré-processamento

No Pré-processamento são realizados os seguintes passos:

a) *desenvolvimento de um entendimento do domínio da aplicação:* aquisição do conhecimento prévio e das metas do usuário final, identificando a informação requerida para uma ação específica, ou seja, o que o usuário quer conhecer e o que ele quer fazer com esse conhecimento. Isto significa que o especialista e o analista deverão trabalhar juntos para especificar o problema a ser resolvido. O domínio da aplicação envolve também a elaboração de uma lista de requisitos; a avaliação do hardware, do software e da qualidade dos dados disponíveis; o desenvolvimento de um inventário das bases de dados disponíveis; a formulação do conhecimento necessário à organização no momento e no futuro; a identificação das pessoas que trabalham com o conhecimento, o tipo de decisões que elas costumam fazer, os padrões que elas utilizam e a funcionalidade que elas necessitam como apoio ao processo de decisão; e a identificação dos processos e transformações que as bases de dados terão que sofrer antes de serem utilizadas;

b) *seleção dos dados:* criação de um conjunto de exemplos a partir de uma cópia dos dados operacionais armazenados nas bases de dados da organização, nos quais a descoberta será realizada. Isto envolve a definição dos atributos que serão importantes e utilizados para a descoberta, bem como a recuperação dos exemplos necessários. Entretanto, extrair e integrar dados nem sempre é uma tarefa fácil, pois pode envolver, além da conversão em baixo nível dos dados, a união de tabelas relacionais, pois o algoritmo de mineração não consegue trabalhar com múltiplas tabelas;

c) *limpeza dos dados:* definição de quais estratégias serão utilizadas para a manipulação de informações inconsistentes, coleta de informações necessárias ao modelo e realização de operações básicas de limpeza, como remoção de redundâncias, inconsistências e valores nulos. Entretanto, para efetuar a limpeza dos dados é essencial e necessário, primeiramente, explorar os dados de forma a se familiarizar com os mesmos;

d) *transformação dos dados:* redução e projeção dos dados, encontrando características comuns para representá-los e reduzindo, assim, o número efetivo de valores de variáveis em consideração. Isto pode significar expressar alguma informação da base de dados de uma forma diferente da original.

2.1.2 Mineração de Dados

Na etapa de Mineração de Dados, têm-se os seguintes passos:

a) *escolha da tarefa de mineração de dados:* decisão de qual tarefa será utilizada: classificação, associação, regressão, agrupamento, sumarização, dependência ou desvio (descritos na seção 2.3);

b) *escolha do algoritmo de mineração de dados:* seleção da(s) técnica(s) a ser(em) utilizada(s) na próxima etapa e decisão de quais modelos e parâmetros podem ser apropriados. As técnicas mais difundidas e utilizadas são: árvores de decisão, regras de classificação, redes neurais e algoritmos genéticos;

c) *aplicação da mineração de dados:* busca por padrões de interesse particular em uma forma representacional ou em um conjunto de representações, a partir da base customizada na etapa de pré-processamento. Pode ser utilizado, também, outro conjunto independente de dados para avaliação do modelo final.

2.1.3 Pós-processamento

O Pós-processamento possui as seguintes etapas:

a) *interpretação dos padrões enumerados:* avaliação e interpretação dos padrões encontrados para verificar o que constitui e o que não constitui conhecimento útil. Nesta etapa pode ocorrer a necessidade de retorno a uma das etapas anteriores;

b) *consolidação da descoberta de conhecimento:* incorporação do conhecimento adquirido à organização, documentando-o e relatando-o aos usuários através de técnicas de visualização de dados e implementando alguma estratégia baseada na descoberta. Nesta etapa deve-se verificar e resolver eventuais conflitos com o conhecimento extraído.

2.2 Funcionamento de um sistema de Descoberta de Conhecimento

O funcionamento de um sistema capaz de descobrir conhecimento em bases de dados é mostrado na Figura 2.2 [FEL 97].

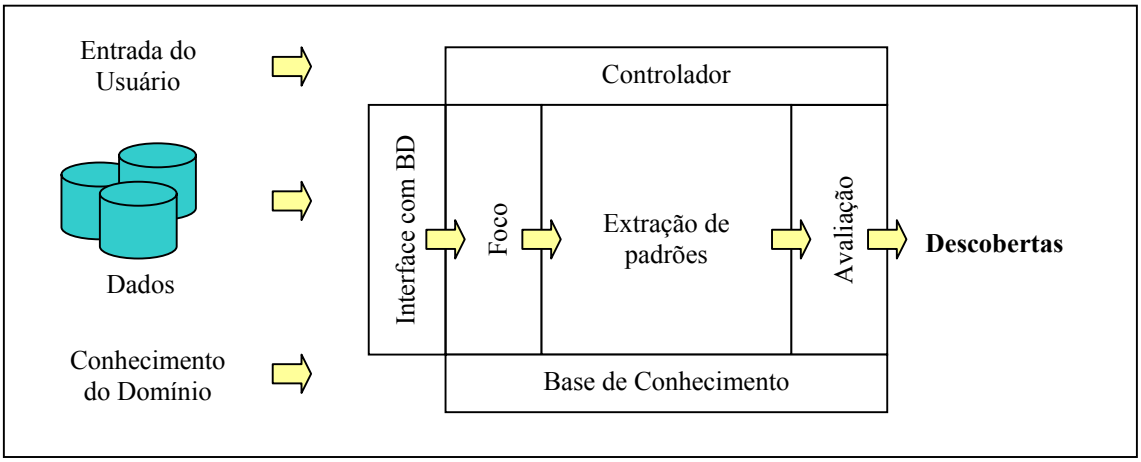

FIGURA 2.2 – Arquitetura de um sistema de DCBD
Fonte: FELDENS, 1997. p.17

Na Figura 2.2, identificam-se as seguintes entradas de informação:

a) *entrada do usuário:* dada através de uma linguagem de alto nível, sendo que o tipo de informação fornecido pelo usuário depende de cada sistema;

b) *fonte de dados:* fornecida pelo Sistema de Gerenciamento de Bases de Dados (SGBD);

c) *conhecimento do domínio:* passado ao sistema através de regras já conhecidas ou conhecimentos já adquiridos. Alguns algoritmos de aprendizagem utilizam este conhecimento para guiar e otimizar o processo de busca.

Identificam-se também, na Figura 2.2, os seguintes módulos:

a) *controlador:* faz o interfaceamento entre os demais módulos, tendo autonomia para disparar uma série de algoritmos, selecionar estratégias, orientar o foco de busca e avaliar as descobertas. Geralmente é o usuário quem desempenha o papel do controlador;

b) *base de conhecimento:* contém o conhecimento do domínio, tanto o fornecido pelo usuário, quanto o descoberto durante o processo;

c) *interface com a base de dados:* responsável pela comunicação do sistema de Descoberta de Conhecimento com a base de dados;

d) *foco:* extrai amostras das bases de forma aleatória ou conforme alguma restrição informada pelo usuário, indicando, assim, quais atributos e registros da base de dados serão analisados;

e) *extração de padrões:* núcleo do sistema, onde estão os algoritmos de mineração de dados responsáveis pela extração de padrões;

f) *avaliação:* avalia se os padrões encontrados são interessantes e úteis.

Assim, após todo o processamento, o resultado obtido como saída serão as descobertas, ou seja, padrões potencialmente válidos, novos, úteis e interpretáveis.

2.3 Mineração de Dados

Como visto anteriormente, a Mineração de Dados é um processo da Descoberta de Conhecimento, que "consiste na aplicação de algoritmos específicos, sob alguma limitação aceitável de eficiência computacional, para produzir uma enumeração particular de padrões" [FAY 96].

Esses algoritmos utilizam técnicas de aprendizado indutivo sobre bases de dados, sendo capazes de extrair conhecimento através de exemplos, envolvendo repetidas aplicações iterativas dos métodos.

O aprendizado indutivo consiste na criação de um modelo. Durante o processo de aprendizado, o sistema cognitivo observa e reconhece similaridades entre objetos e eventos do ambiente, agrupando objetos similares em classes e construindo regras que fornecem o comportamento dos exemplos de cada classe [AVI 98].

Existem dois tipos de técnicas de aprendizado indutivo:

a) *aprendizado supervisionado,* ou *aprendizado de exemplos,* no qual são fornecidas as classes e exemplos de cada classe ao sistema. Este, por sua vez, precisa descobrir a descrição da classe, ou seja, as propriedades comuns nos exemplos de cada classe;

b) *aprendizado não-supervisionado,* ou *aprendizado por observação,* no qual o sistema precisa descobrir a própria classe, baseando-se nas propriedades comuns dos objetos.

A busca por descrições no aprendizado indutivo é considerada Mineração de Dados quando o conjunto de treinamento (classes e exemplos dados) for uma base de dados ou uma amostra da mesma.

Existem duas metas primárias, que podem ser alcançadas através da Mineração de Dados: a *previsão,* que utiliza algumas variáveis para prever valores desconhecidos ou futuros; e a *descrição*, que procura por padrões que descrevem os dados e que sejam interpretáveis pelos seres humanos.

As metas de previsão e descrição em Mineração de Dados são alcançadas utilizando-se *tarefas básicas*. Segundo Ávila [AVI 98], "uma das dificuldades existentes é que a variedade de algoritmos para a solução de cada uma dessas tarefas confunde tanto experientes analistas de dados quanto os mais novatos".

A seguir são apresentadas as tarefas de mineração de dados, segundo Fayyad [FAY 96] e John [JOH 97]:

a) *classificação:* é o aprendizado supervisionado de uma função que classifica, ou seja, atribui um rótulo a um item de dado dentro de várias classes estabelecidas previamente. Os métodos de classificação podem ser utilizados para identificar objetos de interesse em grandes bases de dados, sendo as árvores e regras de decisão, bem como as redes neurais, técnicas que podem ser usadas para este propósito. A Figura 2.3 mostra a utilização da classificação para decidir automaticamente se empréstimos serão ou não realizados, através de um simples particionamento de dados de empréstimos bancários em duas regiões de classes, ou seja, usando os atributos como elementos discriminantes para dividir os dados em subconjuntos;

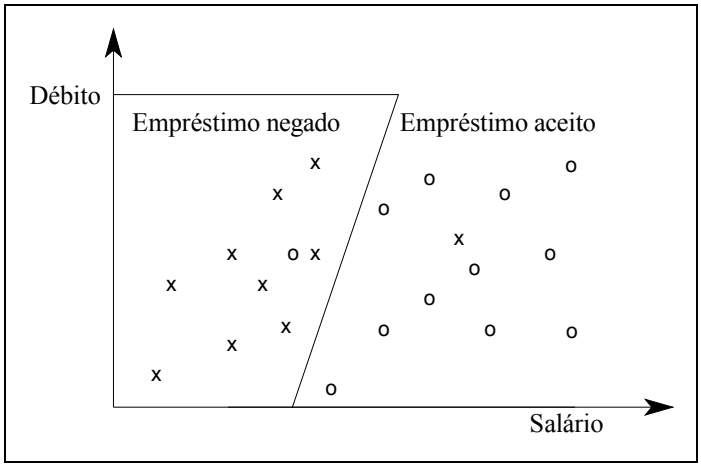

FIGURA 2.3 – Exemplo de classificação

b) *associação:* são padrões informativos na forma $X \rightarrow Y$, em que X e Y são conjuntos de itens. São pesquisadas todas as possíveis regras $X \rightarrow Y$, representando estruturas de valores dos atributos de um registro. Esta regra significa que as transações da base de dados que contêm X tendem a conter também Y. Exemplo: 90% das pessoas que compram pão também compram leite na mesma operação;

c) *regressão linear:* é o aprendizado de uma função que mapeia um item de dado para uma variável de valor real. É similar à classificação, exceto pelo valor contínuo do atributo categórico, ao invés de nominal, como ocorre na classificação. Os métodos de regressão linear permitem a discriminação dos dados através da combinação dos atributos de entrada, o que equivale a determinar retas de separação dos dados. A Figura 2.4 mostra os resultados de regressão linear simples, onde débito é considerado como uma função linear de salário;

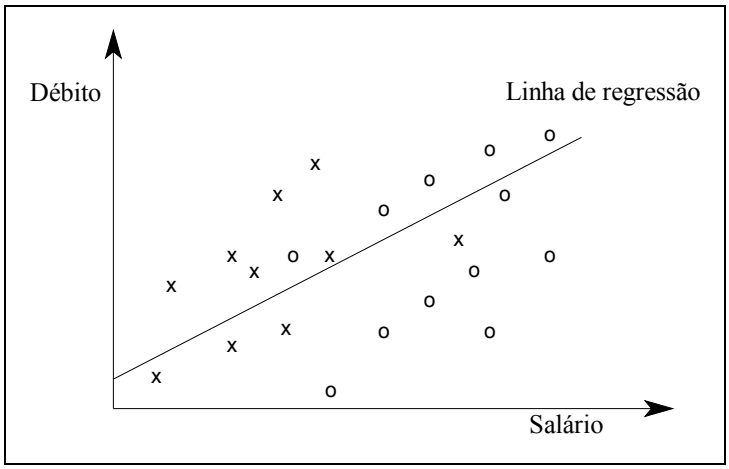

FIGURA 2.4 – Exemplo de regressão linear

d) *agrupamento*, ou *clusterização:* é um aprendizado que identifica um conjunto finito de categorias ou agrupamentos para descrever os dados, ou seja, é uma classificação não-supervisionada. O objetivo é particionar a base de dados em um número de *clusters* (grupos), em que as entidades de um grupo sejam similares. As categorias podem ser mutuamente exclusivas, ser categorias hierárquicas ou, ainda, possuir características em comum. Segmentação demográfica e redes neurais são técnicas usadas para agrupar dados. A Figura 2.5 mostra um agrupamento do conjunto de dados de empréstimos bancários em três grupos distintos, usando retângulos como delimitadores dos *clusters*. Pode-se notar que alguns dados pertencem a mais de um grupo, devido à interseção de grupos;

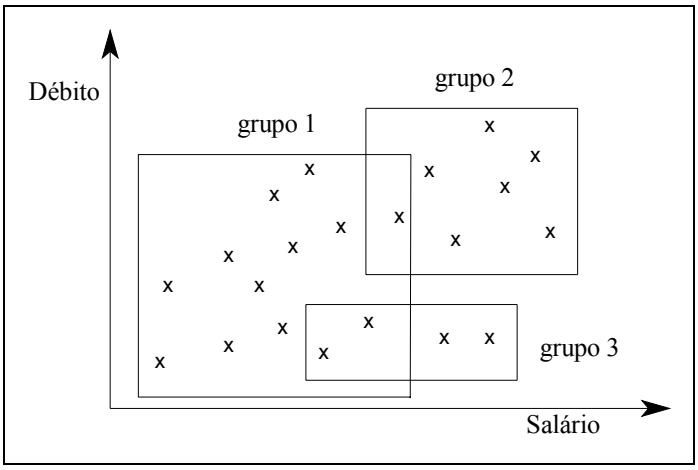

FIGURA 2.5 – Exemplo de agrupamento

e) *sumarização:* envolve métodos que encontram uma descrição compacta para um subconjunto de dados. Métodos mais sofisticados envolvem a derivação de regras de resumo e descobertas de relações funcionais entre variáveis. As técnicas de sumarização são sempre aplicadas à análise exploratória de dados e à geração automática de relatórios;

f) *dependência:* consiste em encontrar um modelo que descreva dependências significativas entre variáveis. O modelo de dependência pode ser de *nível estrutural* ou de *nível quantitativo*. No nível estrutural, o modelo está sempre representado de uma forma gráfica e com variáveis localmente dependentes em relação a outras, enquanto que, no nível quantitativo, o modelo especifica a força das dependências com alguma escala numérica. As redes de dependência probabilística usam independência condicional para especificar o aspecto estrutural do modelo e probabilidades, e usam correlação para especificar a força das dependências;

g) *desvio*, ou *detecção de desvio:* enfoca a detecção de mudanças significativas nos dados com relação a valores normativos medidos anteriormente. A partir de um conjunto de dependências, seqüências e/ou descrições de conceitos, o algoritmo procura os elementos contidos no banco de dados que estão fora dos padrões, que são exceções às regras ou anômalos.

2.4 Desafios da Descoberta de Conhecimento

A realização totalmente automatizada e de propósito geral da DCBD continua distante, pois existem vários problemas que ainda necessitam da orientação do usuário para guiar o processo. Dessa forma, esta área tem evoluído através de sistemas projetados e implementados com objetivos específicos, que podem ser utilizados sobre outras bases de dados, mas com a mesma intenção de conhecimento para o qual foi projetado. Por outro lado, os sistemas que tendem a ser mais genéricos dependem muito mais da intervenção do usuário, exigindo deste um conhecimento significativo sobre o processo de Descoberta de Conhecimento [FEL 97].

Existem algumas decisões que devem ser tomadas durante o desenvolvimento de um sistema de Descoberta de Conhecimento, tais como: a forma de representação do conhecimento extraído, a complexidade da pesquisa, o uso de prioridades de conhecimento, o controle da operação de descoberta e a seleção do método de mineração de dados mais apropriado. Também é problemática a escolha de esquemas adequados, de amostras e de projeções dos dados antes da enumeração. Estas decisões dependem da base de dados utilizada, do domínio e da aplicação do conhecimento descoberto.

Além destes problemas, existem alguns desafios referentes às bases de dados e ao próprio sistema a ser implementado, segundo Fayyad [FAY 96] e Feldens [FEL 97]:

a) *volume da base de dados.* As bases de dados com centenas de campos e tabelas ocupam muito espaço de armazenamento, o que pode resultar numa variedade enorme de padrões, combinações e hipóteses;

b) *alta dimensionalidade da base de dados.* A alta dimensionalidade é medida pelo grande número de campos de uma base de dados, o que aumenta de forma explosiva o tamanho do espaço de busca e também as chances do algoritmo de encontrar padrões falsos. Métodos para reduzir efetivamente a dimensionalidade e o uso de prioridades para identificar variáveis irrelevantes são utilizados como soluções possíveis;

c) *bases de dados redundantes.* A redundância, a estrutura hierárquica dos atributos e as relações entre os atributos encontradas nas bases de dados não podem ser consideradas conhecimento pelo algoritmo de extração;

d) *dados inconsistentes.* Além de atributos com valores nulos ou errados na base de dados, alguns atributos importantes para o processo de descoberta podem não estar presentes na base de dados. Uma solução seria utilizar estratégias estatísticas sofisticadas para identificar variáveis ocultas e utilizar amostras muito grandes dos dados, tornando a inconsistência menos significativa;

e) *dados irregulares.* Diferentes bases de dados são utilizadas em várias partes da organização, e conseqüentemente, os dados operacionais podem ter diferentes domínios para definir uma mesma informação e variar em termos de qualidade. Uma das soluções para este problema seria uma análise efetiva de qual a melhor base de dados para selecionar os dados, ou então utilizar um *Data Warehouse,* que integra em um único repositório, os dados operacionais necessários ao processo decisório e que estão espalhados em diferentes e heterogêneas bases de dados, e apresentando, assim, um ambiente estável e integrado dos dados;

f) *dados constantemente alterados.* A natureza dinâmica dos dados faz com que eles sejam constantemente alterados, o que pode levar a conclusões apenas momentâneas, pois as variáveis medidas podem ter sido removidas ou modificadas. Uma possível solução seria utilizar métodos para atualizar os padrões ou utilizar apenas os padrões que sofreram mudanças;

g) *interação com o usuário*. Os sistemas de Descoberta de Conhecimento devem ser autônomos e extrair apenas hipóteses úteis. Por outro lado, esses sistemas precisam ser configurados para a aplicação e base de dados de cada usuário, de acordo com as suas necessidades e o conhecimento que ele possui;

h) *privacidade dos dados.* Como os sistemas de Descoberta de Conhecimento possuem um teor investigatório, eles podem acabar revelando informações de caráter particular dos indivíduos, que não devem ser utilizadas de forma indevida ou não autorizada. Dessa forma, a privacidade deve ser mantida através da troca ou exclusão de campos de identificação, descobrindo, assim, os padrões sem invadir a privacidade dos indivíduos presentes nos dados [NOG 2000];

i) *conhecimento anterior.* Muitos métodos e ferramentas de Descoberta de Conhecimento não são verdadeiramente interativos e não podem incorporar o conhecimento anterior sobre um problema de modo simples. O uso do domínio de conhecimento e de probabilidades anteriores retiradas dos dados são importantes em todas as etapas do processo de Descoberta de Conhecimento;

j) *representação da informação*. Se a informação descoberta não for claramente compreensível e acessível ao usuário, ele poderá ter errôneas ou diferentes interpretações do conhecimento. Uma possível solução para isto é a inclusão de representações gráficas, de linguagem natural e de técnicas modernas de visualização de dados;

k) *integração com outros sistemas.* Um sistema de descoberta isolado pode não ser muito útil. As abordagens típicas de integração incluem comunicação com o SGBD, com planilhas eletrônicas e com ferramentas modernas de visualização.

Os problemas relacionados às bases de dados ocorrem porque as bases não foram geradas com o propósito de Descoberta de Conhecimento, e sim com propósitos distintos, o que torna o processo de verificação de descrições muito oneroso.

É importante salientar que uma aplicação de DCBD deve buscar atender, também, às expectativas do usuário, que pode desejar uma interface amigável, respostas apenas das descobertas mais interessantes, preferir uma determinada forma de visualizar o conhecimento e/ou um tempo de resposta, no mínimo, satisfatório. Caso esta ferramenta não esteja adequada ao domínio da aplicação, às necessidades do usuário ou tenha sérios problemas de velocidade e grau de sofisticação, poderá trazer grandes prejuízos e decepções, pois essas características são essenciais para qualquer sistema de Descoberta de Conhecimento.

Todos estes aspectos apresentados são importantes e devem ser considerados durante o processo de desenvolvimento e utilização de um sistema de Descoberta de Conhecimento. Segundo Adriaans [ADR 97], a maior parte (cerca de 80%) dos esforços da DCBD se referem às etapas de preparação dos dados, enquanto que os outros 20% se referem à etapa de mineração de dados em si e ao pós-processamento. A manipulação de dados utilizando rotinas comuns para limpeza ou codificação é muito mais importante do que o próprio reconhecimento de padrões, pois sem os dados corretos dificilmente algum conhecimento útil será extraído.

2.5 Aplicações da Descoberta de Conhecimento

Conforme Prado [PRA 2001], a pesquisa em DCBD tem se realizado em duas dimensões principais: (I) expansão da capacidade de processamento dos algoritmos para bases de dados cada vez maiores e (II) automatização ou semi-automatização do processo de descoberta de conhecimento, principalmente no que se refere às etapas de pré-processamento.

Dessa forma, as ferramentas de DCBD têm evoluído muito e podem ser utilizadas em uma grande variedade de áreas de aplicação, estendendo-se pelos meios empresariais, industriais e científicos. A seguir, serão apresentados alguns domínios de aplicação da Mineração de Dados, conforme Halmenschlager [HAL 2000] e Noguez [NOG 2000]:

a) *análise de riscos:* realizada, por exemplo, através do agrupamento das características que identificam o cliente como um bom ou mau pagador, em sistemas de prevenção de níveis de maus empréstimos em um banco. Dessa forma, o banco pode verificar a probabilidade do novo cliente ser um mau pagador antes da assinatura do contrato, evitando riscos desnecessários;

b) *marketing direto*: uma das áreas que mais está utilizando Mineração de Dados, através da identificação dos clientes que devem ser incluídos em uma mala direta para se obter um retorno mais alto. Isso ocorre através da análise da base de dados e da seleção apenas dos clientes que possuem uma chance de resposta à mala direta significativamente mais alta. Isto permite à companhia remeter seus prospectos seletivamente, maximizando assim a resposta e reduzindo seus gastos;

c) *segmentação de mercado:* identificando as características comuns dos clientes que compram um determinado produto da empresa, ou que estão interessados em um determinado tipo de produto, pode-se maximizar as vendas dos clientes existentes, fazendo com que os clientes que compram um só tipo de produto comprem também os outros tipos;

d) *análise de mercado:* identificando a associação entre produtos ou serviços que são normalmente adquiridos juntos numa mesma transação. Dessa forma, pode-se desenvolver melhores estratégias de estoque, de *layout* e de promoção destes produtos;

e) *manufatura:* realizando previsões das vendas, determinando níveis confiáveis de estoques e programas de distribuição de mercadorias;

f) *detecção de fraudes:* encontrando comportamentos habituais que indicam transações fraudulentas no meio comercial (seguros, bancos), na medicina (despesas e procedimentos de hospitais), entre outros;

g) *análise de tendências:* revelando a diferença entre o consumidor típico atual e o consumidor típico de antes, dado um determinado período. Também usa mineração de dados para procurar padrões nas preferências dos clientes existentes, utilizando esses padrões para seleção de clientes futuros, o que revela clientes em potencial para um novo produto.

Tudo isso demonstra a grande aplicabilidade e importância da Descoberta de Conhecimento, e conseqüentemente, das técnicas de Mineração de Dados.

3 Classificadores

A classificação é uma das possíveis tarefas de mineração de dados, em que os algoritmos buscam por padrões que classifiquem os dados, ou seja, por classificadores.

No contexto de classificadores, um *exemplo*, *instância* ou *registro*, é um conjunto ordenado de *variáveis,* descrevendo um objeto de interesse. Cada variável, também chamada de *atributo* ou *característica*, possui valores de um conjunto pré-definido de valores que são dependentes do problema, e que descrevem algum aspecto do exemplo. Este conjunto pré-definido de valores é conhecido como *domínio* da variável. Existem dois tipos de domínios [GAM 99]:

a) *domínio nominal:* também conhecido como *domínio discreto,* do qual pode-se enumerar todos os valores possíveis que uma variável pode ter. Exemplo: {vermelho, azul, amarelo};

b) *domínio contínuo:* do qual nem sempre é possível enumerar todos os valores possíveis. Exemplo: temperaturas do corpo.

Cada exemplo possui uma variável especial, também conhecida como *atributo categórico, rótulo* ou *classe,* que descreve o fenômeno de interesse, sobre o qual será realizada a predição e, conseqüentemente, a classificação.

Um *conjunto de dados* usado na mineração de dados é constituído por um *conjunto de exemplos*, que possui *n* exemplos e *m* atributos. O *conjunto de treinamento* é um conjunto de exemplos rotulados e o *conjunto de teste* é um conjunto de exemplos não rotulados. O conjunto de teste também pode ser um conjunto de exemplos rotulados, proveniente do conjunto de exemplos, usado para estimar a qualidade do classificador.

Ao algoritmo de indução é fornecido um conjunto de treinamento. A tarefa do algoritmo é gerar um bom classificador a partir deste conjunto de instâncias classificadas. O classificador pode, então, ser usado para classificar instâncias não rotuladas, fazendo uma predição correta do rótulo de cada uma dessas novas instâncias.

Assim, pode-se considerar a *classificação* como uma tarefa de previsão, em que um conjunto de atributos previsores (características) é usado para prever um atributo meta (rótulo). A tarefa consiste em descobrir o relacionamento entre os atributos previsores e o atributo meta, usando um conjunto de exemplos cuja classe é previamente conhecida. Após o aprendizado, o objetivo é utilizar o relacionamento descoberto para prever a classe de novos dados, a partir, apenas, de seus atributos previsores [GAM 99].

O classificador também pode ser avaliado em relação à sua acurácia (precisão), compreensibilidade, tempo de aprendizado, requisitos de armazenamento, compactabilidade (simplicidade) e alguma outra propriedade que defina o quão bom ele é para esta determinada tarefa [BAR 2001].

De maneira geral, o processo de classificação pode ser ilustrado pela Figura 3.1, em que o conhecimento do domínio, proveniente do especialista, pode ser utilizado ao escolher dados ou como entrada do indutor.

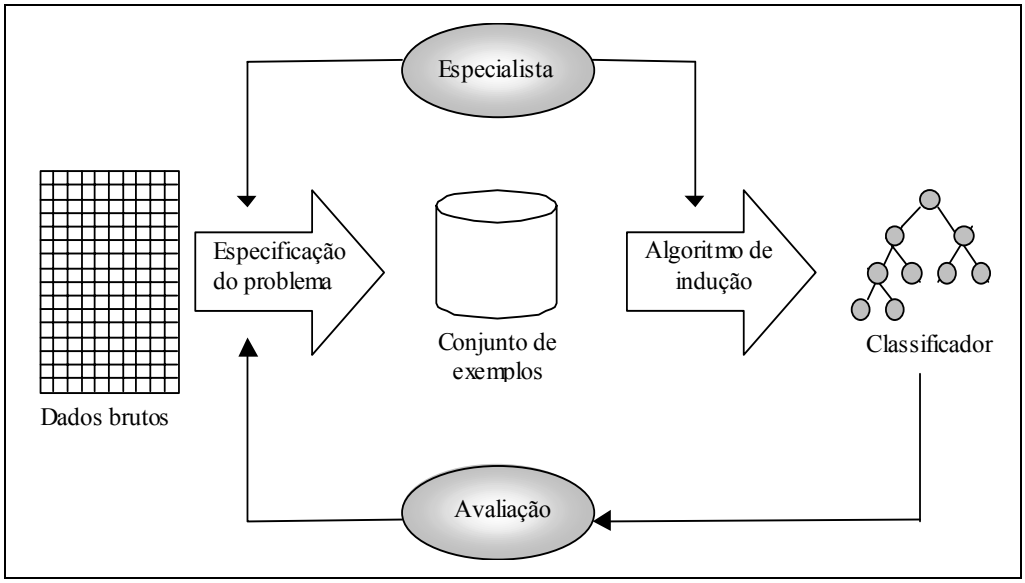

FIGURA 3.1 – Processo de classificação
Fonte: BARANAUSKAS, 2001. p.5

Para exemplificar o processo de classificação, suponha que se queira aprender uma forma de predizer se um paciente tem problemas cardíacos. Para isso, é necessário verificar os históricos de pacientes com informações de idade, sexo, dor no peito, nível de colesterol, nível de sedentarismo, taxa máxima de batimentos cardíacos, entre outros. Cada registro histórico é rotulado por um especialista médico como saudável ou doente. Após, o conjunto de exemplos composto pelos históricos é fornecido como entrada para um algoritmo de indução. A saída resultante consiste de algumas regras que permitem classificar novos pacientes, verificando se um novo paciente apresenta ou não problemas cardíacos.

Este exemplo mostra apenas uma aplicação da grande variedade de domínios em que os classificadores podem ser aplicados. Entretanto, segundo Clark [CLA 89], existem alguns requisitos que um classificador deve possuir:

a) *classificação precisa:* as regras induzidas deverão ser hábeis para classificar novos exemplos corretamente, permitindo um bom poder de predição;

b) *regras simples:* devido a compreensibilidade, o indutor de regras deverá induzir regras tão pequenas quanto possível, permitindo a entendibilidade dos resultados;

c) *eficiente geração de regras:* é importante que o indutor tenha capacidade de descobrir padrões também em situações complexas.

Há uma grande variedade de técnicas, ou métodos, que são utilizados na Mineração de Dados, com o objetivo de atingir a tarefa de classificação. Porém, ao selecionar um destes algoritmos, devem ser considerados vários aspectos decisivos para um bom desempenho da ferramenta de Descoberta de Conhecimento, pois algumas técnicas são mais adequadas para trabalhar com grandes volumes de dados ou com grande número de atributos, enquanto outras são boas no tratamento de atributos quantitativos ou do tipo qualitativos. Outro aspecto importante é a qualidade de saída desejada, ou seja, se a técnica é capaz de derivar regras e também de aprender incrementalmente, acrescentando novos conhecimentos aos adquiridos anteriormente. Também a possibilidade de avaliar os resultados gerados do ponto de vista estatístico e a performance geral devem ser consideradas no momento de seleção da técnica de Mineração de Dados.

Entre as técnicas de Mineração de Dados usadas para classificação, as *redes neurais* possuem capacidade de classificação indiscutível [FON 94]. Já as *árvores de decisão* e as *regras de classificação* são as mais populares e utilizadas.

As árvores de decisão apresentam, como principal vantagem, estruturas simples e de grande legibilidade, expressando seus resultados de uma forma muito clara, que podem ser facilmente entendíveis e usados diretamente pelo usuário [ADR 97]. Elas podem ser muito bem aplicadas a grandes conjuntos de dados e são adequadas para qualquer tipo de dados, capazes de manipular atributos contínuos e discretos. Apesar de ser uma das técnicas mais simples e eficiente, elas são propensas ao *overfitting*, que ocorre quando o modelo aprende detalhadamente os padrões ao invés de generalizar [ESP 97].

Já as regras de classificação resolvem a maior desvantagem das árvores de decisão: a dificuldade de compreensão quando aplicadas em grandes bases de dados, pois as regras podem ser compreendidas sem que haja a necessidade de se referenciar a outras regras. Porém, a principal desvantagem das regras de classificação é que elas encontram muitas associações que são apenas ruídos, além de não trabalharem muito bem com atributos numéricos [ADR 97]. Estas deficiências podem ser resolvidas, respectivamente, com a adição de um grau mínimo de suporte e de técnicas de discretização.

As representações através de regras de classificação e árvores de decisão são, geralmente, utilizadas em conjunto. Podem ser também equivalentes – se as regras forem derivadas da árvore, sendo a eficiência da classificação de árvores de decisão mantida quando transformada em regras de classificação.

Além dos aspectos citados acima, o conhecimento prévio das técnicas de árvores de decisão e regras de classificação motivaram o estudo das mesmas em sistemas de indução de classificadores.

3.1 Árvores de Decisão

A partir da idéia inicial de Hunt [QUI 93], no final da década de 50, as árvores de decisão foram usadas com sucesso em sistemas de Aprendizado de Máquina e têm sido estudadas tanto na área de reconhecimento de padrões quanto na área de aprendizado de máquina. A partir disso, outros pesquisadores trabalharam em métodos similares, como Breiman e Friedman apud [FON 94], que desenvolveram os fundamentos do sistema CART, utilizado por Quinlan como base para o sistema ID3 [QUI 88] e, posteriormente, no sistema C4.5 [QUI 93].

Uma *árvore de decisão* (AD)*,* ou *árvore de classificação,* usa uma representação baseada em árvores, que é um tipo de estrutura de dados não linear, que possui um número finito de *elementos* ou *nodos*. Cada árvore possui um único nodo especial, chamado *nodo raiz*, colocado no topo da representação gráfica, que é o nodo pai de suas subárvores, cujos nodos, por sua vez, são os nodos filhos do nodo raiz. Todos os nodos que possuem o mesmo pai são considerados nodos irmãos. Os nodos que não possuem filhos são considerados nodos terminais e são chamados de *folhas*. Já o *nível* de um nodo é a distância do mesmo até a raiz, ou seja, o número de ligações percorridas até chegar a raiz. A *profundidade* de uma AD é definida pela maior distância entre uma folha e a raiz, existindo árvores com profundidade uniforme em todas as folhas e outras não. Dependendo do número de filhos de cada nodo, a árvore pode ser considerada *binária*, quando possuir sempre dois filhos em cada nodo; *ternária*, quando possuir três ligações em cada nodo; ou *mista*, quando o número de filhos for variável [MUR 97].

Em uma AD, os *nodos* da árvore correspondem a nomes de atributos, as *ligações* de um nodo representam valores deste atributo e as *folhas* representam as diferentes classes a que pertencem as entidades. Um objeto é classificado seguindo o percurso da raiz até a folha, enquanto as suas características satisfizerem os nodos e as suas ligações.

A Figura 3.2 mostra um exemplo de uma árvore de decisão para a prescrição do uso, ou não, de lentes de contato. Na figura, cada elipse é um teste de uma característica e cada retângulo é uma classe.

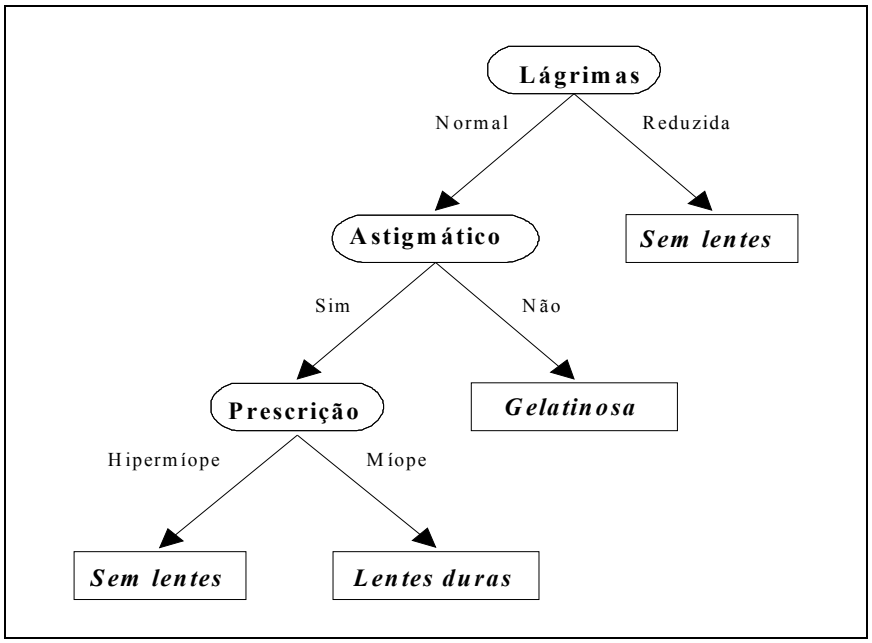

FIGURA 3.2 – Exemplo de uma árvore de decisão
Fonte: WITTEN, 2000. p.13

Formalmente, uma árvore de decisão é um gráfico acíclico, em que cada nodo é um nodo de decisão, que possui algum teste baseado nos valores do atributo [GAM 99]. Cada nodo possui um ou mais sucessores, ou um nodo folha, que é rotulado com uma classe. Cada caminho da árvore de decisão forma uma regra com a parte condicional contendo nodos e uma conclusão formada pela folha.

Dessa forma, após a geração da AD, a mesma pode ser utilizada para classificar novas instâncias, percorrendo a árvore de forma *top-down* pelos nodos de decisão até encontrar a folha, que rotulará essa nova instância, de acordo com a respectiva classe.

Além das inúmeras vantagens já citadas, Gama [GAM 99] apresenta algumas outras características das árvores de decisão, que são utilizadas tanto na pesquisa acadêmica quanto em aplicações reais:

a) *flexibilidade*: não assumem uma distribuição única dos dados, sendo métodos não-paramétricos. O espaço da instância é particionado em subespaços e cada subespaço é adaptado a diferentes modelos;

b) *robustez:* a seleção interna de características produz árvores que tendem a ser bastante robustas mesmo com a adição de variáveis irrelevantes;

c) *interpretabilidade*: todas as decisões são baseadas nos valores (conhecidos) dos atributos usados para descrever o problema;

d) *velocidade*: a maioria dos algoritmos constrói a AD de forma *top-down,* usando uma abordagem de "dividir para conquistar", sem *backtracking*[3].

[3] Capacidade de seguir um caminho e depois retornar ao início para percorrer outro.

Já entre os problemas mais comuns, pode-se citar [GAM 99]:

a) *fragmentação*: causa o particionamento de dados em pequenos subconjuntos, o que é indesejável na presença de muitos atributos relevantes;

b) *replicação*: é a duplicação de uma seqüência de testes em diferentes ligações de uma árvore de decisão, levando a uma representação inconcisa que tende a diminuir a precisão preditiva;

c) *valores desconhecidos*: podem causar problemas na decisão de que ligação seguir, pois uma árvore é uma hierarquia de testes;

d) *valores contínuos*: a presença de atributos contínuos representa o gargalo do algoritmo, pois é necessário muito tempo de processamento para realizar o tratamento destes atributos;

e) *instabilidade*: pequenas variações no conjunto de treinamento podem produzir grande variação na árvore final.

Como dito anteriormente, os algoritmos de indução de árvores de decisão se baseiam na estratégia de "dividir para conquistar" [SOU 98], que consiste na sucessiva divisão do problema em vários subproblemas de menores dimensões, até que uma solução para cada um dos problemas mais simples possa ser encontrada. Apesar disso, existem alguns aspectos diferentes e determinantes usados na construção dos mesmos, que são descritos na próxima seção.

3.1.1 Construção de árvores

A tarefa de construção de uma AD é chamada de *indução.* Já *discriminação,* ou *derivação,* é o processo de derivar regras a partir da AD, e *classificação* é a tarefa de aplicar estas regras para novos objetos ainda não classificados [MUR 97].

A abordagem mais comum para indução de árvores é particionar, recursivamente, conjuntos de exemplos classificados até que um critério de parada seja encontrado. A partição é definida relacionando um teste que tenha um pequeno conjunto de saídas, criando uma ligação para cada saída possível, passando cada exemplo para a ligação correspondente e tratando cada bloco da partição como um subproblema para o qual uma subárvore é construída recursivamente. Um critério comum de parada é quando todos os exemplos de uma ligação são da mesma classe.

O algoritmo padrão para a construção de árvores de decisão utiliza em cada nodo um teste baseado em somente um atributo, adicionando ligações com seus possíveis valores. As árvores assim resultantes são chamadas de *árvores univariadas*. Também pode-se utilizar testes baseados em combinação linear de características, isto é, as decisões são baseadas em valores de um conjunto de variáveis, gerando *árvores multivariadas*. Os testes univariados podem resultar em árvores grandes e difíceis de entender, enquanto testes multivariados podem produzir árvores menos complexas, mas com um processamento mais lento [MUR 97].

O processo de indução de árvores de decisão é realizado através da estratégia *Top Down Induction of Decision Trees* (TDIDT), ou simplesmente *top-down,* que inicia a geração da árvore pela raiz e continua por seus filhos. Esta, por sua vez, é dividida em duas outras abordagens [BAR 2000a]:

a) *por profundidade (depth-first):* dado um nodo inicial, é aplicado um operador (ligação) para chegar a um novo nodo. Se este não for uma folha, é adicionada nova ligação e um novo nodo, e assim, sucessivamente. Quando for incluída uma folha, é feito um recuo, retornando ao nodo pai da folha e adicionando um novo operador e um novo nodo. Se novamente não for possível adicionar um novo nodo, é retornado até o nodo que permita a inclusão de uma nova ligação. A pesquisa termina quando nenhum nodo puder ser encontrado. A Figura 3.3 mostra a ordem em que os nodos são incluídos;

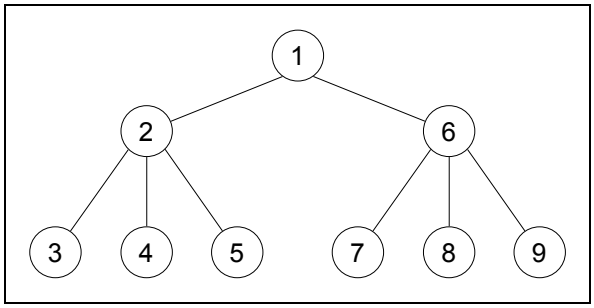

FIGURA 3.3 – Indução por profundidade

b) *por largura (breadth-first):* aplica todas as ligações de um nodo, chegando a novos nodos. Então, para cada nodo resultante, é testado o critério de término, adicionando a sua folha ou novos nodos. Não utiliza, portanto, *backtracking,* pois não retorna ao nodo pai. A Figura 3.4 mostra a ordem em que os nodos são percorridos nesta abordagem.

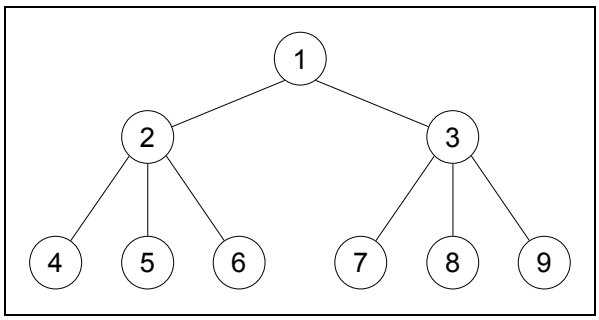

FIGURA 3.4 – Indução por largura

A estratégia por profundidade nem sempre encontra uma solução ótima ou completa, porém não necessita de muitos recursos de memória. Já a estratégia por largura é completa e ótima, porém os requisitos computacionais são grandes.

Embora existam diversas linhas de pesquisa na busca de algoritmos mais eficientes, em que o objetivo seja efetuar o aprendizado sobre grandes conjuntos de dados, o maior desafio é projetar algoritmos eficazes, que produzam um classificador com um bom poder de predição. Para isso, diversas heurísticas utilizadas na construção de árvore de decisão têm sido propostas, conforme apresentado a seguir.

3.1.1.1 Escolha do nodo

Existem várias técnicas para a eleição do melhor atributo a ser utilizado em um nodo de uma árvore de decisão, constituindo a função de avaliação de cada partição e a chave para o sucesso do algoritmo de indução. Assim, a função de avaliação verifica cada atributo candidato e seleciona aquele que maximiza (ou minimiza) alguma função heurística sobre os subconjuntos.

Entre as principais funções para a escolha do atributo a utilizar como nodo estão:

a) *escolha randômica:* citado por [BAR 2000a], este método não utiliza nenhuma heurística, sendo escolhido qualquer atributo entre os atributos disponíveis;

b) *critério da entropia:* desenvolvido por Quinlan [QUI 93], este critério mede a quantidade de informação necessária para codificar a classe do nodo. Assim, se for selecionado aleatoriamente um caso de um subconjunto S de casos e resultar que ele pertence a uma classe C_j, a probabilidade será:

$$P = \left(\frac{C_1}{S}, \ldots, \frac{C_n}{S} \right)$$

onde:
C_j: ocorrências da classe
S: número de exemplos

Para um determinado atributo A_i, com valores $\{A_1, \ldots, A_n\}$, a distribuição de probabilidade destes diferentes valores de A é representada por $P\ \{p_1, \ldots, p_n\}$. A informação contida nesta distribuição é chamada *Entropia (P)*:

$$Entropia(P) = -(p_1 * \log_2(p_1) + p_2 * \log_2(p_2) + \ldots + p_n * \log_2(p_n))$$

Se o conjunto de registros *S* for dividido em diferentes classes *{C1, ..., Cn}* conforme o valor de um determinado atributo categórico *C*, a quantidade de informação necessária para identificar a classe de um indivíduo em *S* será dada por:

$$Info(S) = Entropia(P) = -\sum_{j=1}^{k} p_j * \log_2(p_j)$$

onde:

p_j:	probabilidade relativa da classe C_j em *S*
k:	número de classes

Para encontrar o valor esperado (média) da informação de um atributo, pertencente a uma classe qualquer, deve-se somar as informações sobre todas as classes, ponderadas pelas suas respectivas freqüências em *S* para cada uma das partições:

$$Info(S, A) = \sum_{i=1}^{m} \left(\frac{S_i}{S} \right) * Info(S_i)$$

onde:

S_i:	número de exemplos para a partição
m:	número de partições

O ganho do atributo *A*, que mede a informação ganha pela partição de *S* de acordo com o teste *A*, é, então, definido por:

$$ganho = Info(S) - Info(S, A)$$

Para uma melhor compreensão, suponha-se que haja dois atributos candidatos ao próximo nodo da AD: A_1 e A_2, que possui dez registros nesta partição. A Figura 3.5 mostra a distribuição das classes *S* e *N* nas folhas de cada um destes atributos:

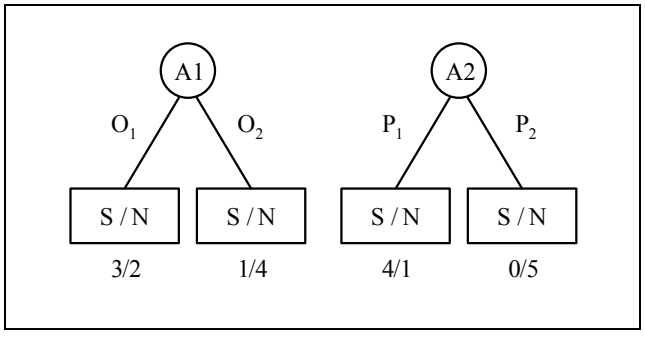

FIGURA 3.5 – Exemplo de duas possíveis partições

Em cada folha da Figura 3.5, têm-se dois valores: um deles se refere aos registros corretamente classificados e o outro valor aos incorretamente classificados. Por exemplo: o atributo A_1, possui três registros corretos e dois incorretos para a classe *S*, e um registro incorreto e quatro corretamente classificados para a classe *N*.

A quantidade de informação para os atributos A_1 e A_2, pela figura acima, seria:

$$Info(S) = \left(-\frac{4}{10} * \log_2\left(\frac{4}{10}\right) - \frac{6}{10} * \log_2\left(\frac{6}{10}\right)\right) = 0{,}971$$

$$Infd(S, A_1) = \frac{5}{10} * \left(-\frac{3}{5} * \log_2\left(\frac{3}{5}\right) - \frac{2}{5} * \log_2\left(\frac{2}{5}\right)\right) + \frac{5}{10} * \left(-\frac{1}{5} * \log_2\left(\frac{1}{5}\right) - \frac{4}{5} * \log_2\left(\frac{4}{5}\right)\right) = 0{,}846$$

$$ganho(A_1) = 0{,}971 - 0{,}846 = 0{,}125$$

$$Infd(S, A_2) = \frac{5}{10} * \left(-\frac{4}{5} * \log_2\left(\frac{4}{5}\right) - \frac{1}{5} * \log_2\left(\frac{1}{5}\right)\right) + \frac{5}{10} * \left(-\frac{0}{5} * \log_2\left(\frac{0}{5}\right) - \frac{5}{5} * \log_2\left(\frac{5}{5}\right)\right) = 0{,}361$$

$$ganho(A_2) = 0{,}971 - 0{,}361 = 0{,}61$$

Logo, o atributo A_2 seria considerado o melhor atributo pelo critério de entropia, por possuir o maior ganho de informação.

c) *técnica de Laplace:* citado por [FON 94], esta técnica escolhe e usa, para realizar a partição do conjunto de treinamento em cada nodo, a característica que minimizar o valor do erro esperado:

$$E = \sum_{i=1}^{m} \frac{n_i}{n} \sum_{j=1}^{K} p_{i,j}\left(1 - p_{i,j}\right) \qquad \text{com} \qquad p_{i,j} = \frac{n_{i,j} + 1}{n_i + k}$$

onde:

k:	número total de classes
m:	número de subconjuntos da partição
n:	número de exemplos da partição
$n_{i,j}$:	número de exemplos da classe C_j que possuem o *i-ésimo* valor
n_i:	número de exemplos que possuem o *i-ésimo* valor
$p_{i,j}$:	probabilidade de um exemplo da classe C_j ter o *i-ésimo* valor

Neste caso, o *i-ésimo valor* se refere a cada um dos *m* valores do atributo em questão. Também utilizando o exemplo da Figura 3.5, o erro esperado para cada atributo seria:

$$E(A_1) = \frac{5}{10}\left(\frac{4}{7} * \left(1 - \frac{4}{7}\right) + \frac{3}{7} * \left(1 - \frac{3}{7}\right)\right) + \frac{5}{10}\left(\frac{2}{7} * \left(1 - \frac{2}{7}\right) + \frac{5}{7} * \left(1 - \frac{5}{7}\right)\right) = 0{,}45$$

$$E(A_2) = \frac{5}{10}\left(\frac{5}{7} * \left(1 - \frac{5}{7}\right) + \frac{2}{7} * \left(1 - \frac{2}{7}\right)\right) + \frac{5}{10}\left(\frac{1}{7} * \left(1 - \frac{1}{7}\right) + \frac{6}{7} * \left(1 - \frac{6}{7}\right)\right) = 0{,}33$$

O atributo A_2 seria escolhido por possuir o menor erro esperado: 0,33.

d) *critério de Gini:* este método, desenvolvido por Breiman apud [ALS 98], visa minimizar a impuridade de cada nodo. A impuridade de um nodo é máxima quando todas as classes possuem igual distribuição, e mínima quando existe apenas uma classe. Considerando um conjunto de dados *S,* que contém *n* registros, cada um com uma classe C_i, o índice Gini de *S* é:

$$gini(S) = 1 - \sum_{i=1}^{k} p(C_i \mid n)^2$$

onde:

p_i:	probabilidade relativa da classe C_i em *S*
n:	número de registros em *S*
k:	número de classes

Se *S* for particionado em dois subconjuntos: S_1 e S_2, um para cada ligação, o índice Gini dos dados particionados será definido por:

$$gini(S, A) = \frac{n_1}{n} gini(S_1) + \frac{n_2}{n} gini(S_2)$$

onde:

n_1:	número de exemplos de S_1
n_2:	número de exemplos de S_2

Também considerando os atributos da Figura 3.5, o índice de Gini para A_1 seria:

$$gini(S_1) = 1 - \left(\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2 \right) = 0{,}48$$

$$gini(S_2) = 1 - \left(\left(\frac{1}{5}\right)^2 + \left(\frac{4}{5}\right)^2 \right) = 0{,}32$$

$$gini(S, A_1) = \frac{5}{10} * 0{,}48 + \frac{5}{10} * 0{,}32 = 0{,}40$$

Da mesma forma, para A_2, teria-se:

$$gini(S_1) = 1 - \left(\left(\frac{4}{5}\right)^2 + \left(\frac{1}{5}\right)^2 \right) = 0{,}32$$

$$gini(S_2) = 1 - \left(\left(\frac{0}{5}\right)^2 + \left(\frac{5}{5}\right)^2 \right) = 0$$

$$gini(S, A_2) = \frac{5}{10} * 0{,}32 + \frac{5}{10} * 0 = 0{,}16$$

Assim, seria escolhido como nodo o atributo que permitisse a menor impuridade: o atributo A_2, com índice Gini igual a 0,16.

e) *método da paridade:* aplicado em árvores binárias para definir a medida de impuridade, que é citado por Fonseca [FON 94]:

$$\Delta i(S) = \frac{p_E p_D}{4}\left[\sum_{i=1}^{k} \left| p(C_i \mid n_E) - p(C_i \mid n_D) \right|\right]^2$$

onde:

p_E:	probabilidade do nodo descendente esquerdo
p_D:	probabilidade do nodo descendente direito
$p(C_i \mid n_E)$:	probabilidade da classe C_i do nodo descendente esquerdo
$p(C_i \mid n_D)$:	probabilidade da classe C_i do nodo descendente direito

Considerando o exemplo da Figura 3.5, teria-se, respectivamente, para A_1 e para A_2:

$$\Delta i(S, A_1) = \frac{\frac{5}{10} * \frac{5}{10}}{4}\left[\left|\frac{3}{5} - \frac{1}{5}\right| + \left|\frac{2}{5} - \frac{4}{5}\right|\right]^2 = 0{,}04$$

$$\Delta i(S, A_2) = \frac{\frac{5}{10} * \frac{5}{10}}{4}\left[\left|\frac{4}{5} - \frac{0}{5}\right| + \left|\frac{1}{5} - \frac{5}{5}\right|\right]^2 = 0{,}16$$

Neste método, também seria escolhido como melhor atributo aquele que minimizasse a impuridade. Neste caso, o atributo A_1 seria escolhido, com impuridade igual a 0,04.

Assim, os critérios apresentados selecionam o melhor atributo para um determinado subconjunto de treinamento, em que a sua medida indica quão bem este atributo discrimina a classe.

3.1.1.2 Definição da partição

Ao estabelecer o atributo a ser utilizado como nodo, é necessário definir juntamente o particionamento das ligações do mesmo, pois quase todos os critérios de escolha do nodo (seção anterior) utilizam a distribuição da partição em seus cálculos. É importante ressaltar que o tipo de partição efetuada em cada nodo afeta de forma decisiva o desempenho da árvore induzida. O tipo dos atributos também é um importante fator, pois o tratamento dado aos atributos discretos é distinto do dado aos atributos contínuos.

Dessa forma, para definir o particionamento das ligações de cada nodo para atributos discretos, pode-se utilizar os seguintes critérios:

a) *criação de uma ligação para cada valor do atributo:* é atribuída uma ligação para cada valor do atributo. Embora permita extrair do atributo todo o seu conteúdo informativo, a principal desvantagem deste método é a criação de um número muito grande de ramificações, o que ocasiona a geração de árvores muito complexas e com ligações desnecessárias. Entretanto, este é o método mais simples;

b) *criação de nodos binários:* esta solução foi criada por Hunt apud [FON 94], em que é atribuído a uma das ligações um dos valores do atributo eleito (pela entropia, Gini, etc.) e à outra ligação todos os outros valores. Esta solução é bastante simples e inteligível, porém não aproveita todo o poder de discriminação do atributo;

c) *ordenação dos valores:* também cria duas ligações, sendo atribuídos a uma delas os valores de $x_n \leq A$, em que x_n é um atributo e A é um valor, e à outra ligação os valores $x_n > A$. Este método, desenvolvido por Breiman apud [FON 94], somente pode ser utilizado para atributos que possuam uma relação de ordem entre os seus possíveis valores. Para um atributo com cardinalidade n, serão possíveis $n-1$ diferentes partições (ou valores de A), tornando-se necessário testar todas para escolher a melhor delas. Proporciona uma árvore bastante compreensível para o usuário, apesar de não utilizar a capacidade total de cada característica;

d) *agrupamento de valores em dois conjuntos:* apresentado por Breiman apud [FON 94], este método também propõe a criação de duas ligações, associando um subconjunto de valores do atributo a uma delas e um outro subconjunto à outra. Como todos os subconjuntos possíveis deverão ser testados, totalizando $(2^{n-1}-1)$ partições, de modo a garantir a seleção da melhor partição, possui a grande desvantagem de necessitar de um número muito grande de testes, principalmente quando a cardinalidade do atributo é elevada;

e) *agrupamento de valores em vários conjuntos:* este método proposto por Quinlan apud [FON 94], permite agrupar valores do atributo em várias partições, não somente em duas. Primeiramente, é calculado o valor da solução, atribuindo a cada diferente valor da característica uma ligação própria. Em seguida, são testadas todas as combinações possíveis tendo uma das partições dois valores. Se nenhuma dessas soluções for considerada boa pelo critério de avaliação, o processo pára, sendo a solução anterior adotada como solução. Caso contrário, é repetido o processo de testar as diversas combinações, tendo como base a melhor das soluções anteriores. Possui uma complexidade de cálculo razoável, mas a solução encontrada não é, necessariamente, a melhor possível.

Já o particionamento de atributos contínuos implica uma maior complexidade de cálculo, podendo-se utilizar os seguintes critérios:

a) *testes simples:* citado por Fonseca [FON 94], este critério escolhe para a partição do nodo o ponto de cisão considerado de melhor valor pelo critério de avaliação adotado, testando e avaliando todas as partições possíveis com base em cada uma das características. O ponto de cisão consiste em um teste binário com resultados $x \leq A$ e $x > A$, em que x é o atributo e A é um valor de limiar. Para encontrar este ponto de cisão, ou seja, o valor de A, primeiramente ordenam-se todos os valores do atributo de forma crescente: $\{v_1, v_2, ..., v_n\}$. A seguir, o ponto médio de cada dois valores consecutivos, dado por $A=(v_i+v_{i+1})/2$, é um dos possíveis valores do ponto de cisão. Este valor será, então, avaliado pela função utilizada na escolha do nodo. O ponto de cisão que obtiver o melhor resultado será utilizado para efetuar a partição do atributo contínuo. Deve-se examinar todos os *m-1* limiares possíveis. Por exemplo, o conjunto de valores de umidade: {70, 85, 90, 95}, corresponde a três possíveis valores do ponto de cisão: A_1*: umidade≤77;* A_2*: umidade≤87;* A_3*: umidade≤92*. Desta forma, para cada teste deve ser calculado o resultado da função de avaliação, como, por exemplo, o ganho de informação, e escolhido aquele que maximizar esta função;

b) *testes múltiplos:* definem-se múltiplos intervalos de partição do valor de um atributo, conforme citado por Fonseca [FON 94]. Para definir estes múltiplos intervalos, uma das soluções possíveis é utilizar a segmentação global ou a segmentação em nível de nodo. A segmentação em nível global transforma o problema em atributos apenas discretos, e a segmentação em nível de nodo segmenta os valores contínuos em cada nodo. Isso pode aumentar significativamente a capacidade discriminativa de cada medida, permitindo árvores de dimensões menores sem perda de desempenho. No entanto, os cálculos podem penalizar, e muito, este método;

c) *combinação linear de características:* segundo Baranauskas [BAR2000a], este método permite a criação de árvores multivariadas, combinando linearmente características em cada nodo da seguinte forma: $c_1x_1+c_2x_2+...+c_mx_m$ *op* A, em que c_i é uma constante, x_i é uma característica, *op* é um operador do conjunto $\{<, \leq, >, \geq\}$ e A é um valor constante. A combinação linear de características permite contornar a

limitação apresentada pelas árvores baseadas em testes efetuados sobre uma só característica. Neste tipo de teste, o espaço de busca não é particionado em regiões retangulares e sim em hiperplanos, existindo alguns algoritmos que implementam este tipo de teste – como o OC1, descrito com maiores detalhes em Murthy [MUR 97].

Por outro lado, não pode-se usar valores desconhecidos ao efetuar a partição, pois ao utilizar o classificador para um novo exemplo, percorre-se a árvore da raiz até a folha, executando um teste dos valores dos atributos em cada nodo. Como uma AD consiste de testes hierárquicos, se o valor deste teste não for conhecido, a função pode não determinar o caminho a seguir. Assim, é necessário o tratamento prévio de valores desconhecidos, que, conforme Fonseca [FON 94], pode ser feito por:

a) *exclusão do valor:* utiliza em cada nodo apenas atributos que possuem um valor definido, excluindo os exemplos com valores desconhecidos;

b) *rotulação do valor:* utiliza um novo valor para todos os valores indefinidos. Assim, todos os exemplos para os quais o valor do atributo testado é desconhecido são atribuídos a uma mesma ligação;

c) *substituição:* substitui o valor desconhecido pelo valor mais comum do atributo encontrado no conjunto de treinamento;

d) *partições alternativas:* apresentado por Breiman apud [FON 94], este método permite obter variáveis de teste alternativas em cada nodo, de modo a ser possível a classificação de exemplos incompletos. Uma partição alternativa consiste em, baseada em outro atributo, imitar a partição normal, tendo por objetivo, na falta do valor do atributo testado normalmente, efetuar uma divisão dos exemplos tão idêntica quanto possível. A semelhança entre as partições será dada pela porcentagem de exemplos de cada classe e pelos próprios exemplos.

3.1.1.3 Definição da folha e da classe

Com as ligações já adicionadas, o próximo passo é verificar se o próximo nodo será terminal, ou seja, uma folha. A decisão de quando o nodo é uma folha, segundo Esposito [ESP 97], pode ser feita quando:

a) todos os exemplos atingem um nodo pertencendo à mesma classe;

b) todos os exemplos atingem um nodo, possuindo o mesmo vetor de características, mas não necessariamente a mesma classe;

c) o número de exemplos em um nodo é menor que certo percentual;

d) o resultado do melhor atributo para todos possíveis testes que particionam o conjunto de observações é muito baixo.

Finalmente, a definição de qual classe atribuir a uma determinada folha pode ser feita pela atribuição da classe de maior probabilidade ou pela determinação baseada na noção de custos [FON 94]:

a) *atribuição da classe mais provável:* é atribuída a classe mais freqüente dentro dos exemplos que se encontram nesta folha:

$$\max(p_j) = \max_{j=1}^{k} \frac{n_j}{n}$$

onde:

n:	número total de exemplos na folha
n_j*:*	número de exemplos da classe C_j na folha
k:	número de classes

Considerando novamente o exemplo da Figura 3.5, para definir a classe da partição esquerda do atributo A_1, tem-se:

$$(p_1) = \frac{3}{5} = 0{,}6$$

$$(p_2) = \frac{2}{5} = 0{,}4$$

Logo, a probabilidade que maximiza esta função é 0,6 (60%), atribuída à classe *S*.

b) *determinação baseada na noção de custos:* é atribuída uma classe que, baseada na noção de custos, minimiza os custos provenientes desta classificação:

$$Custo(j) = \sum_{i=1}^{K} p_i C_{i,j}$$

onde:

k:	número de classes
$C_{i,j}$*:*	valor da linha *i*, coluna *j* da matriz de custo
p_i*:*	probabilidade da classe C_i

A matriz de custos contém o custo para cada classe. Assim, para o atributo A_1 do exemplo da Figura 3.5, tem-se a seguinte matriz de custos:

TABELA 3.1 – Matriz de custos

Classe	S	N
S	0	1
N	2	0

Com base nesta matriz de custos, o custo proveniente da adoção de cada uma das classes é:

$$Custo(S) = \frac{3}{5} * 0 + \frac{2}{5} * 2 = 0{,}8$$

$$Custo(N) = \frac{3}{5} * 1 + \frac{2}{5} * 0 = 0{,}6$$

Tendo como objetivo a minimização do custo, seria escolhida a classe *N* para a folha esquerda do atributo A_1, mesmo possuindo uma menor quantidade de exemplos.

3.1.1.4 Limites

As árvores de decisão estão propensas ao *overfitting*, que ocorre quando o modelo aprende detalhadamente os padrões ao invés de generalizar [ESP 97], ou seja, o classificador gerado é muito específico para o conjunto de treinamento utilizado. Isso acontece uma vez que a construção de uma AD implica o crescimento da mesma, enquanto novas partições, que melhorem a habilidade da árvore em separar os casos do conjunto de treinamento em classes, são encontradas. Esta tática ajusta a árvore em relação ao conjunto de treinamento, mas pode gerar uma estrutura ajustada em excesso nesses casos, falhando na generalização do conhecimento em um conjunto de teste, com casos não vistos durante o treinamento.

Para contornar o problema do crescimento exagerado das árvores de decisão, pode-se substituir nodos profundos por folhas, o que é conhecido como *poda* da AD, retirando ligações que fornecem pouco poder de previsão por folha. Isto é realizado analisando-se a taxa de erro do nodo e a taxa de erro que ocorre quando se poda o mesmo. A taxa de erro de uma folha representa a razão entre o número de casos com classificação errada (*ce)* e o número de casos classificados corretamente (*cc*) pela partição:

$$E(T) = \frac{ce}{ce + cc}$$

A poda na árvore de decisão causa erro de classificação em alguns exemplos do conjunto de treinamento. Porém, a sua vantagem aparece quando é feita a classificação de novos exemplos que não foram usados no processo de construção da árvore, conduzindo a um erro de generalização menor.

Assim, a idéia é remover partes da árvore que não contribuem para a precisão da classificação, produzindo árvores menos complexas, e, conseqüentemente, mais compreensíveis.

Os métodos de poda podem ser divididos em duas abordagens principais: *pré-poda*, quando critérios de parada no processo de indução são satisfeitos, ou seja, alguma condição é satisfeita, parando a geração da árvore; e *pós-poda,* que constrói toda a árvore para depois podá-la, reduzindo a mesma para dimensões ótimas. Ambos os métodos fazem uma troca entre o tamanho da árvore e a taxa de erro estimada, podendo-se utilizar diferentes métodos para avaliar a AD e estimar a taxa de erro, como os descritos a seguir, na seção 3.3 [GAM 99].

Os métodos de pré-poda consistem na interrupção e/ou limitação do desenvolvimento da árvore durante a sua construção, prevenindo a construção da árvore naquelas partições que não fornecem precisão preditiva à árvore. São ignorados deliberadamente alguns exemplos do treinamento e incluída uma folha ao invés de um nodo (como visto na seção 3.1.1.3).

Alguns critérios de parada, segundo Fonseca [FON 94]:

a) *parada baseada na informação mútua:* se o ganho de informação obtido com o melhor atributo for inferior a um determinado limiar δ, o nodo é considerado folha e o desenvolvimento encerra:

$$ganho(S; A) < \delta$$

onde:
δ: parâmetro a ser ajustado

b) *parada baseada no teste de independência:* também conhecido como teste do *qui-quadrado* (χ^2), este método foi desenvolvido por Quinlan apud [FON 94]. Baseia-se na presença de um conjunto de treinamento S e de uma característica A que possibilita a partição $\{S_1, S_2, ..., S_m\}$. Sendo p e n as freqüências das classes P e N no conjunto de treinamento e p_i e n_i as freqüências da partição S_i. Assim, as freqüências de p'_i e n'_i são dadas por:

$$p'_i = p.\frac{p_i + n_i}{p + n} \qquad n'_i = n.\frac{p_i + n_i}{p + n}$$

O método, baseado no teste χ^2, pode ser usado para a determinação da confiança com que se rejeita a hipótese da característica ser independente da classe de objetivos em S:

$$\chi^2 = \sum_{i=1}^{m} \frac{(p_i - p'_i)^2}{p'_i} + \frac{(n_i - n'_i)^2}{n'_i}$$

Já as técnicas de pós-poda consistem em calcular o erro de uma árvore e de todas as suas subárvores, examinando cada um dos nodos não folhas da árvore, começando pelos nodos mais próximos das folhas e subindo de forma *bottom-up*. Se a substituição do nodo por uma folha ou pela sua ligação mais freqüente conduzir a um erro menor, é realizada a substituição, ou seja, sempre que o erro de qualquer subárvore decrescer, o erro global da árvore decrescerá também.

Portanto, depois de construída a árvore, é possível podá-la. Este processo, mostrado na Figura 3.6, reduz o número de nodos internos e, conseqüentemente, a complexidade da árvore, enquanto ganha uma melhor performance em relação à árvore original [BAR 2000a].

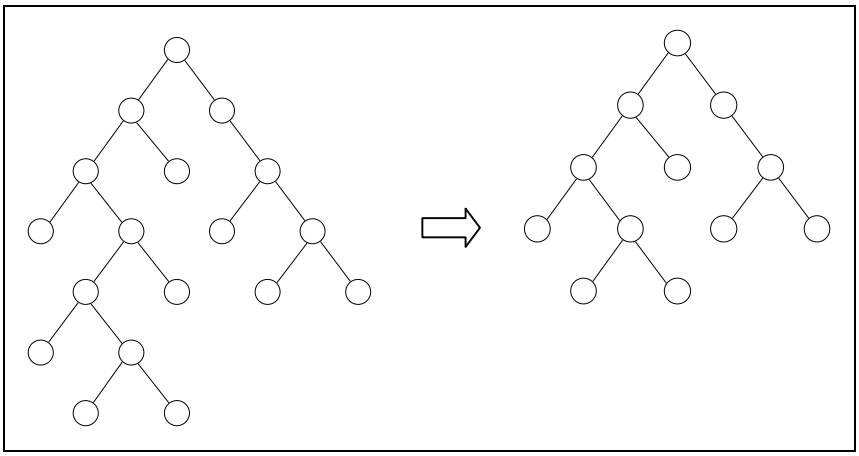

FIGURA 3.6 – Exemplo de poda

As técnicas de poda eliminam algumas partições da árvore de decisão de acordo com um dos seguintes critérios, citados por Esposito [ESP 97] e Fonseca [FON 94]:

a) *poda pelo algoritmo Laplaciano:* desenvolvido por Christie apud [FON 94], é um método bastante simples e eficiente. Inicialmente, é gerada a árvore completa, guardando o erro de cada nodo:

$$E = \sum_{i=1}^{m} \frac{n_i}{n} \sum_{j=1}^{K} p_{i,j}\left(1 - p_{i,j}\right) \qquad \text{com} \qquad p_{i,j} = \frac{n_{i,j} + 1}{n_i + k}$$

Todos os nodos internos (não folhas) da árvore desenvolvida são visitados, calculando o erro *E'* para os descendentes diretos de cada nodo:

$$E'_n = \frac{1}{n + d} \sum_{i=1}^{d} (n_i + 1) E_i$$

onde:

n_i:	número de exemplos abrangidos pelo *i-ésimo* descendente
E_i:	erro do nodo *i-ésimo* descendente
d:	número de descendentes do nodo *n*

Se $E'_n > E_n$, deve-se substituir o nodo por uma folha, pois os descendentes do nodo apresentam um erro maior do que o erro global do nodo pai. Caso contrário, deve-se substituir o erro E_n por E'_n;

b) *minimização do custo-complexidade (cost complexy pruning):* este método, desenvolvido por Breiman apud [FON 94], baseia-se nos seguintes passos:

1) criar uma árvore inicial tão grande quanto possível, com um valor *E(T)* de erro estimado suficientemente baixo;

2) podar sucessivamente a árvore criando um conjunto de árvores de menores dimensões, mas evitando a geração de um número demasiado elevado de subárvores através da seleção da melhor árvore de cada dimensão, utilizando, para isso, a taxa de erro por ressubstituição. A idéia é reduzir a árvore e, simultaneamente, o critério de complexidade e o critério de custo, obtendo uma combinação linear entre o custo e a sua complexidade. A medida de custo-complexidade é dada por:

$$Custo(T) = E(T) + \alpha|T|$$

onde:
E(T): erro aparente da árvore
|T|: número de folhas da árvore
α*:* fator de complexidade

O parâmetro α pesa a importância relativa do tamanho da árvore para a taxa de erro. Este parâmetro inicia em 0, variando até que esteja na presença de uma AD com apenas um nodo. Para cada valor de α deve ser encontrada a árvore $T_\alpha \leq T$, que minimiza o *Custo(T);*

3) utilizar alguma estimativa de erro, como a estimativa por utilização de um conjunto de exemplos independentes ou a validação cruzada, de modo a selecionar a melhor árvore criada no passo anterior. Também é necessário adotar um critério de comparação entre árvores de iguais dimensões (considerando o número de folhas).

c) *poda segundo Guo e Gelfand:* permite o cálculo de uma árvore podada ótima, com o menor erro estimado possível. Exige um conjunto de teste independente, reduzindo o conjunto de treinamento inicial. Examina cada nodo interno e substitui pela melhor folha possível, caso o número de erros dos não classificados permaneça igual ou seja menor. Os nodos são examinados sucessivamente usando uma abordagem *bottom-up*, ou seja, da folha em direção à raiz [FON 94];

d) *poda reduzindo o erro (reduced error pruning):* proposto por Quinlan apud [ESP 97], este método é considerado o mais simples, utilizando um conjunto de poda para avaliar a eficácia da subárvore. O processo inicia com a árvore completa e, utilizando uma abordagem *bottom-up,* examina cada nodo interno*,* comparando o número de erros de classificação feitos no conjunto de poda quando a subárvore é mantida, com o número de erros de classificação feitos quando é trocada por uma folha associada com a melhor classe. Se a árvore simplificada tiver uma precisão melhor que a original, será feita a poda. Esta operação de poda da partição é repetida na árvore simplificada até promover acréscimos da taxa dos não classificados. Uma propriedade positiva deste método é a sua complexidade computacional linear, uma vez que cada nodo é visitado somente uma vez para avaliar a oportunidade de podá-lo;

e) *poda baseada no erro (error based pruning):* desenvolvido por Quinlan apud [ESP 97] para resolver o problema da utilização de um conjunto independente de exemplos, utilizando o próprio conjunto de treinamento para efetuar a poda da árvore. Em um nodo, que classifica *n* casos do conjunto de treinamento, *k* deles incorretamente, o erro aparente do nodo será *k/n*. A árvore é percorrida de forma *bottom-up,* analisando o erro de cada nodo no caso de se substituir o mesmo por uma folha. Sempre que o valor do erro do nodo for inferior à soma dos erros de seus descendentes, o nodo é podado, sendo substituído por uma folha com a classe mais provável;

f) *poda do erro pessimista (pessimistic error pruning):* proposto por Quinlan apud [ESP 97], este método também é caracterizado por usar o mesmo conjunto de treinamento na construção e na poda da árvore. Porém, como a taxa de erro aparente é otimista, ela não pode ser utilizada para escolher a melhor árvore podada. Uma folha é substituída quando a taxa de erro da subárvore não é significativamente mais alta que o erro da folha. O método efetuará a poda se $|T|+raiz|T| >= 2E(T)$. Ou seja, a poda ocorre se *T* tem um número de folhas suficientemente alto em relação ao número de erros a cobrir e, se uma ligação *T* é podada, os descendentes de *T* não são examinados. O principal diferencial é o percurso da avaliação: o algoritmo avalia cada nodo iniciando na raiz, o que dá à técnica de poda alta velocidade de execução, pois caso um nodo seja podado, seus descendentes não serão examinados.

Várias comparações dos métodos de pós-poda já foram realizadas. Segundo Gama [GAM 99], os métodos baseados no custo-complexidade e na redução do erro são bons, enquanto os outros podem causar eventuais problemas.

A pós-poda é a abordagem mais utilizada e mais confiável, mas requer um processo mais lento, enquanto que a pré-poda tem a vantagem de não gastar tempo na construção de uma estrutura que não será utilizada no final da árvore.

Nem sempre a árvore podada é mais precisa que a correspondente gerada, mas a poda ajuda a simplificar a árvore, o que é essencial em árvores muito complexas.

3.1.2 Algoritmo CART

O algoritmo CART – *Classification And Regression Trees* foi apresentado por Leo Breiman, Jerome Friedman, Richard Oslen e Charles Stone apud [FON 94], em 1984. É um algoritmo não paramétrico. Possui grande capacidade de pesquisa de relações entre os dados, prevendo o tratamento de variáveis dependentes discretas, através da *classificação*, ou de variáveis contínuas, pela *regressão*.

O CART realiza a indução pela abordagem *top-down* de forma automática, requerendo uma mínima intervenção humana. O algoritmo se baseia em um arquivo de treinamento com dados previamente rotulados e constrói uma árvore de decisão, particionando em duas ligações cada nodo, em função de apenas um atributo, e separando os registros de cada partição.

O atributo a ser particionado é escolhido como aquele que gera grupos com a menor *diversidade*, ou seja, onde uma única classe predomina. O processo é aplicado recursivamente a cada um dos subconjuntos assim gerados, até que não seja possível ou necessário efetuar mais nenhuma partição, sendo cada registro do conjunto de treinamento atribuído a alguma folha da AD.

O resultado do CART é a geração de uma AD binária univariada, de grande simplicidade e legibilidade, que pode ser percorrida da raiz até as folhas através de testes do tipo *"sim/não"*, podendo ser verificada a classe e a taxa de erro de cada folha. O CART permite resultados bastante superiores aos obtidos pelas técnicas estatísticas clássicas.

Segundo Fonseca [FON 94], os critérios utilizados no CART são:

a) *eleição do melhor atributo:* efetuada, principalmente, com base no critério de Gini, com opção também para entropia e paridade;

b) *tratamento de atributos discretos:* forma dois subconjuntos de possíveis valores, através da criação de nodos binários;

c) *tratamento de atributos contínuos:* utiliza a combinação linear para particionar nodos contínuos, realizando a pesquisa por um conjunto de coeficientes que minimizem o critério de partição definido previamente;

d) *tratamento de valores desconhecidos:* utiliza uma estratégia mais complexa, conhecida como *surrogate splits* [GAM 99]. Ao invés de armazenar em cada nodo apenas o atributo que minimiza a função de impuridade, o CART armazena um *ranking* de atributos que produzem uma partição similar. Quando está classificando um novo exemplo e o valor do melhor atributo é desconhecido, é verificado o valor de atributo com um valor conhecido segundo a ordem dada por este *ranking;*

e) *determinação da classe associada à folha:* efetuada pela atribuição da classe mais provável ou pela minimização dos custos;

f) *método de poda:* efetua a poda por redução do fator custo-complexidade, após a geração total da árvore.

3.1.3 Algoritmo C4.5

Apresentado por Ross Quinlan [QUI 93], o C4.5 visa a geração de árvores de decisão com tratamento de atributos contínuos e discretos, construindo uma árvore com um número de partições variável e com as folhas sendo indicadas pelos valores do atributo categórico.

Para evitar a geração de todas as árvores possíveis, o algoritmo C4.5 se baseia no atributo mais informativo, escolhido entre todos os atributos ainda não considerados no caminho desde a raiz. O algoritmo seleciona como sendo o atributo mais informativo aquele que possuir o maior *ganho de informação*, resultante da diferença do *valor da informação* do atributo categórico e do *valor da informação* (entropia) do atributo em questão.

Para cada atributo é calculado o seu ganho de informação. O atributo que tiver o maior ganho de informação será considerado pelo algoritmo como o próximo nodo da árvore. Assim, a partição começa pelo nodo raiz e continua pelos nodos filhos da mesma maneira, até que todos os exemplos desta partição possuam a mesma classe, rotulando-se este nodo como folha e recebendo sua respectiva classe.

O núcleo do C4.5 [QUI 93] é apresentado na Figura 3.7.

```
Função C4.5   (R: conjunto de atributos não categóricos,
                  C: atributo categórico,
               S: conjunto de treino): árvore de decisão
Início
        Se S está vazio
                retorna um nodo com valor negativo (falha);
                Se S contém exemplos de mesma classe
                retorna uma folha com o valor do atributo categórico;
        Se R está vazio
                retorna uma folha com o valor mais freqüente do atributo categórico;
                Senão
                calcula o atributo D com maior ganho de informação,  entre os
                                        atributos restantes de R;
                        acrescenta nodo D;
                divide S em subconjuntos de acordo com os valores de D;
                        chama novamente a Função C4.5 para cada um desses valores;
Fim.
```

FIGURA 3.7 – Núcleo do C4.5
Fonte: FELDENS, 1997. p.33

O algoritmo recebe o atributo categórico *C* (que indica a classe), um conjunto de atributos não categóricos *R* (demais atributos da tabela), e um conjunto de treinamento *S* (que contém os exemplos da tabela). Ele verifica qual é o atributo *D* mais informativo de *R*, ou seja, o atributo com maior ganho de informação para o conjunto de treinamento *S* e, então, subdivide este conjunto *S* de acordo com cada um dos valores deste atributo mais informativo *D* [FEL 97].

O algoritmo é chamado recursivamente para cada subconjunto obtido, até que:

a) não haja mais exemplos para classificar no conjunto de treinamento *S*. Neste caso, a árvore de decisão é um nodo com valor negativo;

b) os exemplos no conjunto de treinamento *S* pertençam todos à mesma classe de *C*. A árvore de decisão é uma folha identificando a classe C_i;

c) não haja mais atributos não categóricos *R* para a classificação. A árvore é uma folha identificando a classe C_i mais freqüente de *S*.

Quando o conjunto de treinamento *S* contém casos pertencentes a várias classes, a idéia é refinar *S* em subconjuntos de casos, que tendem a pertencer a apenas uma classe. A árvore de decisão para *S* consiste de um nodo de decisão que identifica o teste e uma ligação de cada valor possível do atributo. O mesmo processo de construção é aplicado recursivamente em cada subconjunto de casos de treinamento até que a *i-ésima* ligação tenha a árvore de decisão construída do subconjunto S_i de casos de treinamento. Quando o algoritmo parar, basta percorrer os caminhos desde a raiz até as folhas para verificar as descobertas extraídas da base de dados.

Concluindo, os critérios presentes no algoritmo C4.5 são:

a) *eleição do melhor atributo:* utiliza o critério do ganho de informação;

b) *tratamento de atributos discretos:* atribui uma ligação distinta a cada valor do atributo ou forma agrupamentos de valores em vários conjuntos;

c) *tratamento de atributos contínuos:* utiliza a técnica do teste simples para a partição, escolhendo como ponto de cisão o ponto médio entre os valores;

d) *tratamento de valores desconhecidos:* desconsidera os atributos com valores desconhecidos, utilizando apenas aqueles com valores totalmente conhecidos;

e) *determinação da classe associada à folha:* é efetuada por atribuição da classe mais provável nesta folha;

f) *método de poda:* utiliza a técnica de pós-poda baseada no erro, examinando a árvore de forma *bottom-up* e substituindo uma subárvore por uma folha;

g) *complexidade do algoritmo:* é dada por $O(mn^2)$, em que *m* é o número de atributos e *n* é o número de instâncias do conjunto de treinamento.

3.2 Regras de Classificação

As *regras de classificação, regras de produção* ou *regras de decisão* são estruturas apresentadas na forma de regras do tipo *"Se <condição> então <conclusão>"*.

A parte esquerda da regra, conhecida por *condição*, *premissa*, *corpo da regra* ou *complexo*, é uma conjunção de testes de atributos na forma: X_i *op* A, onde X_i é um atributo, *op* é um operador pertencente ao conjunto $\{=, \neq, >, \geq, <, \leq\}$ e A é um valor constante válido para X_i. Já a parte direita é a *conclusão* ou *cabeça da regra,* sendo constituída de uma classe C_i.

As regras podem ser também *disjuntas*, ou seja, somente uma é disparada quando uma nova instância não rotulada é classificada. Apesar disto, nenhum *"Senão"* explícito precisa realmente ser incluído.

Assim, as regras são uma disjunção de condições conjuntivas, sendo a condição formada por um ou mais pares de atributo-valor, em que a Forma Disjuntiva Normal (FDN) é a representação mais utilizada por muitos indutores de regras.

As regras de classificação são comumente usadas para representar conhecimento em sistemas especialistas, e podem ser facilmente interpretadas por especialistas humanos por causa da sua grande modularidade, ou seja, uma simples regra pode ser interpretada sem a necessidade de outras regras.

Existem algoritmos como o AQ1, de Michalski apud [FEL 97], que realizam a indução de regras diretamente a partir do conjunto de treinamento. Outros, entretanto, geram as regras a partir de uma árvore de decisão, cujo processo é chamado de *derivação* de regras, rescrevendo a árvore como uma coleção de regras.

A partir da derivação, as árvores de decisão podem ser facilmente representadas através de regras de classificação*,* traduzindo os conceitos aprendidos por uma árvore de decisão em uma forma mais compreensível, em que cada regra representa um caminho da raiz até uma folha da árvore, sendo a condição formada pelos nodos da árvore e a conclusão pela folha do respectivo caminho.

Em relação ao tamanho, há tantas regras quanto folhas. Outra característica é que a eficiência de classificação das árvores de decisão é mantida, quando transformadas em regras de classificação.

Na Figura 3.8, tem-se a árvore de decisão da Figura 3.2 representada através de regras.

Se lágrimas = reduzida Então classe = sem lentes
Se lágrimas = normal E Astigmático = não Então classe = gelatinosas
Se lágrimas = normal E Astigmático = sim E Prescrição = míope Então classe = duras
Se lágrimas = normal E Astigmático = sim E Prescrição = hipermíope Então classe = sem lentes

FIGURA 3.8 – Exemplo de regras de classificação

A possibilidade de derivar regras de uma ferramenta de DCBD representa um importante aspecto da mesma, tendo em vista que grandes árvores de decisão são difíceis de entender. Dessa forma, as regras possuem a vantagem da modularidade e, conseqüentemente, da interpretabilidade.

3.2.1 Construção de regras

Como dito anteriormente, algumas regras são soluções traduzidas de árvores de decisão, segundo os quais uma árvore é primeiramente gerada e, então, derivada em um conjunto de regras. O processo de derivação é repetido para cada caminho da árvore, em que cada regra cobre um conjunto de instâncias que iniciam com a classe específica.

As regras também podem ser geradas independentemente das árvores de decisão. Neste processo, cada regra cobre um subconjunto de exemplos que iniciam com uma classe específica. O resultado da indução é uma lista de regras, também conhecida como *lista de decisão*.

Após a indução, cada regra pode ser simplificada removendo condições que não ajudam a diferenciar a classe nominal de outras classes, usando a precisão da regra. A regra é avaliada como se uma condição fosse excluída. A condição que, abandonada, produzir uma diminuição na taxa de erro da regra, será eliminada. Depois, é verificado se existem regras idênticas, sendo removidas as duplicidades. As regras são, então, agrupadas de acordo com a sua classe. Neste momento, pode ser realizada também uma simplificação em cada grupo, em que o conjunto de regras é simplificado através da eliminação de regras, nos casos em que a remoção não diminui a precisão do conjunto completo.

O algoritmo C4.5 possui uma variação, o C4.5rules, que realiza esta simplificação: remove uma por uma as condições das regras derivadas e avalia cada regra resultante pela taxa de erro. A melhor condição a ser removida será aquela que, mediante sua ausência, permitir uma taxa de erro menor. O processo é repetido até não ser possível deletar nenhuma condição da regra sem aumentar sua taxa de erro. Então, para cada classe, todas as regras simplificadas são avaliadas em ordem para remover aquelas que não contribuem com a acurácia de todo o conjunto de regras.

Conforme Baranauskas [BAR 2000a], um algoritmo de indução de regras pode induzir as regras através de duas formas: regras ordenadas ou não ordenadas.

3.2.1.1 Indução de regras ordenadas

A indução de regras ordenadas trabalha de uma forma iterativa, pesquisando por uma condição que cubra um grande número de instâncias de uma simples classe C_i e poucas das outras classes C_j, $j \neq i$. Tendo encontrado uma boa condição, as instâncias cobertas pela condição (independentemente da classe) são removidas do conjunto de treinamento, e a regra *"Se <condição> então classe=<C_i>"* é adicionada no final da lista de regras. Este processo continua até não serem encontradas mais condições para esta classe.

Considerando uma primeira regra descoberta, todas as instâncias que satisfazem a condição da mesma e a sua classe, bem como as instâncias que satisfazem a condição da regra e não satisfazem a classe, são removidas do conjunto de treinamento e um *"Senão"* pode ser introduzido na lista de regras, antes de induzir uma próxima regra. Depois disso, a próxima regra é induzida pelo algoritmo e o processo continua.

A última regra na lista de regras é a regra *default*, que simplesmente predita a classe que mais aparece nos dados de treinamento [CLA 89]. Os resultados em regras não classificam, necessariamente, todo o conjunto de treinamento corretamente, mas classificam bem novos dados [CLA 89].

Para classificar novas instâncias, o classificador testa cada regra em ordem até ser encontrada alguma cujas condições sejam satisfeitas pela nova instância. A classe predita por esta regra é então atribuída a esta nova instância.

Por isso, a ordem das regras é importante. Entretanto, se nenhuma regra for satisfeita, a regra *default*, que é a regra com a classe mais freqüente do conjunto de treinamento, será atribuída a este exemplo.

3.2.1.2 Indução de regras não ordenadas

A indução de regras não ordenadas é similar à indução ordenada, porém a principal diferença é que as instâncias das classes C_j, $j \neq i$ incorretamente cobertas pela atual condição deverão permanecer no conjunto de treinamento, enquanto as instâncias cobertas tendo a classe C_i deverão ser removidas para não ser encontrada a mesma regra novamente. Dessa forma, somente são removidas as instâncias que satisfazem tanto a condição quanto a classe da regra.

Neste tipo de indução, uma complicação causada é que as regras deixam de ser mutuamente exclusivas, podendo um caso ser satisfeito por mais de uma regra. Assim, para classificar novos exemplos em um classificador não ordenado, todas as regras são testadas, e aquelas que possuem suas condições satisfeitas pelo novo exemplo são selecionadas. Se mais de uma classe for preditada pelas regras selecionadas, deverá ser utilizado algum mecanismo de escolha entre elas.

O método geralmente utilizado é verificar a distribuição, ou seja, o número de exemplos cobertos em cada regra para cada classe, e então somar essas distribuições. A classe que possuir a maior distribuição pela soma nas regras selecionadas será a classe da nova instância.

Por exemplo, considere as regras da Figura 3.9 e a distribuição das classes [amigo, inimigo] para cada uma:

Se segura = balão	Então classe = amigo	[15, 2]
Se sorri = não	Então classe = inimigo	[1, 10]
Se cabeça = quadrada	Então classe = inimigo	[0, 8]

FIGURA 3.9 – Exemplo de regras não ordenadas

Para classificar uma nova instância que possui os seguintes valores: *{balão, não sorri, redonda}*, seria percorrida toda a lista de regras e duas delas seriam selecionadas: a primeira e a segunda. Assim, as classes poderiam ser consideradas para classificar este novo robô: *amigo*, *inimigo*. A primeira regra selecionada cobre quinze instâncias para a classe *amigo* e duas para *inimigo*, enquanto que a outra regra cobre apenas um exemplo para a classe *amigo* e dez para a outra classe. Assim, a classe *amigo* possui dezesseis instâncias cobertas pelas regras selecionadas e a classe *inimigo* cobre doze instâncias. Dessa forma, a classe atribuída ao novo exemplo seria amigo, por possuir maior número de ocorrências.

Em geral, induzir regras é mais complexo do que induzir uma árvore de decisão, ou derivar regras a partir de uma AD. Diferentemente das árvores de decisão, as regras induzidas de forma não ordenada não são disjuntas, significando que a mesma instância pode ser coberta por diferentes regras.

Por este motivo, as regras não ordenadas devem fornecer um mecanismo a mais para resolver eventuais conflitos com a relação de ordem. Já as regras ordenadas não necessitam de nenhum controle adicional, mas têm a desvantagem da necessidade do cuidado em hierarquia das mesmas.

3.2.2 Algoritmo CN2

Desenvolvido por Clark e Niblett apud [BAR 2000b], em 1987, o CN2 é um algoritmo que induz regras utilizando o método de "dividir para conquistar", consistindo de dois procedimentos: um mecanismo de pesquisa por boas regras e um mecanismo de controle para executar repetidamente esta pesquisa. O CN2 usa uma função de avaliação para selecionar o atributo que ajuda a distinguir uma classe C_i das outras, e adiciona o teste resultante em uma regra conjuntiva. Este processo é repetido até que a regra exclua todos os exemplos de outras classes e remova os exemplos da classe que a regra cobre, repetindo este processo nos exemplos de treinamento remanescentes.

São examinadas todas as possíveis especializações de uma condição, para escolher a melhor, usando a entropia ou o critério de Laplace, como heurística de pesquisa. A cada iteração é pesquisada uma condição que cubra um grande número de exemplos. A condição é tanto preditiva quanto confiável, como determina a função de avaliação. Tendo encontrado uma boa condição, todas as instâncias cobertas são removidas do conjunto de treinamento, e a regra é adicionada à lista de regras. O processo continua até que nenhuma outra boa condição possa ser encontrada. É importante salientar que, na regra final, a condição é um conjunto de pares de atributo-valor.

O CN2 lida com atributos contínuos, dividindo a média dos valores de cada característica em subconjuntos discretos, utilizando o método do teste simples, gerando um par de atributo-valor. Este algoritmo também realiza tratamento para valores desconhecidos, utilizando o método de substituir esses valores pelo valor mais comum, caso o atributo desconhecido seja nominal. Já para atributos contínuos, o valor médio do intervalo mais comum substitui o valor desconhecido.

Este indutor gera tanto regras ordenadas quanto não ordenadas [BAR 2000b]. A Figura 3.10 apresenta o núcleo do algoritmo CN2 para a indução de regras ordenadas, enquanto que a Figura 3.11 apresenta a forma de indução não ordenada do CN2.

```
Função CN2 Ordenado  (S: conjunto de treino): ListaRegras
Início
        ListaRegras:={};
        Repetir
                MelhorCondição:=PesquisaMelhorCondição(S);
                Se MelhorCondição≠{ } Então
                        C:={classe mais freqüente em S que satisfaz MelhorCondição};
                        Regra:="SE MelhorCondição ENTÃO classe=C";
                        ListaRegras:=ListaRegras+Regra;
                        S:=S-{s que satisfazem MelhorCondição};
                Fim se;
        Até MelhorCondição={ };
Fim.
```

FIGURA 3.10 – Núcleo do CN2 (ordenado)
Fonte: BARANAUSKAS, 2000. p.37

A lista de regras, inicialmente vazia, recebe as regras induzidas. É realizada uma pesquisa para escolher a melhor regra para *S*. É adicionada a regra na lista de regras com a classe mais freqüente dos exemplos que satisfazem esta melhor condição, que, por sua vez, são removidos do conjunto de treinamento. O processo se repete até não ser encontrada mais nenhuma boa condição.

```
Função CN2 Não Ordenado  (S: conjunto de treino): ListaRegras
Início
        ListaRegras:={};
        Para todas as classes C em S Fazer
                RegrasParaClasse:={ };
                Repetir
                        MelhorCondição:=PesquisaMelhorCondição(S,C);
                        RegraParaClasse:=RegraParaClasse+"SE MelhorCondição
                                                        ENTÃO classe=C";
                        S:=S-{s que satisfazem MelhorCondição e C)};
                Até MelhorCondição={ };
        Fim para;
        ListaRegras:=ListaRegras+RegrasParaClasse;
Fim.
```

FIGURA 3.11 – Núcleo do CN2 (não ordenado)
Fonte: BARANAUSKAS, 2000. p.39

A lista de regras, também inicialmente vazia, recebe as regras induzidas de acordo com a regra. É realizada uma pesquisa para escolher a melhor condição para *S*. É adicionada a regra na lista de regras com a classe mais freqüente dos exemplos que satisfazem esta melhor condição e a classe, que, por sua vez, são removidos do conjunto de treinamento. Enquanto não forem cobertos todos os exemplos da classe, o processo é repetido.

Para classificar um novo exemplo utilizando as regras induzidas de forma ordenada, o CN2 classifica o novo exemplo pela primeira regra que dispara, desconsiderando todas as demais. Já para classificar novos exemplos nas regras não ordenadas, é feita uma avaliação de todas as regras e aquelas cujas condições são satisfeitas, são selecionadas. Se mais de uma regra for selecionada, o CN2 associará, a cada regra, a distribuição de exemplos cobertos entre as classes e totalizará essas distribuições para encontrar a classe mais provável, que será atribuída ao novo exemplo.

Em ambas as induções (ordenada e não ordenada) existe a regra *default* que é utilizada caso nenhuma regra seja selecionada.

3.2.3 Algoritmo Prism

O algoritmo Prism, de Centrowska [CEN 87], é um algoritmo para indução de regras modulares, possuindo muitas características empregadas no ID3.

Para cada regra é escolhida a melhor condição, de acordo com a heurística desejada, criando um subconjunto de treinamento que englobe esta condição. O próximo passo é identificar a melhor regra para o conjunto de instâncias que não são exemplos da primeira regra. Isto é feito removendo de *S* todas as instâncias que são cobertas pela regra. Isto é repetido até que não haja nenhuma instância da classe C_1 em S_1.

Quando as regras para uma classe foram induzidas, o conjunto de treinamento é restaurado ao seu estado inicial e o algoritmo é aplicado novamente para induzir o conjunto de regras para a próxima classe, de forma não ordenada. Como todas as classes são induzidas separadamente, a ordem de apresentação das mesmas é insignificante. Se todas as instâncias forem da mesma classe, então a classe será retornada como regra e o algoritmo terminará.

Segundo Centrowska [CEN 87], para cada classe C_i deverá ser realizado:

1) calcular a probabilidade $p(C_i \mid \alpha_i)$ de ocorrência da classe C_i para cada par de atributo-valor α_i;

2) selecionar o α_i para cada $p(C_i \mid \alpha_i)$ com maior ganho de informação;

3) criar um subconjunto de treinamento englobando todas as instâncias que contêm o α_i selecionado;

4) repetir os passos 1 a 3 para este subconjunto até ele conter somente instâncias da classe C_i. A regra induzida é a conjunção de todos os atributos usados na criação do subconjunto homogêneo de dados;

5) remover todas as instâncias cobertas por esta regra do conjunto de treinamento;

6) repetir os passos 1 a 5 para o subconjunto formado no passo 5, até todas as instâncias da classe C_i terem sido removidas;

7) restaurar o conjunto de treinamento e retornar ao passo 1 para a próxima classe.

3.3 Qualidade

Para que o resultado da Mineração de Dados possa ser utilizado com segurança, o objetivo da construção de um classificador é obter o menor erro de classificação possível. Para isso, é necessário utilizar métricas de avaliação para estimar esse valor com base nos exemplos disponíveis no conjunto de treinamento ou em dados ainda não apresentados.

Os modelos gerados por um algoritmo de aprendizado podem ser analisados de diferentes dimensões. Os parâmetros freqüentemente considerados são: a taxa de erro; a compreensibilidade e a compactividade, que estão relacionadas ao tamanho do modelo; o tempo de aprendizagem, que é o tempo necessário para o algoritmo induzir o classificador; e o tempo de classificação, que é o tempo necessário para classificar um exemplo de consulta. O parâmetro mais importante é a taxa de erro, que é computada como a proporção entre o número de exemplos não-classificados corretamente e o número total de exemplos.

Baranauskas [BAR 2000a] apresenta uma metodologia de avaliação de um sistema de indução de árvores de decisão:

1) agrupar um grande número de instâncias, todas com valores dos atributos e classes corretos;

2) dividir este conjunto randomicamente em dois conjuntos distintos: um conjunto de treinamento e um conjunto de teste;

3) aplicar o algoritmo de indução para o conjunto de treinamento, obtendo um classificador;

4) medir a performance do classificador gerado com o conjunto de teste através, simplesmente, do erro medido. Também podem ser analisados, entre outros, o tempo de aprendizado, o tamanho da árvore e a qualidade das regras;

5) para medir a eficiência e robustez do classificador, devem ser repetidas as etapas 2, 3 e 4 para diferentes conjuntos de exemplos e diferentes tamanhos do conjunto de treinamento.

Na teoria, para avaliar um classificador, devem ser realizadas estas atividades, construindo o conjunto de treinamento e o conjunto de teste randomicamente a partir de um grande conjunto de casos. Na prática, entretanto, é difícil obter tais volumosos conjuntos de exemplos. Os dados disponíveis são, geralmente, de tamanho limitado e utilizados tanto para construir o classificador quanto para avaliar a sua taxa de erro.

Segundo John [JOH 97], além de uma avaliação objetiva, é necessária uma avaliação qualitativa dos resultados encontrados, por parte do especialista. É possível que, mesmo com níveis de precisão aceitáveis, o especialista interprete os resultados da mineração da seguinte forma:

a) fica satisfeito com os resultados e surpreso com alguns dos padrões obtidos;

b) fica satisfeito com os resultados, mas percebe que já conhecia os padrões obtidos;

c) fica insatisfeito com os resultados.

No segundo caso, o modelo ainda pode ser utilizado como base para a construção de outro que forneça melhores resultados. No último caso, é necessária uma análise mais profunda dos resultados e a revisão do processo.

A estimativa da qualidade de um classificador consiste em estimar a percentagem de erro que se espera que o classificador venha a obter na classificação de exemplos futuros. Genericamente, a percentagem de erro pode ser dada pela fórmula:

$$\textit{Percentagem de erro} = \frac{\textit{Número de erros * 100}}{\textit{Número de casos testados}}$$

Assim, a medida mais utilizada é a *taxa de erro* de um classificador *h – ce(h)*, também conhecida como *taxa de classificação incorreta*, que compara a classe de cada instância y_i com o rótulo $h(x_i)$ do classificador, tendo como *n* o número de instâncias. Assim, a taxa de erro é dada por [BAR 2000a]:

$$err(h) = \frac{1}{n}\sum_{i=1}^{n} \| y_i \neq h(x_i) \|$$

A operação $\| y_i \neq h(x_i) \|$ retorna 1 se a comparação for verdadeira e 0 se for falsa.

Já o complemento da taxa de erro, a *acurácia* ou *precisão* do classificador, é dada por:

$$ac(h) = 1 - err(h)$$

Dessa forma, considera-se acurácia a proporção de objetos corretamente classificados, usada para determinar quão bom um modelo será para dados não vistos. Já erro é a proporção de objetos não classificados.

As medidas de desempenho de um classificador efetuadas sobre o conjunto de treinamento são comumente designadas como *aparentes*. Já as medidas efetuadas sobre o conjunto de teste são conhecidas como *reais* ou *verdadeiras* [BAR 2001]. Desse modo, a medida aparente é um estimador ruim do desempenho futuro do classificador, uma vez que tem a tendência de ser otimista, pois utiliza o próprio conjunto de treinamento para fazer a avaliação.

A seguir são apresentadas algumas técnicas de estimativa de erros, de custos, de suporte e de confiança, de acordo com Fonseca [FON 94] e Baranauskas [BAR 2001].

3.3.1 Estimativa por ressubstituição

A *estimativa por ressubstituição* ou *erro aparente,* de Breiman apud [FON 94], consiste na proporção de resultados incorretos quando o conjunto de treinamento é novamente apresentado para a classificação, depois que o classificador foi construído utilizando o mesmo, ou seja, o conjunto de treinamento e de teste são idênticos. A Figura 3.12 mostra a relação entre o conjunto de treinamento e o conjunto de teste.

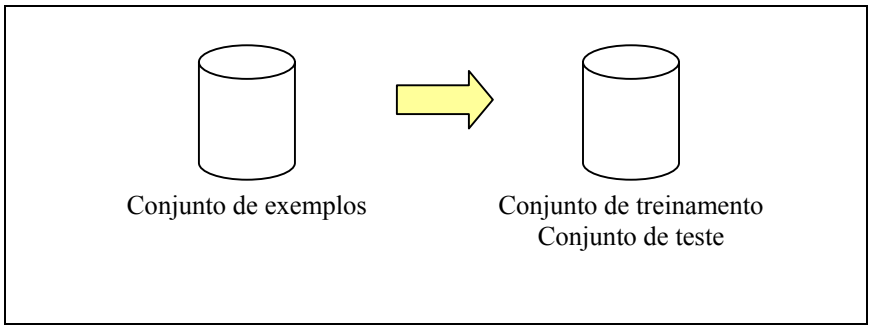

FIGURA 3.12 – Conjuntos usados na estimativa por ressubstituição

A estimativa por ressubstituição é dada por:

$$err(h) = \frac{1}{n}\sum_{i=1}^{n} \| y_i \neq h(x_i) \|$$

onde:

n:	número de exemplos
y_i:	classe correspondente ao exemplo *i*
$h(x_i)$:	rótulo do classificador para o exemplo *i*

3.3.2 Estimativa por conjunto independente

Desenvolvido por Breiman apud [FON 84], a *estimativa por utilização de conjunto independente* avalia a taxa de erro a partir de um conjunto de exemplos independente. O conjunto de exemplos inicial é dividido em dois subconjuntos, como mostra a Figura 3.13, um para o treinamento e outro para a estimativa do erro. Geralmente, o conjunto de exemplos para o treinamento possui 2/3 dos exemplos totais, escolhidos de forma aleatória, enquanto que o conjunto de teste possui 1/3 restante. Embora este percentual seja bastante utilizado, não há nenhum fundamento teórico em que seja baseado.

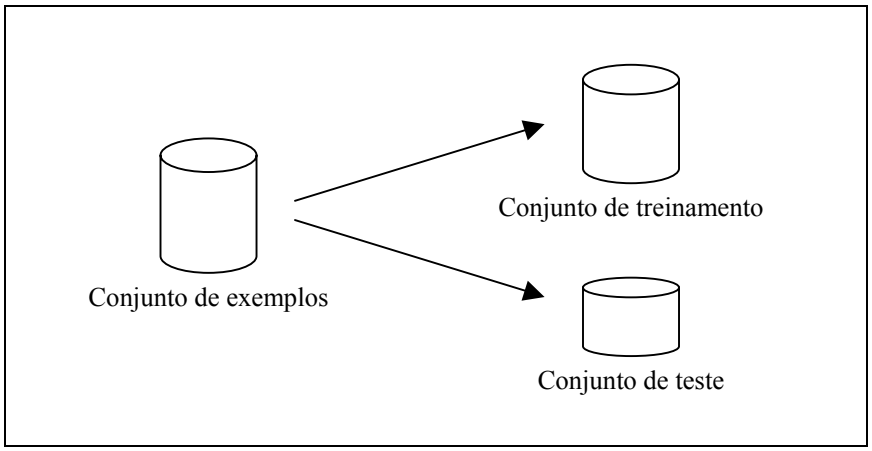

FIGURA 3.13 – Conjuntos usados na estimativa por conjunto independente

Assim, a estimativa por conjunto independente é dada por:

$$err(h) = \frac{1}{n_2} \sum_{i=1}^{n} \| y_i \neq h(x_i) \|$$

onde:

n_2: número de exemplos do conjunto de teste

Esse método reduz a dimensão do conjunto de treinamento, o que pode ser prejudicial em casos nos quais o número de exemplos for pequeno.

3.3.3 Estimativa por custos

Em algumas aplicações, existem diferentes custos associados aos vários erros possíveis. O custo de um erro é a penalização imposta ao sistema no caso deste cometer algum tipo de erro de classificação.

Assim, ao invés de minimizar a taxa de erro, o objetivo é minimizar o custo de erro de classificação. Para tal, é utilizada uma *matriz de custos* (Tabela 3.2), que contém o custo de cada tipo de erro possível, de acordo com a classe [BAR 2001]. A combinação da coluna da classe real com a linha da classe atribuída ou predita representa o custo para cada ocorrência de erro.

TABELA 3.2 – Exemplo de uma matriz de custos

Classe Atribuída	Classe real		
	A	B	C
A	0	2	12
B	10	0	8
C	5	6	0

Juntamente é utilizada uma *matriz de confusão* (Tabela 3.3), que representa o número de exemplos por classe obtida como resultado do teste do classificador.

TABELA 3.3 – Exemplo de uma matriz de confusão

Classe Atribuída	Classe real		
	A	B	C
A	92	6	2
B	5	89	6
C	10	15	75

O *custo do classificador,* será, então, dado por:

$$Custo = \sum_{i=1}^{k} \sum_{j=1}^{k} C_{i,j} M_{i,j}$$

onde:

$C_{i,j}$:	valor da linha *i*, coluna *j* da matriz de custo
$M_{i,j}$:	valor da linha *i*, coluna *j* da matriz de confusão
k:	número de classes

Conseqüentemente, o *custo médio* será:

$$Custo\ médio = \frac{Custo}{Número\ de\ classificações}$$

Dessa forma, para o exemplo das Tabela 3.1 e 3.2, o valor do custo médio seria:

$$\text{Custo médio} = \frac{0*92+2*6+12*2+10*5+0*89+8*6+5*10+6*15+0*75}{300}$$

Custo médio = 0.91

Este método leva em consideração apenas os custos provenientes do erro, não tendo em conta o ganho das classificações corretas.

3.3.4 Estimativa por validação cruzada

Na *estimativa por validação cruzada, por camadas,* ou *cross-validation,* de Breiman apud [FON 94], os exemplos do conjunto de treinamento são divididos randomicamente em *v* subconjuntos contendo, aproximadamente, o mesmo número de exemplos.

Cada subconjunto é utilizado uma única vez para a estimativa do erro de classificação, enquanto constrói-se o classificador utilizando todos os outros *v-1* subconjuntos como um único conjunto de treinamento. O algoritmo gera um modelo do conjunto de treinamento e utiliza o conjunto de teste para classificar os exemplos. Este processo é realizado para as *v* partições, ou seja, cada subconjunto será utilizado como conjunto de teste. Assim, é realizada uma estimativa por utilização de um conjunto independente para cada um dos classificadores parciais. O valor estimado para o erro será dado pela média dos erros estimados para cada um dos classificadores parciais. Abaixo, na Figura 3.14, está apresentada a relação do conjunto de exemplos com o conjunto de treinamento e teste.

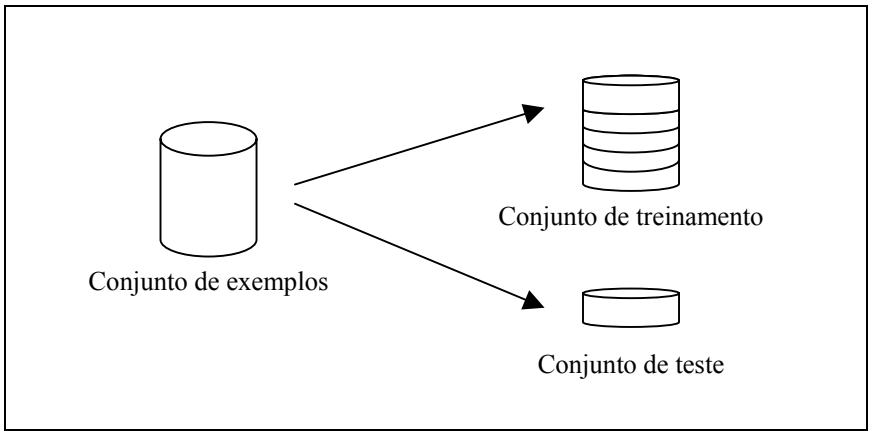

FIGURA 3.14 – Conjuntos usados na estimativa por validação cruzada

A estimativa de erro de cada um dos *v* classificadores é dada por:

$$err\left(h^V\right) = \frac{1}{n_V} \sum_{i=1}^{n} \left\| y_i \neq h\left(x_i\right) \right\|$$

Assim, o erro final estimado será:

$$err\left(h\right) = \frac{1}{v} \sum_{i=1}^{v} err\left(h_i\right)$$

Geralmente, o conjunto inicial de treinamento é dividido em dez subconjuntos. Assim, todos os exemplos são utilizados nove vezes para a geração de um classificador e uma vez para o teste da sua eficiência.

Este método possui uma tendência pessimista, dado que cada um dos classificadores utilizados para a estimativa é construído apenas com um subconjunto do treinamento. Já o classificador final, que será aquele que obtiver a menor taxa individual de erro, terá uma qualidade superior à média dos classificadores parciais.

Segundo Fonseca [FON 94], esta estimativa é ideal para conjuntos de treinamento com aproximadamente cinqüenta exemplos, pois como a sua complexidade é grande, este método não é adequado para conjuntos de grandes dimensões.

3.3.5 Estimativa por *bootstrapping*

Na *estimativa por bootstrapping,* é formado um conjunto de treinamento a partir do conjunto original, contendo o mesmo número de registros escolhidos aleatoriamente. Um exemplo pode ser escolhido mais de uma vez e os casos que não se encontrarem no conjunto de treinamento constituirão o conjunto de teste.

O algoritmo de aprendizado gera um modelo a partir do conjunto de treinamento, que é utilizado para classificar os exemplos do conjunto de teste. O processo é repetido várias vezes gerando diferentes conjuntos de treinamento e de teste, e, conseqüentemente, classificadores.

A estimativa mais comum por *bootstrapping* é a *e0,* em que o erro obtido pelo classificador será a média das várias iterações deste algoritmo (geralmente de duzentas iterações):

$$e0(h) = \frac{1}{v}\sum_{i=1}^{v} err(h_i)$$

Como pode ser observado, é um método de grande complexidade. Deve ser utilizado somente para conjuntos de pequenas dimensões.

3.3.6 Suporte e confiança

O suporte e a confiança são medidas comumente utilizadas para avaliar a qualidade de uma regra [BAR 2000a].

O *suporte* é uma medida usada para avaliar a significância estatística de um padrão para o espaço de busca, ou seja, indica o percentual de casos suportados por uma determinada regra:

$$Suporte = \frac{n_2}{n}$$

Já a *confiança* é a probabilidade condicional de eventos associados com uma regra particular, indicando o acerto da regra:

$$Confiança = \frac{n_2}{n_1}$$

onde*:*

n:	número total de instâncias
n_1*:*	número de instâncias que satisfazem as condições
n_2*:*	número de instâncias que satisfazem as condições e a classe

Por exemplo, tem-se um conjunto de treinamento *S* com 20 registros e uma determinada regra *R*. Neste conjunto, existem 10 registros que satisfazem a condição de *R*, mas apenas 5 que satisfazem tanto a condição quanto a classe predita por *R*. O suporte desta regra será 0,25 e a confiança 0,50 - ou seja, apenas 25% dos exemplos do conjunto de treinamento são suportados corretamente pela regra, enquanto que 50% dos casos cobertos pela regra são classificados corretamente.

Assim, essa medida é usada para estimar a qualidade de uma regra depois de induzida ou no momento de indução da mesma, funcionando como um método de poda de regras.

3.4 Comparação entre algoritmos

O objetivo da avaliação de dois algoritmos para um mesmo conjunto de dados é verificar qual deles generaliza melhor estes dados. Assim, a taxa de erro estimada, usando os métodos descritos anteriormente, é usada como medida da performance do algoritmo.

Porém, de acordo com Gama [GAM 99], existe uma grande variabilidade na taxa de erro, que se origina:

a) *na randomicidade interna dos algoritmos de aprendizagem:* alguns algoritmos têm estruturas internas de dados que são inicializadas de forma randômica. Assim, duas diferentes execuções do algoritmo com as mesmas condições podem produzir diferentes resultados;

b) *na variabilidade do conjunto de treinamento:* o modelo induzido é instável no que se refere a pequenas variações do conjunto de treinamento, ou seja, pequenas mudanças no conjunto de treinamento podem levar a diferentes modelos;

c) *na variabilidade do conjunto de teste:* como o erro de generalização é estimado no conjunto de teste, se houver quantidade suficiente de dados, a variabilidade do conjunto de teste não afeta muito a variabilidade da taxa de erro estimada. Para pequenas bases de dados, entretanto, isso tem fundamental importância e pode levar a conclusões errôneas;

d) *nos parâmetros do ambiente:* na maioria dos algoritmos, o usuário pode controlar o valor de alguns parâmetros, como o nível de poda. Estes diferentes valores podem levar a distintos modelos e estimativas da taxa de erros.

Dessa forma, quando comparamos algoritmos através da verificação apenas da taxa de erro de classificação, não é fácil saber se um determinado algoritmo é melhor que o outro.

O tempo de aprendizado também é utilizado para comparar classificadores. Essas comparações são difíceis de serem feitas, pois os resultados dependem muito dos detalhes de implementação, bem como do hardware suportado. Entretanto, a menor ordem de magnitude de complexidade do tempo é um indicador de sucesso, que pode indicar um classificador melhor.

Dados dois modelos com o mesmo erro de generalização, também o mais simples deverá ser preferido devido à simplicidade desejável. O principal problema é como medir a simplicidade. Geralmente pode-se utilizar o número de nodos, o número de folhas, o número de regras ou o número de condições, como medida de complexidade, ou seja, quanto menores estes números, maior a simplicidade do classificador.

Segundo Baranauskas [BAR 2000a], uma alternativa para a comparação de diferentes algoritmos aplicados em um mesmo domínio é calcular, primeiramente, a *média e o desvio padrão* de cada algoritmo isoladamente, utilizando o erro obtido através da técnica de validação cruzada:

$$media(A) = \frac{1}{v}\sum_{i=1}^{v} err(h_i)$$

$$desvio(A) = \sqrt{\frac{1}{v}\left(\frac{1}{v-1}\sum_{i=1}^{v}(err(h_i) - media(A))^2\right)}$$

onde:
$err(h_i)$: taxa de erro para cada classificador
v: número de classificadores gerados

Para ficar mais claro, imagine um algoritmo A_s com validação cruzada de dez interações e com os seguintes erros para cada uma delas: {5.5, 11.4, 12.7, 5.20, 5.90, 11.30, 10.90, 11.20, 4.90, 11.00}. Assim, tem-se a *media* $(A_s) = 9.00$ e o *desvio* $(A_s) = 1.00$.

Geralmente, o erro é representado pela média seguida do símbolo "±", que, por sua vez, é seguido pelo desvio padrão. No exemplo acima, o erro seria definido por: 9.00 ± 1.00.

Como o desvio padrão pode ser visto como a robustez do algoritmo, pode-se, a partir dele, determinar se a diferença entre os algoritmos é significativa ou não. Para isso, é calculada a *diferença absoluta* entre os algoritmos, a partir da média e do desvio padrão, citado por [BAR 2001]:

$$media(A_s - A_p) = media(A_s) - media(A_p)$$

$$desvio(A_s - A_p) = \sqrt{\frac{desvio(A_s)^2 + desvio(A_p)^2}{2}}$$

$$da(A_s - A_p) = \frac{media(A_s - A_p)}{desvio(A_s - A_p)}$$

onde:
A_s: algoritmo padrão *(standard)*
A_p: algoritmo proposto

Se $da(A_s–A_p)>0$, então A_p tem melhor qualidade do que A_s. Porém, se $da(A_s–A_p)\geq 2$ desvios padrões, então A_p supera A_s, com 95% de confiança. Por outro lado, se $da(A_s–A_p)\leq 0$, então A_s tem melhor qualidade do que A_p. Se $da(A_s–A_p)\leq -2$, então A_s supera A_p, com grau de confiança de 95%.

Por exemplo, assume-se que $A_s=9.00\pm1.00$ e $A_p=7.50\pm0.80$. Assim, tem-se: $media(A_s–A_p)=1.50$, $desvio(A_s–A_p)=0.91$ e $da(A_s–A_p)=1.65$. Logo, $da(A_s–A_p)>0$ e $da(A_s–A_p)<2$, ou seja, A_p supera A_s, mas não é significativamente melhor que A_s, com grau de confiança de 95%.

Já para a obtenção de um *ranking* de vários algoritmos, não basta apenas verificar a taxa de erro em um determinado domínio. Uma alternativa, segundo Gama [GAM 99], seria calcular a taxa média de erro para todos os conjuntos de dados e algoritmos, e então ordenar os algoritmos de acordo com esses valores. Porém, esta não é uma solução muito satisfatória, principalmente porque as taxas de erro não são comparáveis entre diferentes conjuntos de dados. Uma taxa de erro de 5% em um conjunto de dados pode ser um excelente resultado, mas pobre em outro. Uma solução mais satisfatória seria armazenar as posições de todos os algoritmos para todos os conjuntos de dados e calcular posições médias.

4 Modelo proposto

Como visto anteriormente, as Árvores e Regras de Decisão são importantes métodos utilizados na Mineração de Dados, populares por sua simplicidade, flexibilidade e interpretabilidade. Os algoritmos existentes para a indução destes classificadores são relativamente simples, mas o processamento pode tornar-se mais demorado, devido à sucessiva divisão e percurso do conjunto de treinamento. Além disto, o classificador final pode ser muito grande e propenso ao *overfitting* dos dados.

Outro aspecto interessante deste métodos é que tanto as árvores quanto as regras de decisão podem ser aplicadas em diversos domínios. Entretanto, os conjuntos de dados reais para aplicação na Mineração de Dados são geralmente muito grandes, envolvendo milhares de registros. Executar a tarefa de classificação em tais dados requer, cada vez mais, o desenvolvimento de novas técnicas e/ou o aperfeiçoamento das existentes, de forma a induzir um classificador que generalize esta grande quantidade de dados, minimizando, juntamente, o tempo de execução.

Dentro deste contexto, é proposto neste trabalho um modelo para a indução de regras e árvores de decisão que procura generalizar os dados, percorrendo os mesmos uma única vez, o que consiste no principal diferencial do modelo apresentado.

4.1 Inspiração

A Inteligência Artificial Distribuída (IAD) se baseia no comportamento social de indivíduos, principalmente nas ações e interações entre eles, tendo como principal objetivo a construção de sistemas inteligentes formados por entidades autônomas, denominadas *agentes*, dirigidos a objetivos próprios e possuindo uma existência própria, independente de outros agentes [FRO 97].

Em um sistema, "um *agente* é uma entidade ativa e um conjunto de agentes forma uma *sociedade*. As entidades passivas são consideradas *ambiente*. Um agente recebe informações e pode raciocinar sobre o ambiente e sobre outros agentes para, então, decidir racionalmente quais objetivos e ações executar" [ALV 97].

Segundo Ferber apud [ALV 97], um agente pode ser considerado também uma entidade real ou virtual, inserida em um ambiente sobre o qual é capaz de agir, dispondo de capacidade de percepção e de representação parcial deste ambiente, podendo se comunicar com outros agentes e possuindo um comportamento autônomo, que é conseqüência de suas observações, conhecimento e interações com outros agentes.

A IAD enfatiza o trabalho conjunto de uma sociedade de agentes, organizados para trabalharem em conjunto solucionando problemas. A resolução de tais problemas é mais adequadamente estruturada através de modelos distribuídos, sendo o comportamento e as características do grupo de agentes pontos importantes para a construção de sistemas inteligentes avançados.

A IAD pode ser dividida em duas sub-áreas: Resolução Distribuída de Problemas (RDP) e Sistemas Multiagentes (SMA). Enquanto a primeira tem como estratégia a divisão dos procedimentos a serem executados para a solução de um problema em conjunto, a segunda possui um conjunto de agentes autônomos, que cooperam entre si para a solução de um problema [ALV 97].

A idéia de distribuir inteligência entre um conjunto de agentes é necessária e adequada em sistemas grandes e complexos. Muitas vezes, estes sistemas são constituídos de um único módulo, sendo difíceis de desenvolver e evoluir, além de trazerem complexos problemas de manutenção.

A necessidade de operações eficientes de acesso aos dados e de escalabilidade dos algoritmos de extração de padrões representa um grande problema para os sistemas de Mineração de Dados. Uma forma de tornar mais efetiva a fusão entre o banco de dados e a tecnologia de extração de informação é paralelizando estas operações através do desenvolvimento de agentes distribuídos para a mineração de dados.

Existem, atualmente, várias pesquisas sendo realizadas no desenvolvimento de ferramentas e tecnologias para a utilização de agentes na Descoberta de Conhecimento. Esses sistemas de Descoberta de Conhecimento baseados em agentes – como o PADMA, de Kargupta, e o JAM, de Stolfo – fornecem uma nova forma de executar a mineração de dados sobre dados distribuídos, oferecendo um bom tempo de resposta ao usuário final, bem como autonomia e versatilidade ao sistema [HAL 2001].

Pode-se utilizar os agentes tanto no processo de Descoberta de Conhecimento, no qual cada agente é responsável por uma determinada etapa do processo, quanto na evolução e/ou adaptação de técnicas específicas de Mineração de Dados, utilizando apenas modelos da área de Sistemas Multiagentes.

Por exemplo, pode-se empregar agentes que navegam através de um ambiente rico em informações. Esses sistemas empregam técnicas de aprendizado para observar as tarefas executadas pelos usuários, aprendendo a identificar um perfil de interesse do usuário e a pesquisar por informações relacionadas a ele em uma ampla variedade de fontes de informação disponíveis na Web.

Assim, o interesse na utilização da abordagem de agentes em diversas áreas, especialmente na Descoberta de Conhecimento, provém da necessidade de aplicar novas técnicas para a construção de sistemas, proporcionado resultados satisfatórios e sistemas mais robustos, eficazes e velozes.

Inicialmente, a idéia do trabalho era induzir árvores de decisão utilizando agentes para auxiliar no processo. Assim, o modelo proposto para a indução de árvores de decisão foi inspirado em um SMA: no modelo de Redes de Contrato, de Smith [SMI 80], em que a distribuição de atividades do sistema é um processo iterativo que ocorre através da negociação entre agentes autônomos. A rede consiste de um conjunto de agentes, que negociam entre si através da troca de mensagens. Neste modelo pode-se identificar três tipos de agentes:

a) *gerente:* é o agente que identifica a atividade a ser satisfeita e designa um agente para executá-la;

b) *proponentes:* são os agentes que apresentam uma proposta para executar uma atividade;

c) *contratado:* refere-se ao agente-proponente, cuja proposta foi aceita pelo gerente e que, conseqüentemente, passa a ser o responsável pela execução da atividade.

Neste modelo (Figura 4.1), um agente-gerente solicita algum tipo de serviço e difunde uma requisição de tarefa, para a qual os demais agentes enviam propostas com ofertas. O gerente usa, então, uma função de avaliação para escolher a melhor entre as ofertas recebidas. O agente-proponente escolhido passa a ser o agente-contratado, responsável pela execução da tarefa.

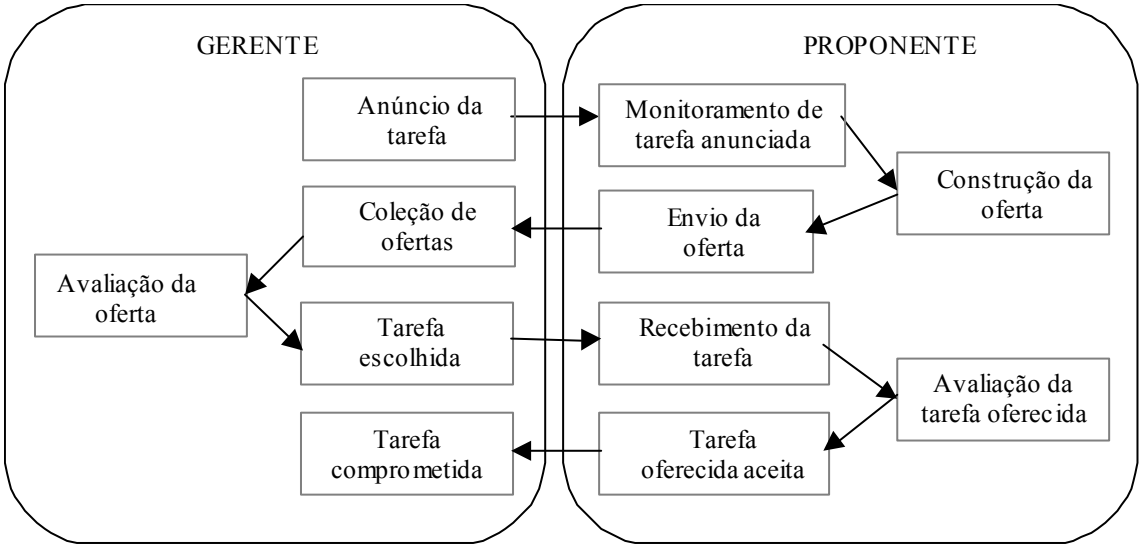

FIGURA 4.1 – Protocolo de Rede de Contrato

De acordo com a Figura 4.1, os agentes comunicam-se através das seguintes classes de mensagens:

a) o gerente divulga uma requisição de atividade, descrevendo a atividade a ser realizada e os critérios para propostas;

b) os proponentes enviam propostas, demonstrando seu interesse e habilidade para executar a atividade;

c) o gerente envia uma mensagem de aprovação ao proponente vencedor, escolhido pela função de avaliação, comunicando que a sua proposta foi aprovada para a execução da atividade;

d) o contratado envia um ciente de aprovação ao gerente, comunicando se aceita ou não a contratação para executar a atividade;

e) o contratado envia relatórios ao gerente, informando sobre o andamento da execução ou do término da atividade.

Inspirado no paradigma acima, um modelo para indução de AD foi proposto por Halmenschlager [HAL 2001], possuindo dois tipos de atributos (agentes):

a) *atributos-simples:* serão os nodos da árvore, ou seja, o agente-proponente que representa um atributo da base de dados;

b) *atributo-categórico:* serão as folhas da árvore, ou seja, o agente-gerente que representa o rótulo da árvore.

O atributo-categórico será definido pelo usuário entre todos os atributos da base de dados, enquanto que os demais atributos serão considerados atributos-simples. Similarmente aos agentes-proponentes de Smith [SMI 80], os atributos-simples competem entre si para realizar uma determinada atividade. Neste caso, a atividade pela qual eles competem é para ser a raiz da árvore ou o nodo da mesma em um determinada partição. O atributo-simples que vencer a competição será o nodo da AD, gerando as ligações de acordo com seus valores.

Para definir o atributo-simples vencedor, o atributo-categórico utiliza uma função de avaliação, que considerará vencedor o atributo que possuir um maior limiar, como, por exemplo, o maior ganho de informação. Dessa forma, um atributo com valor superior ficará sempre em um nível mais alto da árvore em relação aos atributos de menor valor.

Para cada ligação do atributo vencedor, os demais atributos-simples competem novamente entre si pelos nodos desta respectiva partição.

Quando todos os atributos-simples tiverem sido considerados no caminho desde a raiz, ou quando não houver nenhum outro atributo importante para uma determinada ramificação, deve ser incluído o atributo-categórico como a folha da árvore. A classe com maior probabilidade será considerada a classe desta respectiva folha. Dessa forma, os nodos ou folhas são adicionados sucessivamente até que o sistema apresente a árvore induzida com todas as suas folhas.

O modelo apresentado acima possibilita a implementação de agentes com o objetivo de aplicar Sistemas Multiagentes na Descoberta de Conhecimento. Porém, o principal problema que surge é a identificação de uma função de avaliação a ser usada para escolher o agente vencedor.

Assim, devido à complexidade de implementação dos agentes e o fato de poderem tornar o processo de indução um pouco mais lento, o protocolo de Redes de Contrato serviu apenas de inspiração para uma nova forma de induzir regras e árvores de decisão, sem ter sido, no entanto, efetuada a implementação física dos agentes, havendo uma preocupação maior com uma forma de indução, que não tivesse a necessidade de passadas recursivas nos dados, e com a função de avaliação utilizada, temas importantes deste trabalho.

4.2 Algoritmo

Com base no modelo descrito anteriormente, um algoritmo de indução de regras e árvores de decisão é proposto - com o objetivo de generalizar tanto regras quanto árvores de decisão a partir de qualquer base de dados.

O algoritmo proposto induz um classificador utilizando um atributo como rótulo, que é definido previamente entre os atributos fornecidos pelo conjunto de treinamento. O classificador final pode ser uma AD induzida, uma lista de regras derivadas desta árvore ou, ainda, ser uma lista de regras de decisão diretamente induzidas do conjunto de treinamento.

A AD é gerada utilizando a abordagem por largura, que adiciona todas as ligações do nodo antes de continuar a indução em um deles. A árvore final é mista, pois os nodos podem possuir um número variado de filhos, de acordo com os valores de seus atributos. É também univariada, pois cada nodo representa um único atributo do conjunto de treinamento.

A indução da AD inicia escolhendo um atributo que será a raiz da árvore. Para escolher este atributo, o algoritmo se baseia em uma função de avaliação, que utiliza uma tabela de ocorrências, juntamente com o critério de entropia. A função de avaliação está descrita com mais detalhes a seguir (seção 4.2.1).

Esta função de avaliação consiste no principal diferencial do modelo proposto. Ao contrário da maioria dos algoritmos de indução de árvores de decisão, que geram a árvore de decisão ao mesmo tempo que particionam os dados, o modelo proposto percorre os dados uma única vez, armazenando as ocorrências dos relacionamentos dois a dois de atributos em uma tabela. A tabela de ocorrências gerada é, então, utilizada para verificar o critério de entropia dos atributos para toda a árvore de decisão, sem a necessidade de particionamento ou de novos percursos no conjunto de dados.

Como a função de avaliação utiliza esta mesma tabela de ocorrências para gerar toda a árvore de decisão, este algoritmo não utiliza a abordagem de “dividir para conquistar”. O único percurso nos dados permite, além de uma economia de processamento, uma maior generalização e, conseqüentemente, uma menor especialização dos dados, evitando o *overfitting* dos mesmos.

Assim, a função de avaliação verifica o atributo que possui o maior ganho de informação entre os atributos não categóricos. Todos os atributos são testados e aquele que maximizar o critério da entropia, ou seja, que tiver o maior ganho de informação, será considerado o vencedor e adicionado como nodo. Também é adicionada uma ligação para cada valor deste atributo.

Para cada uma destas ligações é utilizada, novamente, a função de avaliação para escolher o próximo nodo de cada subárvore, desconsiderando os atributos que já estão presentes no caminho desde a raiz e evitando, assim, uma replicação de testes.

Quando uma determinada ligação do atributo vencedor possuir um ganho de informação maior que o limiar de parada definido previamente (o que é conhecido como pré-poda baseada na informação mútua), é adicionada a folha desta ligação com a classe mais provável, ou seja, mais freqüente na tabela de ocorrências. E assim são adicionados os nodos ou folhas, sucessivamente, até a AD estar completa.

Após, é feita uma simples redução da árvore de decisão: a AD é percorrida recursivamente, procurando por folhas de um mesmo nodo que tenham a mesma classe. Caso todas estas folhas possuam o mesmo rótulo, as folhas são removidas, e no lugar do nodo pai é adicionada uma única folha com a respectiva classe. Isto simplifica a AD em casos em que todos os testes de um nodo levem a uma mesma conclusão.

O núcleo do algoritmo proposto para a indução de uma AD está apresentado na Figura 4.2, considerando que a tabela de ocorrências já está gerada, podendo ser utilizada pela função de avaliação.

```
Função Árvore    (A: conjunto de atributos não categóricos,
                  C: atributo categórico,
                  δ: limiar de parada ) : AD
Início
        MelhorAtributo := FunçãoAvaliação (MaiorGanho, A);
        Acrescenta nodo MelhorAtributo;
        Para cada valor do MelhorAtributo faz
                Acrescenta ligação Ri;
                Se FunçãoAvaliação (MaiorGanho, MelhorAtributo(Ri)) ≥ δ então
                        Adiciona folha com a classe Ci mais provável em Ri;
                Senão
                        Chama Função Árvore para Ri (com A=A-MelhorAtributo);
                Fim se;
        Fim para;
 Fim.
```

FIGURA 4.2 – Algoritmo proposto para indução de AD

Pela Figura 4.2, observa-se que é informado ao algoritmo o limiar de parada δ, o atributo categórico C e os demais atributos não categóricos A. A função de avaliação escolhe dentre A o atributo que tiver o maior ganho de informação na tabela de ocorrências, definindo-o como *MelhorAtributo.* Será adicionada uma ligação R para cada valor do *MelhorAtributo* e, para cada um deles, é verificado se o resultado da função de avaliação é superior ao limiar δ. Em caso afirmativo, é adicionada a folha para esta ligação R com a classe mais freqüente de C. Em caso contrário, é chamada novamente a *Função Árvore* que irá analisar os atributos restantes de A (sem o *MelhorAtributo*), para encontrar novamente o atributo vencedor para ser o nodo da partição. E assim, sucessivamente, até que todas as partições possuam suas folhas.

Resumidamente, pode-se afirmar que o modelo induz árvores de decisão utilizando os seguintes métodos (apresentados na seção 3.1.1):

a) *eleição do melhor atributo:* realizada pela função de avaliação, que é implementada através da tabela de ocorrências e que escolhe o atributo com maior ganho de informação;

b) *regras de partição:* atribui uma ligação distinta a cada valor do atributo discreto;

c) *determinação da classe associada à folha:* efetuada pela atribuição da classe mais provável na partição;

d) *método de poda:* utiliza a técnica de pré-poda, com parada baseada na informação mútua;

A partir da AD gerada, é feita a derivação para regras, em que cada caminho da árvore é percorrido de forma *top-down*, traduzindo o mesmo em regras do tipo: *"Se <condição> então <classe>"*, em que a condição é formada pelo conjunto de testes dos nodos e ligações, e a classe é dada pela folha deste caminho. Depois que toda a AD é percorrida, tem-se uma lista de regras ordenadas que pode ser utilizada para a avaliação da árvore e para a classificação de novos exemplos.

Já a geração direta de regras é realizada induzindo as regras de cada classe separadamente, de acordo com os valores do atributo categórico. Assim, inicia-se induzindo todas as regras para uma única classe, seguido da indução das outras categorias, até que seja formada uma lista das regras de todas as classes. Para cada regra, é escolhida a melhor condição, verificando, para isso, todos os pares atributo-valor pela função de avaliação.

Similar à indução da AD, é selecionada a condição que tiver a maior probabilidade dada pela tabela de ocorrências, adicionando à lista de regras a estrutura: *"Se <condição>"*. Se esta condição for considerada uma *regra forte*, ou seja, possuir uma alta probabilidade na tabela de ocorrências para a classe – maior que o limiar de parada, a regra será finalizada pela adição da conclusão da mesma, ou seja, da estrutura *"então <classe>"*.

Em caso contrário, o algoritmo continuará percorrendo a tabela de ocorrências, buscando por novas condições que serão acrescentadas à condição atual, formando uma regra mais complexa e, conseqüentemente, com mais pares atributo-valor.

A condição de parada pela busca de mais condições para a regra atual também é feita pelo limiar de parada, como ocorre na indução de árvores. Ao aumentar o valor do limiar de parada, aumenta-se também o espaço de busca do algoritmo, obtendo-se regras com maior número de condições, pois no momento do teste da regra forte, o algoritmo necessita de uma probabilidade mais alta, e, não encontrando, continua a busca.

Cada vez que o algoritmo conclui uma nova regra, é verificado o número de casos suportados por ela. Quando o somatório do número de casos suportados pelas regras de uma classe for igual ao número total de casos desta classe, o algoritmo pára a indução para esta classe, iniciando a busca por regras da próxima classe, e assim, sucessivamente, até todas as classes possuírem suas regras.

Ao final da lista de regras, é verificado o grau de suporte e de confiança de cada regra, de forma a identificar as mais relevantes e preciosas, e também aquelas que não possuem um grau de suporte e de confiança mínimos, consideradas *regras fracas*. Essas regras fracas poderiam ser eliminadas da lista de regras, simplificando a mesma.

Também com o objetivo de simplificar a lista de regras, são eliminadas as regras duplas que porventura tenham sido geradas. As regras duplas são as regras que possuem as mesmas condições, mas em ordens diferentes.

O núcleo de indução de regras pelo algoritmo proposto está apresentado na Figura 4.3.

```
Função Regras    (A: conjunto de atributos não categóricos,
                  C: atributo categórico,
                  S: conjunto de treino ) : ListaRegras
Início
        ListaRegras:={};
        Para cada classe Ci em S fazer
                Casos:=0;
                Repetir
                        RegraClasse:"Se" ;
                        Repetir
                                MelhorCondição:=FunçãoAvaliação (MaiorGanho, A);
                                RegraClasse:=RegraClasse+" MelhorCondição";
                        Até FunçãoAvaliação(MaiorGanho,MelhorCondição)≥δ;
                        RegraClasse:=RegraClasse+"Então classe=Ci";
                        ListaRegras:=ListaRegras+RegraClasse;
                        Casos:=Casos+Suporte (RegraClasse);
                Até Casos = Suporte (Ci);
        Fim para;
Fim.
```

FIGURA 4.3 – Algoritmo proposto para indução de regras

Inicialmente, a *ListaRegras* está vazia. Para cada um dos valores do atributo categórico é verificada, pela função de avaliação, a *MelhorCondição*, que é o par atributo-valor que possui o maior ganho de informação na tabela de ocorrências. A regra da classe C_i recebe esta condição e procura por outras condições que também serão incluídas nesta regra, até que uma determinada condição possua a probabilidade mais alta ou igual que o limiar δ. Quando isto ocorrer, é atribuída à *RegraClasse* a classe mais freqüente. A lista de regras recebe esta regra formada e é verificado o número de casos suportados pela regra no conjunto de treinamento.

A principal diferença entre o algoritmo proposto e os demais algoritmos de indução de regras é a não exclusão dos exemplos cobertos por uma regra do conjunto de treinamento e também a única passada nos dados.

A última regra da lista de regras é a regra *default*, que apenas predita a classe mais freqüente nos dados. A lista final de regras é não ordenada, e pode ser utilizada para classificar novos exemplos, bem como para avaliar o classificador.

Embora a derivação origine uma lista ordenada de regras e a indução direta origine uma lista não ordenada, a classificação de um novo objeto é realizada da mesma forma para ambas as listas: a lista de regras (derivada da AD ou induzida) é percorrida, de forma a selecionar a primeira regra que possuir suas condições satisfeitas, atribuindo a respectiva classe ao novo objeto.

4.2.1 Função de avaliação

A função de avaliação utiliza uma tabela de ocorrências para identificar o atributo que possui o maior ganho de informação, definindo-o como o melhor atributo e, conseqüentemente, como o nodo da árvore, ou como a condição da regra, juntamente com o seu melhor valor.

O conjunto de dados será percorrido uma única vez, gerando essa tabela de ocorrências, que consiste na distribuição das ocorrências dos valores de cada atributo para cada classe, considerando o relacionamento dois a dois dos atributos. Para cada valor de atributo de cada instância do conjunto de dados, é somada uma ocorrência nas colunas dos outros valores de atributo de acordo com a classe na respectiva linha da tabela de ocorrências.

Exemplificando, a Tabela 4.1 mostra 14 instâncias com os fatores meteorológicos que permitem, ou não, o acontecimento de jogos de golfe, sendo o conjunto de treinamento utilizado pelo algoritmo de indução. O atributo categórico destes dados é o *Jogo* e o limiar de parada é definido em 0,80 (80%).

TABELA 4.1 – Conjunto de dados

Tempo	Vento	Umidade	Jogo
Ensolarado	Sim	Baixa	Joga
Ensolarado	Não	Baixa	Joga
Ensolarado	Sim	Alta	Não joga
Ensolarado	Não	Alta	Não joga
Ensolarado	Não	Alta	Não joga
Nublado	Sim	Alta	Joga
Nublado	Sim	Alta	Joga
Nublado	Não	Baixa	Joga
Nublado	Não	Alta	Joga
Chuvoso	Não	Alta	Joga
Chuvoso	Não	Alta	Joga
Chuvoso	Não	Alta	Joga
Chuvoso	Sim	Alta	Não joga
Chuvoso	Sim	Baixa	Não joga

Inicialmente, a tabela de ocorrências está zerada, e, após percorridos os dados, tem-se a tabela de ocorrências, como a apresentada pela Tabela 4.2.

TABELA 4.2 – Tabela de ocorrências

Atributo	**Tempo**						**Umidade**				**Vento**			
Valor	**Chuvoso**		**Ensolarado**		**Nublado**		**Baixa**		**Alta**		**Não**		**Sim**	
Classe	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**
T. Chuvoso	0	0	0	0	0	0	0	1	3	1	3	0	0	2
T. Ensolarado	0	0	0	0	0	0	2	0	0	3	1	2	1	1
T. Nublado	0	0	0	0	0	0	1	0	3	0	2	0	2	0
U. Baixa	0	1	2	0	1	0	0	0	0	0	2	0	1	1
U. Alta	3	1	0	3	3	0	0	0	0	0	4	2	1	2
V. Não	3	0	1	2	2	0	2	0	4	2	0	0	0	0
V. Sim	0	2	1	1	2	0	1	1	1	2	0	0	0	0

Para cada valor de atributo de cada instância da Tabela 4.1 é somada uma ocorrência nas colunas dos outros valores de atributo de acordo com a classe na respectiva linha do valor da Tabela 4.2. Ou seja, a primeira instância da Tabela 4.1 possui os valores: *Ensolarado, Sim, Baixa e Joga*, sendo somada uma ocorrência em cada uma das seguintes posições da Tabela 4.2:

a) linha *Ensolarado*, coluna *Umidade-Baixa-Joga*;

b) linha *Ensolarado*, coluna *Vento-Sim-Joga*;

c) linha *Baixa*, coluna *Tempo-Ensolarado-Joga*;

d) linha *Baixa*, coluna *Vento-Sim-Joga*;

e) linha *Sim*, coluna *Tempo-Ensolarado-Joga*;

f) linha *Sim*, coluna *Umidade-Baixa-Joga*.

Assim são somadas, sucessivamente, as ocorrências para todas as instâncias de dados. A complexidade desta função é *O(n.m!),* em que *n* é o número de instâncias do conjunto de treinamento e *m* é o número de atributos.

No caso da indução de AD, são somados os subtotais da tabela de ocorrências, ou seja, todas as suas linhas e colunas serão utilizadas para verificar o total de ocorrências de cada classe para cada valor de atributo, como apresentado na Tabela 4.3.

TABELA 4.3 – Subtotais de ocorrências

Valor	**Ocorrências**		**%**	
Classe	**Joga**	**Não**	**Joga**	**Não**
T. Chuvoso	6	4	60	40
T. Ensolarado	4	6	40	60
T. Nublado	8	0	100	0
U. Baixa	6	2	75	25
U. Alta	12	8	60	40
V. Não	12	4	75	25
V. Sim	6	6	50	50

Estes valores são, então, usados para verificar o atributo com o maior ganho de informação, que será considerado a raiz da árvore, calculando, para isso, o ganho de informação de cada um deles.

Por exemplo, utilizando o atributo *Tempo,* o valor esperado da informação, calculado pela soma ponderada da entropia relativa a cada um dos três subconjuntos, correspondentes aos valores de *Tempo* (*ensolarado, chuvoso, nublado*), seria:

T_1*: chuvoso* $\rightarrow$ $freq(C_1, T_1) = 6,\ freq(C_2, T_1) = 4,\ |T_1| = 10$

T_2*: ensolarado* $\rightarrow$ $freq(C_1, T_2) = 4,\ freq(C_2, T_{21}) = 6,\ |T_1| = 10$

T_3*: nublado* $\rightarrow$ $freq(C_1, T_3) = 8,\ freq(C_2, T_3) = 0,\ |T_1| = 8$

$$Info(Tempo) = 10/28 * (-4/10 * \log_2(4/10) - 6/10 * \log_2(6/10)) + 10/28 * (-6/10 * \log_2(6/10) - 4/10 * \log_2(4/10)) + 8/28 * (-8/10 * \log_2(8/10) - 0/10 * \log_2(0/10))$$

$$Info(Tempo) = 0{,}694$$

Analogamente, teria-se:

$$Info(Umidade) = 0{,}925$$

$$Info(Vento) = 0{,}892$$

Logo, será definido o *Tempo* como o melhor atributo e adicionado como raiz da árvore, por possuir o menor valor de entropia, que, conseqüentemente, maximiza o ganho de informação.

É adicionada também uma ligação para cada valor deste atributo vencedor. Dessa forma, são acrescentadas as ligações *Chuvoso*, *Ensolarado* e *Nublado* a partir da raiz. Para cada um destes valores, é verificado se o percentual de ocorrências em uma classe é igual ou superior ao limiar de parada, adicionando, neste caso, uma folha com esta classe. Assim, como a partição com o valor *Nublado,* pela Tabela 4.3, possui 100% de classificação para a classe *Joga*, o algoritmo adiciona diretamente uma folha com esta classe, como mostra a Figura 4.8.

Caso não seja adicionada uma folha para a ligação, serão verificados os próximos atributos importantes para a ramificação. Isto será feito considerando, pela tabela de ocorrências, apenas as colunas com o valor do atributo em questão, desconsiderando as linhas dos atributos que já apareceram no caminho desde a raiz.

Portanto, para as partições *Chuvoso* e *Ensolarado*, será verificado qual o próximo atributo a ser considerado para cada um deles, levando em conta apenas as colunas do valor do atributo em questão, a partir da mesma tabela de ocorrências. A área cinza da Tabela 4.4 representa os valores que serão utilizados para o cálculo da entropia e escolha do próximo nodo de *Chuvoso,* desconsiderando as linhas do próprio atributo (*Tempo*).

TABELA 4.4 – Tabela de ocorrências

Atributo	**Tempo**						**Umidade**				**Vento**			
Valor	**Chuvoso**		**Ensolarado**		**Nublado**		**Baixa**		**Alta**		**Não**		**Sim**	
Classe	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**	**Joga**	**Não**
T. Chuvoso	0	0	0	0	0	0	0	1	3	1	3	0	0	2
T. Ensolarado	0	0	0	0	0	0	2	0	0	3	1	2	1	1
T. Nublado	0	0	0	0	0	0	1	0	3	0	2	0	2	0
U. Baixa	0	1	2	0	1	0	0	0	0	0	1	0	2	1
U. Alta	3	1	0	3	3	0	0	0	0	0	5	2	1	2
V. Não	3	0	1	2	2	0	1	0	5	2	0	0	0	0
V. Sim	0	2	1	1	2	0	2	1	1	2	0	0	0	0

E assim, sucessivamente, será verificado qual o atributo que possui o maior ganho de informação para uma determinada ramificação, considerando apenas as colunas desta ramificação na tabela de ocorrências, até que seja encontrada a respectiva folha com percentual igual ou acima do limiar de parada.

Outra forma de parada é quando todos os atributos tiverem sido considerados no caminho desde a raiz, sendo incluída a classe mais freqüente como a folha da árvore. Dessa forma, os nodos ou folhas são adicionados sucessivamente até que seja gerada a árvore de decisão com todas as suas folhas.

Em contrapartida, quando forem induzidas regras ao invés de uma árvore, a tabela de ocorrências é gerada da mesma forma, porém a sua utilização é um pouco distinta. Para cada classe é verificada a melhor condição, ou seja, o par atributo-valor com maior probabilidade, considerando-o a condição da regra. Se esta probabilidade for igual ou superior ao limiar de parada, será adicionada a conclusão da classe. Para o exemplo anteriormente usado, a regra *"Se Tempo=Nublado então Joga"*, seria induzida desta forma, por possuir, conforme a Tabela 4.3, a maior probabilidade da classe *Joga*.

Em caso contrário, é procurado um novo par de atributo-valor para compor a condição da regra, juntamente com o par anterior. Para isso, é verificada, na tabela de ocorrências, a coluna do par anterior que possua a maior probabilidade. Caso ocorram empates na probabilidade, vencerá a condição que possuir o maior número de instâncias.

E assim ocorre, sucessivamente, até ser encontrada uma conclusão para a regra. Se não for encontrada uma boa conclusão, ou seja, uma condição cujo resultado da função de avaliação seja maior ou igual que o limiar de parada, a condição será abandonada, procurando-se por outra, que possua uma melhor classificação.

Todos os atributos são testados como condição para uma determinada regra. Quando todos os pares de atributo-valor já tiverem sido testados, ou todas as instâncias da regra já tiverem sido cobertas, parte-se para a indução das regras da próxima classe.

Após todo o processamento, a árvore e as regras geradas para este exemplo são as mesmas apresentadas nas Figuras 4.8 e 4.9, respectivamente.

O objetivo da implementação desta tabela de ocorrências é percorrer os dados uma única vez, otimizando a performance da construção do classificador e o acesso às informações, ao mesmo tempo que evita o *overfitting* dos dados, dando uma visão geral dos dados e possibilitando a realização do cálculo da entropia – ou verificação da probabilidade, sem a necessidade de consultas e/ou partições dos dados. Entretanto, este método pode generalizar demais os dados, sendo mais adequado à indução de árvores e regras que possuam poucos níveis ou baixa dimensionalidade dos dados.

4.3 Protótipo

A partir do algoritmo descrito acima, um protótipo, denominado *Midas – Minerando Dados*, foi implementado utilizando a linguagem Borland Delphi, por ser esta uma linguagem orientada a objetos, por possuir muitas facilidades de utilização, um ambiente rico e um compilador rápido, além de ter um bom suporte ao acesso a bases de dados [CAN 2000].

O Midas pode ser utilizado em microcomputadores com ambiente Windows 95/98/2000 e NT. A performance do sistema é proporcional à memória RAM e processador disponíveis.

A adoção destas definições deu-se pela disponibilidade de equipamento, disponibilidade de ferramentas de desenvolvimento (sistema operacional e linguagem) e padronização dentro do grupo de pesquisa, além de maior conhecimento da plataforma. Um esquema da funcionalidade do protótipo é apresentado no Anexo 2, com descrições dos componentes do mesmo.

A tela inicial do Midas (Figura 4.4), disponibiliza um *menu* na parte horizontal superior e uma *barra de ferramentas* (botões) na parte vertical esquerda, que ativam as funcionalidades do sistema.

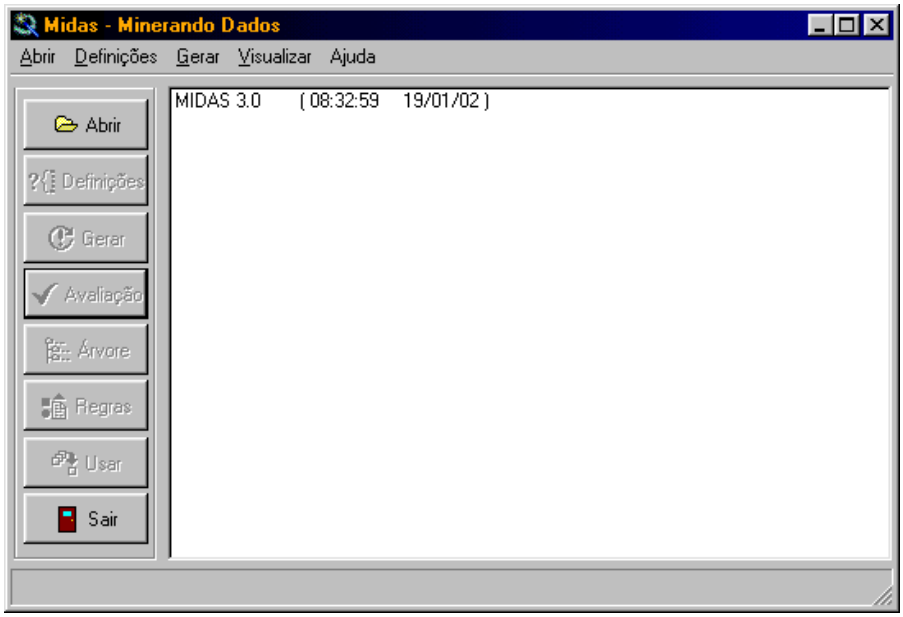

FIGURA 4.4 – Interface principal do Midas

A *barra de status,* área horizontal que localiza-se na parte inferior da tela principal do Midas, fornece informações sobre a evolução do processo de indução, além de exibir informações sobre os botões disponíveis, buscando resolver eventuais dúvidas do usuário.

Também são gerados automaticamente arquivos de *log* que armazenam as informações de cada etapa de indução do classificador. Estes arquivos são armazenados no mesmo diretório que contém o arquivo de dados.

Primeiramente, só é possível selecionar a opção *Abrir*, que permite escolher e abrir uma tabela de dados, a partir da qual a mineração será realizada, estando os demais botões e menus desabilitados.

O conjunto de dados usado na mineração deve estar no formato "*DBF*" ou "*DB*", para que possa ser aberto pelo Midas. Este formato é facilmente obtido pelo software Microsoft Excel, caso os dados tenham sido exportados em um outro formato. A Figura 4.4 apresenta a caixa de diálogo para escolha e abertura do arquivo pelo usuário.

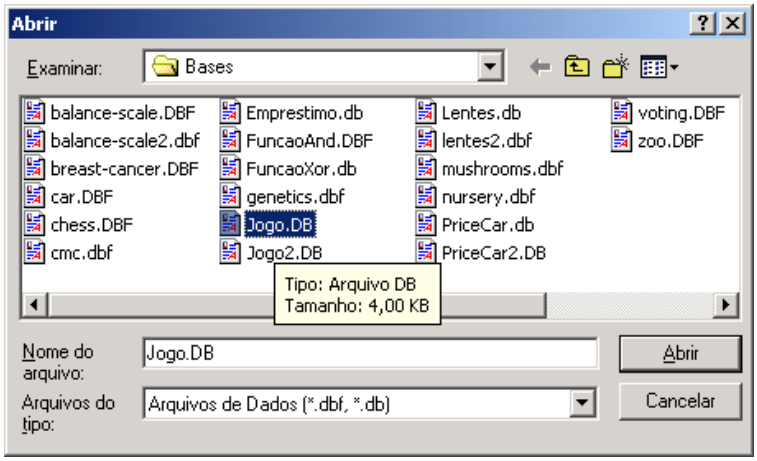

FIGURA 4.5 – Caixa de diálogo para abertura de arquivo

Caso o arquivo seja aberto corretamente, além de uma mensagem de sucesso, será informado na janela principal o número de registros lidos. Este arquivo aberto consistirá no conjunto de treinamento do algoritmo de indução. O limite máximo no número de atributos e de valores de atributos do conjunto de treinamento é conforme a memória principal disponível, tendo em vista que estes números determinarão o tamanho da tabela de ocorrências. Já o número de registros do conjunto de treinamento influenciará no tempo de geração da tabela de ocorrências, mas não na memória ocupada.

Entretanto, o protótipo não realiza, ainda, tratamento para valores desconhecidos ou quantitativos, adicionando sempre uma ligação ou condição para cada valor do atributo, o que é aceitável para domínios discretos, mas não para contínuos. Dessa forma, todos os atributos contínuos do conjunto de treinamento deverão ser, primeiramente, pré-processados.

A seguir, o botão *Definições* ficará habilitado para que o usuário escolha o atributo categórico e outras opções de indução. Ao clicar neste botão, uma caixa de diálogo é aberta (Figura 4.6) e todos os atributos do conjunto de treinamento são exibidos em uma lista *drop-down*. O atributo selecionado será considerado o atributo categórico e, conseqüentemente, o rótulo dos novos objetos.

FIGURA 4.6 – Caixa de diálogo para definições de indução

Algumas configurações de funcionamento do algoritmo de indução são definidas nesta caixa de diálogo. Se a opção *Regras* estiver selecionada, em vez de induzir uma AD, o sistema induzirá uma lista de regras de decisão a partir dos dados. Neste caso, deve-se definir também o grau mínimo de suporte e confiança desejados para cada regra, ou serão adotados os valores *default*: 15% de suporte e 70% de confiança. O grau de confiança também é utilizado como o limiar de parada na indução da árvore.

O Midas também permite a definição de um *conjunto de teste*, criado a partir do conjunto de exemplos, de acordo com o percentual indicado pelo usuário. Os exemplos do conjunto de teste são escolhidos randomicamente e não farão parte do conjunto de treinamento. O sistema induz o classificador a partir do conjunto de treinamento restante e o conjunto de teste é utilizado apenas para a avaliação do classificador, realizando, assim, a estimativa por utilização de um conjunto independente.

O protótipo disponibiliza, ainda, o método de *validação cruzada* para a indução e estimativa de erro do classificador final, sendo necessário informar o número de partições e iterações que deverão ocorrer. Também é possível definir o tamanho máximo de pares atributo-valor na condição de cada regra induzida, ou dos níveis da árvore, através da opção *níveis*.

O botão *Default* da caixa de diálogo *Definições* atualiza as opções de acordo com as configurações iniciais, enquanto o botão *Ok* confirma as escolhas definidas, só estando disponível quando for selecionado algum atributo como atributo categórico.

Após essas definições, o sistema disponibiliza a opção *Gerar*, que, ao ser clicada, criará a tabela de ocorrências e, a partir dela, induzirá a AD ou a lista de regras, conforme descrito na seção 4.2. Os resultados preliminares, como tamanho do classificador, tempo de aprendizado e taxa de erro, são apresentados na janela principal, como mostra a Figura 4.7.

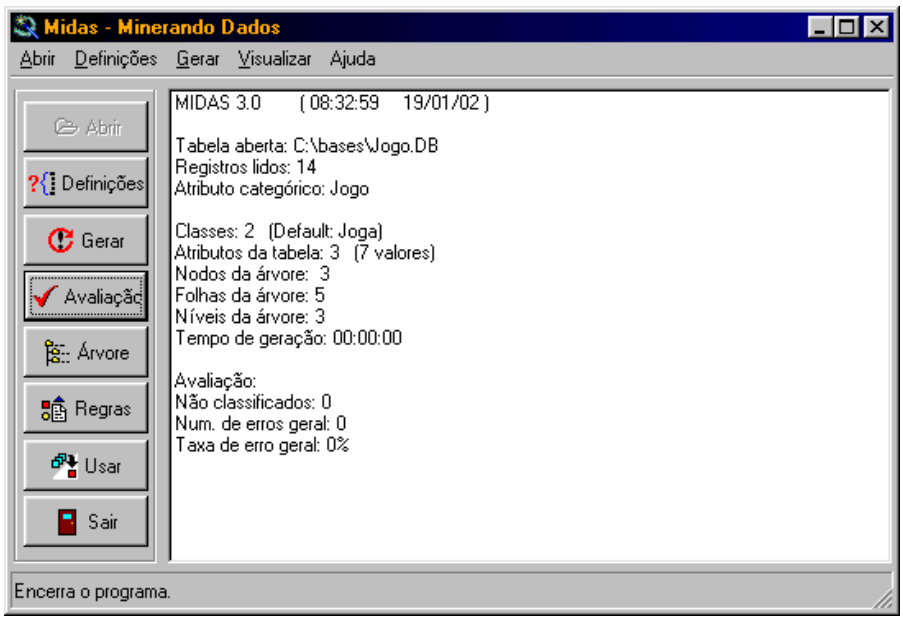

FIGURA 4.7 – Resultados preliminares do classificador induzido

A partir da geração do classificador, todos os outros botões estão disponíveis, permitindo a exibição do classificador em forma de uma estrutura hierárquica ou em forma de lista de regras, bem como a avaliação e a utilização do mesmo, pelos respectivos botões *Árvore, Regras, Avaliação* e *Usar*.

Assim, a demonstração e a análise dos resultados da mineração podem ser feitas de duas formas. Pode-se visualizar a árvore de decisão induzida, ao clicar o botão *Árvore,* que apresenta a estrutura hierárquica indentada da árvore em forma gráfica, e possibilitando, com um simples clique, a expansão dos nodos da árvore. A Figura 4.8 apresenta uma AD induzida em que todos os caminhos da árvore estão expandidos até a sua respectiva folha.

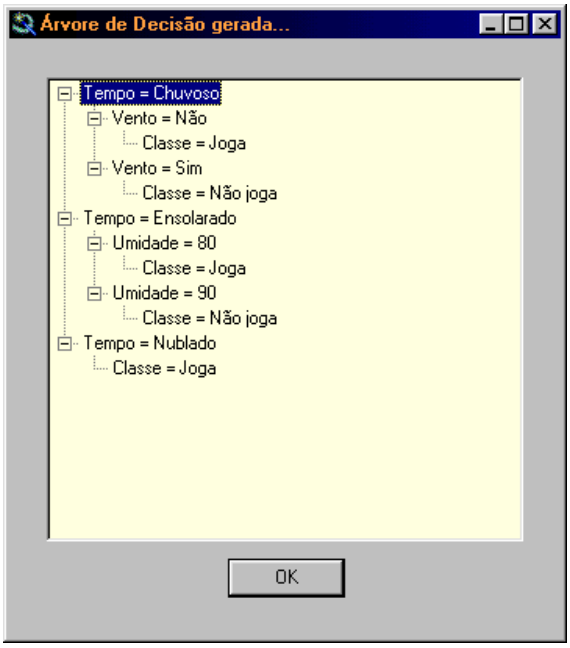

FIGURA 4.8 – AD induzida

Também pode-se visualizar o resultado da mineração através de uma lista de regras. Essa lista pode ter sido derivada da estrutura de árvore ou pode ter sido induzida diretamente dos dados. A Figura 4.9 apresenta a lista de regras equivalente à Figura 4.8.

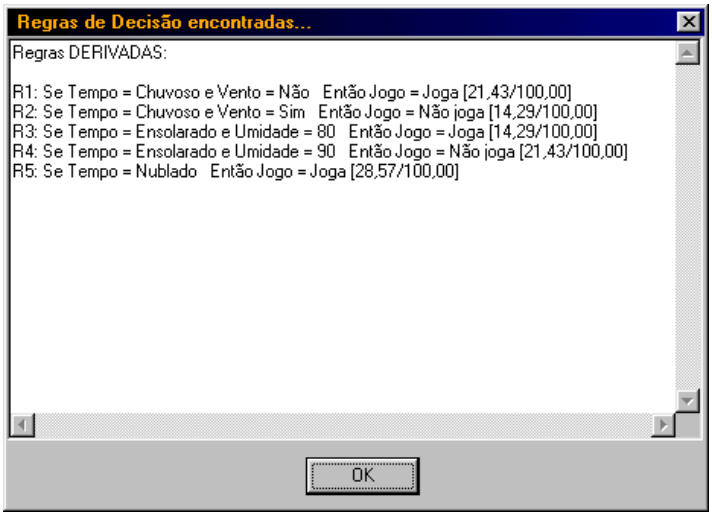

FIGURA 4.9 – Lista de regras

A avaliação do classificador é realizada, por *default*, pela estimativa por ressubstituição, quando todo o conjunto de treinamento é passado novamente pelo classificador e é acrescentado o número e a taxa de erro na janela principal (Figura 4.7). Como citado anteriormente, também pode ser feita a avaliação do classificador utilizando um conjunto independente de dados ou pela validação cruzada, opções escolhidas em *Definições*.

A partir da indução da AD ou das regras, também é possível utilizar o classificador final, com o objetivo de classificar exemplos ainda não rotulados. Isto é possível pelo botão *Usar,* que abre uma tela em que é apresentado cada atributo com seus respectivos valores em uma lista *drop-down*, devendo o usuário selecionar o valor de cada atributo, de forma que estes valores correspondam ao novo objeto.

A partir destes valores indicados pelo usuário, é percorrida a lista de regras para encontrar a primeira regra que possua todas as suas condições satisfeitas pelos valores do novo objeto. Neste momento, é atribuída a classe desta regra para este novo objeto. No exemplo da Figura 4.10, a classe *Joga* proveniente da regra *R1* (Figura 4.9), foi atribuída ao novo objeto, cujos valores estão listados nesta mesma figura. Caso nenhuma regra tenha sido selecionada, prevalece a classe *default*, que é a classe com maior freqüência no conjunto de treinamento.

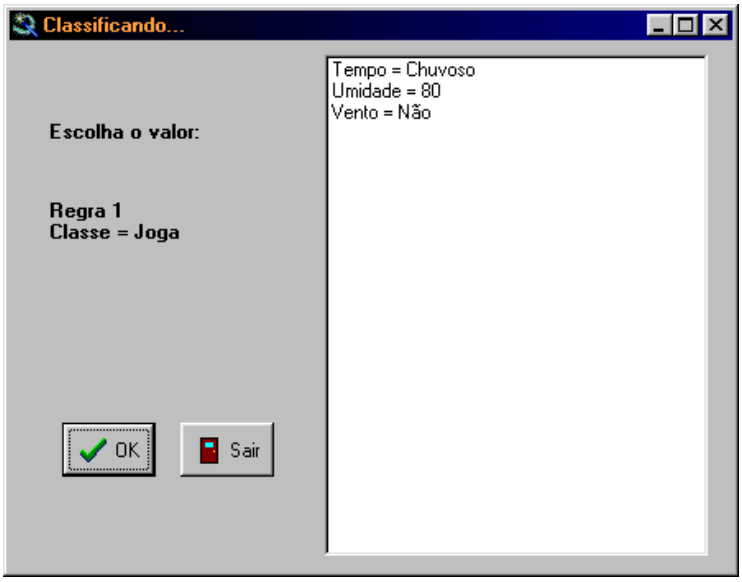

FIGURA 4.10 – Usando o classificador

Assim, devem ser informados os valores dos atributos de cada novo objeto, não sendo possível a utilização de um arquivo de classificação – análogo ao conjunto de treinamento, mas sem os rótulos – de forma a rotular os vários exemplos deste conjunto automaticamente.

4.4 Testes realizados

Primeiramente, foram realizados testes com pequenos conjuntos de dados, como o da Tabela 4.1, para validar o funcionamento do modelo proposto. Os testes foram realizados, geralmente, em uma máquina com sistema operacional Windows 98, processador K6 233 Mhz, 64 Mb de memória RAM e 3.1 Gb de disco rígido.

Os mesmos dados foram testados tanto no Midas quanto na ferramenta See5[4] ([DAT 2001] e [SEE 2001]), que possui em seu núcleo o algoritmo C4.5 (descrito na seção 3.1.3). Esta escolha foi feita pela disponibilidade do software, pela sua importância na evolução deste ramo da ciência e também por ser o mais conhecido na comunidade de Aprendizado de Máquina.

Também foram realizados alguns testes com dados do UCI Machine Learning Repository [UCI 2001], que foram validados no See5 e no Midas. As características dos conjuntos de dados utilizados do repositório estão apresentadas na Tabela 4.5. Estes dados foram escolhidos por possuírem número de instâncias, atributos e classes variados, por possuírem atributos discretos e também por serem bases de dados de aprendizagem automática, de uso comum neste tipo de teste, proporcionando testes comparáveis com as diversas experiências que surgem freqüentemente na literatura da especialidade, além de serem de domínio público. Mais detalhes destas bases de dados podem ser obtidas no Anexo 1.

TABELA 4.5 – Dados usados nos testes

Base	Instâncias	Atributos	Valores	Classes
Balance	625	4	20	3
Breast	699	9	90	2
Car	1.728	6	21	4
Chess	28.056	6	40	17
Genetics	3.190	60	287	3
Mushrooms	8.124	22	117	2
Nursery	12.960	8	27	5
Voting	435	16	48	2
Zoo	101	16	36	7

A seguir, são apresentados os resultados obtidos pelos algoritmos com as bases de dados em cada sistema, que foram aplicadas tanto para indução de regras quanto de árvores. A comparação entre os algoritmos é centrada na simplicidade do modelo, no tempo de processamento e no desempenho de classificação, utilizando todos os registros da base como conjunto de treinamento. Os algoritmos foram utilizados com suas opções de execução *default*.

[4] O See5, para plataforma Windows 95/98/2000 e NT, e o C5.0, para plataforma Unix, são ferramentas de Mineração de Dados que realizam a classificação implementando o algoritmo C4.5.

Apesar do algoritmo C4.5 ser específico para indução de árvores de decisão, a ferramenta See5 também realiza a indução de regras, a partir de uma variação deste mesmo algoritmo, conhecida como C4.5rules, em que as regras são derivadas da árvore e, após, é feita uma generalização das mesmas pelo algoritmo, desconsiderando as condições e/ou regras supérfluas.

A Tabela 4.6 apresenta um resumo dos resultados encontrados no See5 e no Midas, em relação ao tamanho do classificador: em folhas, no caso da indução de árvores, e em número de regras, no caso da indução de regras, para algumas bases de dados do repositório UCI.

TABELA 4.6 – Tamanho dos classificadores

Classificador	**Árvore**		**Regras**	
Base	**SEE5**	**MIDAS**	**SEE5**	**MIDAS**
Balance	**33**	49	15	**8**
Breast	16	**10**	6	**2**
Car	131	**3**	77	**2**
Chess	**4.021**	6.019	2.450	**47**
Genetics	124	**5**	**45**	53
Mushrooms	19	**9**	9	**2**
Nursery	**359**	935	155	**13**
Voting	6	**3**	4	**2**
Zoo	**9**	28	**9**	16
Média	**524**	**784**	**308**	**16**

Analisando a Tabela 4.6, percebe-se que a indução de regras de decisão pelo Midas gera um número quase sempre menor de regras do que o número de folhas induzindo uma árvore de decisão, gerada pelo próprio Midas. Isto deve-se ao fato de que as regras não são estruturas hierárquicas, não necessitando da indução de algumas regras, como ocorre nas árvores. Entretanto, a base Genetics teve o número de regras superior ao número de folhas, por ser a base que possui uma grande dimensionalidade, o que pode ocasionar um aumento do espaço de busca na indução de regras.

As bases Breast, Car, Mushrooms e Voting tiveram tanto o número de regras quanto o de folhas mais baixo no Midas. Já as bases Balance, Chess e Nursery tiveram apenas o número de regras inferior no Midas. Entretanto, a base Zoo teve um número mais elevado de regras e folhas no Midas, enquanto que a base Genetics teve apenas o número de regras maior no modelo proposto. Isto mostra que, de um modo geral, o protótipo generaliza melhor as bases de dados, sendo uma maior simplicidade do classificador obtida na indução de regras.

A média aritmética obtida pelo Midas foi de 16 regras, contra 308 regras do See5. Porém, a base Chess deturpou um pouco os resultados, por possuir um valor muito acima da média. Assim, ao ignorar a mesma para efetuar novamente a média das regras induzidas, encontra-se uma média de 40 regras induzidas pelo See5 e de 12 induzidas pelo Midas, mostrando que, mesmo assim, o Midas gera modelos mais simples, com uma redução relativa de 70%.

Já o tempo de aprendizado em cada base de dados é apresentada na Tabela 4.7, em segundos.

TABELA 4.7 – Tempo de aprendizado

Classificador	**Árvore**		**Regras**	
Base	**SEE5**	**MIDAS**	**SEE5**	**MIDAS**
Balance	0	0	0	0
Breast	0	0	0	1
Car	0	1	0	1
Chess	**5**	116	298	**24**
Genetics	**3**	83	**3**	91
Mushrooms	**1**	34	**1**	34
Nursery	**12**	35	**12**	32
Voting	0	0	0	0
Zoo	0	2	0	1
Média	**2**	**34**	**35**	**20**

Em relação ao tempo, observa-se que no Midas o tempo de indução para regras foi menor do que para a indução da árvore de decisão.

Como o tempo de aprendizado no See5 é mais influenciado pelo número de registros da base ($O(m.n^2)$), as bases Chess, Nursery, Mushrooms e Genetics foram as que necessitaram de maior tempo de indução. Já o Midas possui o tempo de aprendizado influenciado pelo número de atributos ($O(n.m!)$). Por este motivo, a base Genetics foi a mais lenta, seguida das bases Chess, Nursery e Mushrooms.

É interessante observar que a base Zoo, na indução no Midas, teve um tempo de indução superior a zero segundo, mesmo sendo uma base que possui apenas 101 registros. Isso ocorreu pois esta base possui uma dimensionalidade relativamente alta (10), bem como um número de classes elevado (7).

Apesar do Midas não ter tido um melhor tempo de indução de regras em seis bases, a principal diferença e ganho do protótipo se relaciona às bases com maior número de registros, como pode-se observar na base Chess, em que o See5 levou 298 segundos para a indução das regras, contra os 24 segundos que foram necessários pelo Midas.

A Tabela 4.8 mostra a taxa de erro encontrada, utilizando a estimativa por ressusbstituição, em cada uma das bases de dados, em percentual. É importante salientar que na indução de regras, nas duas ferramentas, foi utilizada a classe *default* como a última regra presente na lista de regras.

TABELA 4.8 – Taxa de erro dos classificadores

Classificador	Árvore		Regras	
Base	**SEE5**	**MIDAS**	**SEE5**	**MIDAS**
Balance	25,00	45,12	22,10	15,68
Breast	1,60	7,30	3,10	24,61
Car	3,70	29,98	3,90	29,98
Chess	3,50	75,17	29,50	83,44
Genetics	3,70	48,12	4,00	37,81
Mushrooms	0	1,48	0,24	21,61
Nursery	1,90	19,49	1,40	33,52
Voting	2,80	4,37	2,80	4,37
Zoo	1,00	21,78	1,00	8,91
Média	**4,80**	**28,09**	**7,56**	**28,89**

Pela Tabela 4.8, observa-se que o Midas está com a média da taxa de erro aparente muito elevada, tanto na indução de árvores quanto de regras. A partir disso, pode-se concluir que o algoritmo proposto está generalizando muito o conjunto de treinamento, não aprendendo detalhadamente os padrões presentes nos dados, ou seja, o Midas está gerando poucas regras, mas levando a uma taxa erro aparente mais elevada que o See5.

Por outro lado, a Tabela 4.9 apresenta a média de suporte e de confiança das regras diretamente induzidas pelo Midas, em percentual.

TABELA 4.9 – Média de suporte e confiança

Base	Suporte	Confiança
Balance	14,48	74,40
Breast	30,32	99,72
Car	33,33	100,00
Chess	0,50	57,29
Genetics	1,00	99,53
Mushrooms	16,84	100,00
Nursery	6,98	85,57
Voting	48,74	97,28
Zoo	6,62	95,64
Média	**18,26**	**89,94**

Essas médias mostram que, apesar do baixo número de regras gerado e da alta taxa de erro aparente, as regras que são induzidas possuem um grau de certeza bastante elevado. Nas bases Car e Mushrooms, por exemplo, o grau de confiança atingiu 100%, concluindo-se que as regras induzidas (apenas duas) para cada base possuem uma confiança máxima, ou seja, todos os casos cobertos pela regra também possuem a classe definida pela regra.

Já a média do grau de suporte na base Car é de 33,33%. Como existem duas regras induzidas, o grau de suporte total é de 66,67%. Assim, como o grau de suporte não atingiu 100% dos casos, percebe-se que algumas instâncias da base (cerca de 33,33%), não foram cobertas por nenhuma regra, o que acaba gerando a alta taxa de erro aparente (Tabela 4.8).

Por outro lado, essas instâncias não devem ser consideradas erro e sim não classificações, uma vez que não foi encontrada nenhuma regra para a instância, e que, conseqüentemente, ela não foi classificada. Isto pode ser possível e perfeitamente adequado em casos em que o objetivo do classificador induzido é criar uma visão genérica dos dados, de forma a obter algum novo conhecimento, mas não para casos em que o seu objetivo é, estritamente, o de classificar novos casos.

Assim, as taxas de erro para a indução de regras foram geradas novamente, não considerando-se como erro os exemplos não classificados e obtendo-se os resultados apresentados na Tabela 4.10, em percentual.

TABELA 4.10 – Taxa de erro (sem os não classificados)

Base	**Taxa de erro**
Balance	8,00
Breast	0,14
Car	0
Chess	0,13
Genetics	2,95
Mushrooms	0
Nursery	2,41
Voting	3,68
Zoo	0
Base	**1,92**

Estes resultados mostram que, dessa forma, a taxa de erro diminui, e muito, na indução de regras pelo Midas, pois a maioria das instâncias que são cobertas pelas regras estão corretas. A taxa de erro da indução de árvores, entretanto, não se altera, pois como a AD possui uma estrutura hierárquica, cada instância deve, obrigatoriamente, seguir algum caminho da AD e, conseqüentemente, ser classificada.

Outro teste realizado foi executar dez vezes a validação cruzada para estimar a média e o desvio padrão da taxa de erro destas bases de dados, sendo divididos os dados em dez partições em cada execução. Assim, foram induzidos (10*10) classificadores para obter a taxa de erro. Os resultados podem ser verificados na Tabela 4.11, na qual são apresentados a média e o desvio padrão da taxa de erro (conforme seção 3.4), em percentual.

TABELA 4.11 – Taxa de erro (média e desvio padrão)

Classificador	**Árvore**		**Regras**	
Base	**SEE5**	**MIDAS**	**SEE5**	**MIDAS**
Balance	38,10±1,00	32,94±2,51	25,90±0,80	8,07±1,09
Breast	6,30±0,80	7,98±0,98	5,10±0,90	0,24±0,18
Car	7,50±0,50	29,98±1,13	5,70±0,50	0±0,00
Chess	42,70±0,20	76,20±0,23	39,50±0,30	0,31±0,89
Genetics	5,90±0,50	43,60±6,30	6,30±0,60	3,14±0,73
Mushrooms	0±0,00	1,48±0,14	0,20±0,00	0±0,00
Nursery	2,80±0,10	19,07±0,62	2,00±0,10	2,71±0,88
Voting	3,20±1,00	4,51±1,12	6,90±1,50	3,68±0,94
Zoo	6,90±1,50	28,54±8,33	3,20±1,00	0,36±0,30
Média	**12,60±0,62**	**27,14±2,37**	**10,53±0,63**	**2,05±0,56**

Pela Tabela 4.11, percebe-se que o Midas possui uma boa taxa de precisão nos exemplos cobertos pelas regras, pois a média da taxa de erro ficou em 2,05% de erro, desconsiderando como erro as instâncias não cobertas por nenhuma regra.

Já o desvio padrão pode ser visto como uma imagem da robustez do algoritmo, ou seja, se os erros provenientes de hipóteses induzidas utilizando diferentes conjuntos de treinamento são muito diferentes de um teste para outro, o indutor não é robusto a mudanças no conjunto de treinamento. Isto significa que, o Midas é robusto apenas para indução de regras, por possuir, nesta, uma média de desvio padrão de 0,56%.

Entretanto, a taxa de erro do Midas, na Tabela 4.11, exclui os exemplos não classificados para regras, enquanto que no See5 não foi possível desconsiderar estas instâncias no cálculo da taxa de erro, prejudicando uma comparação e avaliação mais detalhada e precisa.

Uma simples comparação da indução de regras foi realizada através da diferença absoluta entre os algoritmos (conforme seção 3.4). A Tabela 4.12 apresenta os valores da diferença absoluta da média e desvio padrão da taxa de erro de cada base, sendo considerado o C4.5 como o algoritmo padrão.

TABELA 4.12 – Diferença absoluta dos algoritmos

Base	**da**
Balance	18,65
Breast	7,49
Car	16,12
Chess	59,01
Genetics	4,73
Mushrooms	0
Nursery	-1,13
Voting	2,57
Zoo	3,85
Média	**12,37**

Na diferença absoluta, qualquer valor maior que dois (positivo ou negativo) indica que o resultado é significativo, com nível de confiança de 95%. Quando o valor encontra-se acima de zero, significa que o algoritmo proposto (Midas) supera o padrão (C4.5); se o valor encontra-se abaixo, então o algoritmo padrão supera o proposto.

Pela análise da tabela 4.12, pode-se verificar que o modelo proposto para indução de regras supera o C4.5 com confiança acima de 95% na maioria das bases testadas. Nas bases Mushrooms e Nursery, o algoritmo padrão, proveniente do See5, tem, respectivamente, um resultado igual e um melhor, superando, neste último, o modelo proposto, mas não significativamente.

4.5 Avaliação preliminar

Nos testes iniciais com o protótipo Midas, verificou-se que as árvores resultantes foram maiores e mais complexas do que as geradas pelo See5 com os mesmos dados. Isso ocorreu porque as bases de treinamento possuíam alguns atributos contínuos e o protótipo ainda não é capaz de lidar com a informação quantitativa, construindo, portanto, ligações para cada valor dos atributos e gerando árvores grandes e não otimizadas, ao contrário do See5 que discretiza os atributos.

Assim, optou-se por utilizar apenas conjuntos de dados que tivessem apenas atributos discretos ou que pudessem ser pré-processados, transformando-os do domínio contínuo para discreto, até que a ferramenta disponha de algum tipo de tratamento para atributos contínuos.

Pelos testes, observou-se que se obteve maior simplicidade, menor tempo de processamento e melhor taxa de erro, na indução de regras pelo Midas do que na indução de árvores pelo próprio Midas.

Como deseja-se obter regras simples e evitar o *overfitting* dos dados, pode-se afirmar que o Midas generaliza os dados muito bem, levando a descoberta de poucas e simples regras, o que torna a interpretação dos resultados bastante fácil.

Entretanto, esta grande generalização dos dados tem levado a uma alta taxa de erro. Porém, se for considerado o grau de suporte e de confiança das regras – que são as medidas mais importantes em regras do que a própria taxa de erro – verifica-se que esta alta taxa de erro está relacionada, em grande parte, com exemplos que não foram classificados, e não com exemplos que foram classificados erroneamente.

Assim, desconsiderando-se os exemplos não classificados, obtêm-se uma taxa de erro, na indução de regras, inferior ao See5, obtendo-se dessa forma um modelo adequado para a generalização dos dados, mas não para realizar futuras classificações.

Por outro lado, como a ordem de complexidade da geração da tabela de ocorrências é exponencial, e a complexidade da indução de árvores e da indução de regras é polinomial (não sendo exponenciais em relação ao número de atributos), as duas últimas são, dessa forma, desconsideradas na complexidade total do modelo proposto.

Ao comparar-se a complexidade do C4.5 – $O(m.n^2)$, e do Midas – $O(m!.n)$, verifica-se que o primeiro possui um custo computacional influenciado, principalmente, pelo número de registros, enquanto que no Midas o custo computacional é influenciado, principalmente, pelo número de atributos. Dessa forma, para um caso em que tem-se dez atributos e um milhão de registros, o custo computacional no C4.5 é muito superior ao Midas. Já se o número de atributos for cem e o número de registros for dez, o custo do C4.5 é bastante inferior.

Assim, conclui-se que o Midas é mais adequado para grandes conjuntos de dados que contenham baixa dimensionalidade, sendo perfeitamente aplicado em dados reais, que possuem, normalmente, poucos atributos e milhares de registros.

4.6 Aplicação sobre dados reais

Com o objetivo de verificar o comportamento do modelo proposto diante da generalização de bases de dados reais, duas bases de dados da área da saúde foram aplicadas no Midas. A facilidade de acesso a estes dados, a possibilidade de uma maior aproximação com a realidade e o fato de ser um domínio rico em informações foram motivos que levaram à escolha destas bases de dados.

A seguir, é descrita cada uma dessas bases, bem como alguns resultados obtidos dessa aplicação.

4.6.1 Dados de Autorização de Internação Hospitalar

As informações extraídas através da Descoberta de Conhecimento estão sendo utilizadas em diferentes meios, prevendo informações estratégicas e críticas em áreas como análise de risco, marketing direto, detecção de fraudes, qualidade dos produtos, apoio à decisão, segmentação e análise de mercado. Como os dados podem ser encontrados facilmente e os resultados obtidos mostram-se promissores para qualquer domínio de aplicação, as possibilidades de aplicação são inúmeras.

Em vista da ampla aplicabilidade e da grande importância da distribuição correta e equilibrada de verbas na Saúde Pública, a aplicação da Mineração de Dados em dados deste domínio visa promover a utilização de ferramentas de Tecnologia da Informação, fundamentais para o desenvolvimento científico, social e econômico do Estado. A idéia é otimizar os recursos financeiros públicos, melhorando as condições do Sistema Único de Saúde (SUS) no Estado, a partir da extração do conhecimento dos dados e da tomada de decisões gerenciais.

A Secretaria de Saúde do Estado do Rio Grande do Sul (SES) é responsável pela administração das informações do SUS, possuindo controle sobre várias bases de dados de saúde, como as de internações hospitalares, mortalidade, vacinação, laboratoriais, procedimentos complexos, entre outras.

Os dados foram obtidos junto à SES, através do projeto "Desenvolvimento de Metodologia para Extração de Conhecimento de Bases de Dados da Saúde do Estado para Avaliação e Planejamento", realizado como colaboração entre pesquisadores da UFRGS e a SES.

Primeiramente, foi escolhida para mineração a base de dados que contém AIHs (Autorizações de Internação Hospitalar). As AIHs registram as internações, procedimentos e diagnósticos realizados em hospitais dos municípios do Rio Grande do Sul (sem gestão plena) através do Sistema Único de Saúde. As internações mensais são repassadas à Secretaria para que estas instituições recebam seus honorários pelos serviços prestados ao SUS.

A estrutura interna de uma AIH é padronizada para qualquer instituição ligada ao SUS. Cada internação corresponde a um registro de AIH, contendo dados de identificação do paciente, do hospital e do médico responsável, os respectivos diagnósticos, procedimentos solicitados, procedimentos realizados e procedimentos especiais, além dos custos e materiais necessários.

Após definida e obtida a base de dados das AIHs, foram selecionados dados de dois meses do ano 2000, totalizando 96.145 registros, com os quais a Descoberta de Conhecimento seria realizada. Com a ajuda de especialistas da Secretaria da Saúde, foram selecionados os atributos e/ou procedimentos mais interessantes e importantes para a descoberta.

A estrutura básica deste conjunto de dados, que possui oito atributos e foi utilizada para a mineração, está apresentada na Tabela 4.13.

TABELA 4.13 – Estrutura básica da AIH

Campo	**Descrição**	**Valores**
Sexo	Sexo do paciente	F, M
Idade	Idade do paciente	0: <10, 10: 10-19, 20: 20-29, 30: 30-39, 40: 40-49, 50: 50-59, 60: 60-69, 70: 70-79, 80: >80
Diag	Diagnóstico por grupo principal (conforme CID)	
Proc	Procedimento especial	0: não, 1: sim
Nat	Natureza do hospital	10: próprio, 20: contrato, 30: federal, 40: estadual, 50: municipal, 60: filantrópico, 70: universitário, 90: universitário com pesquisa
Espec	Especialidade da AIH	1: cirurgia, 2: obstetrícia, 3: clínica, 4: crônico, 5: psiquiatria; 6: pediatria
Custo	Custo total da AIH	MB: <200, B: 200-499, M: 500-999, A: 1000-1999, MA: >2000
Motivo	Motivo de cobrança da AIH	1: alta, 2: longa permanência, 3: transferência, 4: óbito, 6: alta por reoperação

Os dados foram pré-processados, utilizando os softwares: Microsoft Excel, Microsoft Access, SQL Explorer e o Database Desktop (que acompanham o Borland Delphi), tanto para consultas quanto para a manipulação dos dados, não sendo utilizada nenhuma ferramenta específica de pré-processamento e/ou mineração de dados.

Após a aquisição do conhecimento do domínio junto aos especialistas, foi feita a mineração destes dados, com o objetivo de generalizá-los e identificar as situações ou características das AIHs que levam a óbito.

Várias minerações foram realizadas com estes dados, variando o atributo categórico, o número de condições das regras, o limiar de parada, entre outros. Após o tempo de processamento, que em média foi de 55 segundos, várias regras foram encontradas. O número médio de regras induzidas variou de 23 a 217, dependendo das configurações de mineração.

A Figura 4.11 apresenta algumas das regras induzidas pelo Midas para este conjunto de dados, usando *Motivo* como o atributo categórico.

```
Regras INDUZIDAS:
R1: Se ESPEC = 03                       Então MOTIVO = 1 [46,28/93,83]
R2: Se DIAG = Parto                     Então MOTIVO = 1 [18,8/98,80]
R3: Se DIAG = Olhos                     Então MOTIVO = 1 [0,20/99,00]
R4: Se ESPEC = 05 e DIAG = Transtornos e CUSTO = A
                                        Então MOTIVO = 2 [0,18/67,19]
R5: Se ESPEC = 05 e CUSTO = M           Então MOTIVO = 2 [0,34/45,28]
R6: Se IDADE = 10 e DIAG = Externas     Então MOTIVO = 3 [0,00/10,00]
R7: Se DIAG = Infecção e CUSTO = MA     Então MOTIVO = 4 [0,02/33,33]
R8: Se DIAG = Infecção e CUSTO = A      Então MOTIVO = 4 [0,07/38,37]
R9: Se DIAG = Externas e CUSTO = M      Então MOTIVO = 4 [0,10/33,33]
R10: Se CUSTO = MA e PROC = 1           Então MOTIVO = 4 [0,14/17,51]
R...
```

FIGURA 4.11 – Regras induzidas das AIHs (classe: *Motivo*)

Analisando as regras acima, algumas situações interessantes podem ser verificadas, como a regra *R2*, em que tem-se: *"Se o diagnóstico for parto, então a paciente recebe alta"*. A confiança desta regra, que é de 98,80%, mostra que 1,20% das pacientes não receberam alta. A partir disto, pode ser realizada uma análise para verificar o que ocorreu com estas outras pacientes: se foram transferidas, se faleceram ou se tiveram alguma complicação no momento do parto.

Outra regra interessante é a *R7*, em que a pessoa que sofreu de alguma doença infecciosa ou parasitária teve um custo de AIH bastante elevado – ultrapassando R$2.000,00 – e veio a falecer durante a internação. O que chama a atenção são os custos implicados no tratamento, pois nenhuma outra causa teve custos elevados e resultou em óbito.

Algumas outras regras, induzidas em várias minerações pelo Midas, estão apresentadas na Figura 4.12.

```
Regras INDUZIDAS:
R1: Se DIAG = Circulatório e MOTIVO = 2
                                    Então CUSTO = MA [0,15/51,43]
R2: Se NAT = 90 e MOTIVO = 2        Então CUSTO = MA [0,09/32,59]
R3: Se IDADE = 70 e MOTIVO = 2      Então CUSTO = MA [0,03/26,67]

R4: Se CUSTO = MB                   Então PROC = Não [33,04/92,57]
R5: Se MOTIVO = 3 e CUSTO = A       Então PROC = 1   [0,09/63,64]
R6: Se DIAG = SinaisAnormais e CUSTO = MA
                                    Então PROC = 1   [0,10/73,44]
R...
```

FIGURA 4.12 – Regras induzidas das AIHs

As regras *R1, R2* e *R3* mostram que uma das principais características que levam ao alto custo da AIH é a internação prolongada dos pacientes (*Motivo = 2*), presente nas três regras. Também a ocorrência de procedimentos especiais aumenta, e muito, o custo da AIH, como pode ser observado nas regras *R4*, *R5* e *R6*.

Estas regras fornecem apenas uma visão genérica dos dados. O especialista pode, a partir delas, realizar análises mais profundas, verificando o que consiste em conhecimento útil, bem como avaliar de que forma ele poderá utilizar este conhecimento.

Algumas minerações foram refeitas no See5 para comparações preliminares. O principal ganho do Midas foi em relação ao tempo necessário para indução. A indução de regras utilizando este conjunto de dados levou, no See5, aproximadamente 93 segundos, quase o dobro levado pelo Midas.

Também foram selecionados os dados de todo o ano de 2000, totalizando 565.625 registros e utilizando somente os atributos *Sexo, Idade, Diag, Proc* e *Custo*. O objetivo era verificar o custo computacional necessário para meio milhão de registros de dados reais, que possuíssem uma baixa dimensionalidade. O atributo categórico selecionado foi *Custo*.

No teste realizado no Midas foram induzidas 164 regras no tempo de 11min39, enquanto que o See5 levou 28min18 para induzir 423 regras. Isto prova a eficiência do Midas, em termos computacionais, para grandes bases de dados reais.

4.6.2 Sistema de Informações sobre Mortalidade

O Sistema de Informações sobre Mortalidade (SIM) foi implantando em todo o país em 1975, para a obtenção regular de dados referentes aos óbitos ocorridos anualmente no país, de forma abrangente e confiável. Com o objetivo de permitir análises estatísticas com os dados de mortalidade, o sistema proporciona a classificação de estatísticas de mortalidade e a construção dos principais indicadores de saúde, permitindo estudos não apenas do ponto de vista estatístico epidemiológico, mas também do sócio-demográfico. O sistema é gerido pelo Centro Nacional de Epidemiologia (CENEPI), da Fundação Nacional de Saúde, em conjunto com as Secretarias Estaduais de Saúde, estando em processo de descentralização para as Secretarias Municipais de Saúde [SIM 2001].

As Secretarias de Saúde coletam as Declarações de Óbitos dos cartórios e fornecem ao SIM as informações nelas contidas. Uma das informações primordiais é a causa básica de óbito, a qual é codificada a partir do motivo declarado pelo médico atestante, segundo regras estabelecidas pela Organização Mundial de Saúde. Já para a direção estadual do SUS, compete, entre outras atividades, o acompanhamento, a avaliação e a divulgação dos indicadores de mortalidade no âmbito da unidade federada.

Assim, o objetivo de descobrir algum conhecimento novo e útil a partir desta base de dados e de outros dados correlacionados, é permitir uma forma complementar de avaliação dos resultados e uma nova perspectiva de gestão de programas de prevenção, controle de doenças e planejamento social.

Dentro desse contexto, os dados de mortalidade do ano de 1998 do estado do Rio Grande do Sul, que tiveram como causa básica de morte as doenças do aparelho circulatório, foram também aplicados no Midas. Esta base, de 69.505 registros, foi escolhida em vista da grande aplicabilidade e importância da utilização da mesma na gestão da saúde pública, estabelecendo prioridades orçamentárias e de prevenção, e constituindo-se um elemento precioso para estudos epidemiológicos e para elaboração e análise de vários indicadores de saúde por causa básica de morte (doença).

Esta base de dados também foi obtida com a SES, responsável pela administração das informações e repasse ao CENEPI. A estrutura interna de cada registro de óbito contém, entre outras, as seguintes informações:

a) identificação do óbito: cartório, número de registro, data de registro, número do óbito, tipo de óbito, mês do óbito, ano do óbito, local de ocorrência, município de ocorrência;

b) identificação da pessoa: estado civil, sexo, data de nascimento, idade, grau de instrução, município de residência, ocupação habitual, naturalidade, raça;

c) identificação da causa: causa básica, assistência médica, atestante, exame, cirurgia, necropsia, tipo de violência (homicídio, suicídio, acidente).

Para óbitos fetais e menores de um ano, são acrescidas outras variáveis, como: ocupação habitual e grau de instrução dos pais, idade da mãe, número de filhos da mãe (nascidos vivos e/ou nascidos mortos), duração da gestação em semanas, tipo de gravidez, tipo de parto e peso ao nascer.

Foram realizadas várias consultas SQL e estatísticas sobre esses registros, com a finalidade de adquirir um conhecimento prévio do domínio da aplicação, necessário ao processo de Descoberta de Conhecimento. Dessa forma, observa-se pela Tabela 4.14 que o maior número de óbitos no ano de 1998 foi causado pelas doenças do aparelho circulatório, com 33,46% das mortes de todo o ano no Estado, seguidas das neoplasias, com 18,06% das mortes. Mediante este índice e conforme o Ministério da Saúde [SIM 2001], o Estado do Rio Grande do Sul apresenta o segundo maior índice do país em mortalidade por doenças do aparelho circulatório, sendo a média nacional de 32,4% dos óbitos anuais.

TABELA 4.14 – Óbitos de 1998 do RS, por grupo principal de causa básica

Causa	Óbitos em 98	Percentual
Doenças parasitárias e infecciosa	2.687	3,87 %
Neoplasias	12.552	18,06 %
Doenças do sangue	260	0,37 %
Doenças endócrinas	2.556	3,68 %
Transtornos mentais	319	0,46 %
Doenças do sistema nervoso	858	1,23 %
Doenças do olho ou ouvido	10	0,01 %
Doenças do aparelho circulatório	**23.257**	**33,46 %**
Doenças do aparelho respiratório	9.851	14,17 %
Doenças do aparelho digestivo	3.246	4,67 %
Doenças da pele	104	0,15 %
Doenças osteomuscular	220	0,32 %
Doenças do aparelho geniturinário	921	1,33 %
Gravidez, parto e puerpério	147	0,21 %
Afecções perinatal	1.467	2,11 %
Malformações congênitas	692	1,00 %
Sintomas e sinais anormais	3.874	5,57 %
Lesões e outras causas externas	6.484	9,33 %
Total	**69.505**	**100 %**

A partir desta tabela, definiu-se como escopo para a mineração apenas os 23.257 registros correspondentes aos óbitos por doenças do aparelho circulatório (CID I00-I99), formando o conjunto de dados a ser minerado, tentando-se encontrar algum fator de correlação nestes dados.

A partir do entendimento do domínio, da seleção, conversão e limpeza dos dados, a estrutura dos dados a ser utilizada na etapa de mineração está apresentada na Tabela 4.15.

TABELA 4.15 – Estrutura dos registros do SIM

Nome	Descrição	Valores / intervalos
Dataobito	Mês do óbito	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
Estcivil	Estado civil	0: ignorado, 1: solteiro, 2: casado, 3: viuvo, 4: separado, 5: outro
Sexo	Sexo	0: ignorado, 1: masculino, 2: feminino
Idade	Idade	B: <2, C: 3-6, E: 7-12, A: 13-19, J: 20-35, M: 36-49, S: 50-65, I: 66-79, V: >80
Instrucao	Grau de instrução	0: ignorado, 1: nenhum, 2: 1º grau, 3: 2º grau, 4: superior
Assistmed	Assistência médica durante a doença que ocasionou a morte	0: ignorado, 1: com assistência, 2: sem assistência
Exame	Confirmação do diagnóstico através de exame complementar	0: ignorado, 1: sim, 2: não
Cirurgia	Confirmação do diagnóstico através de cirurgia	0: ignorado, 1: sim, 2: não
LocOcor	Local de ocorrência do óbito	1: Hospital, 2: Via pública, 3: Domicilio, 4: Outro
CausaBas	Causa básica do óbito (segundo a CID)	
Classe	Tipo da doença do ap. circulatório	F: Febre, H: Hipertensão, I: Isquemias, C: Cerebrovascular, A: Asterosclerose, O: Outras

O Midas levou apenas 47 segundos para induzir 133 regras destes dados. A Figura 4.13 apresenta algumas regras encontradas.

```
Regras INDUZIDAS:
R1: Se ESTCIVIL = 1                 Então CLASSE = Cerebrovascular [4,13/43,46]
R2: Se IDADE = M e ASSISTMED = 1    Então CLASSE = Cerebrovascular [2,07/43,16]
R3: Se IDADE = I                    Então CLASSE = Cerebrovascular [9,98/38,10]
R4: Se IDADE = M e ASSISTMED = 1 e CIRURGIA = 1
                                    Então CLASSE = Cerebrovascular [0,37/74,31]
R5: Se DATAOBITO = 12 e LOCOCOR = 4 Então CLASSE = Hipertensão [0,03/17,07]
R...
```

FIGURA 4.13 – Regras induzidas do SIM

A regra *R1* da figura acima mostra que 43,46% das pessoas solteiras morreram por problemas cerebrovasculares, representando 4,13% do conjunto de dados. Já os óbitos de pessoas com idade entre 36 e 49 anos, que receberam assistência médica e que fizeram cirurgia, possuíam como causa de morte doenças cerebrovasculares em 74,31% dos casos.

Por outro lado, não foi descoberto algo mais revelador ou considerado novo, principalmente pelo fato de que os dados não foram projetados com o propósito da Descoberta de Conhecimento, não estando presente na base de dados alguns atributos considerados importantes para o domínio de doenças do coração e diretamente relacionados à vida da pessoa, como sedentarismo, consumo de álcool e/ou cigarro, nível de estresse e outros fatores que aumentariam o poder de precisão dos dados.

Também neste exemplo se torna necessária a participação de um especialista, para avaliação das regras e de que forma empregar este conhecimento.

5 Conclusões

As regras e árvores de decisão são populares por sua simplicidade, flexibilidade e interpretabilidade. Entretanto, a maioria dos algoritmos de indução particionam recursivamente os dados, tornando o processamento mais demorado e a árvore construída muito grande e complexa, propensa ao *overfitting* dos dados. Já a maioria dos métodos existentes para a indução de regras operam pela geração de todos os conjuntos candidatos de condições, o que também envolve múltiplas passadas na base de dados.

Por sua vez, os conjuntos de dados reais para aplicação em mineração de dados podem ser, atualmente, muito grandes, envolvendo milhares ou mesmo milhões de registros. Executar a classificação em tais grandes dados requer o desenvolvimento de novas técnicas que generalizem estes dados e que limitem o acesso à memória secundária, minimizando o tempo de execução.

Assim, o objetivo deste trabalho foi apresentar um novo modelo de indução de classificadores, desenvolver um protótipo, validá-lo com dados de repositórios e aplicá-lo em uma base de dados real, com o objetivo de generalizar a mesma. A idéia principal era reduzir o número de passadas sobre a base de dados, o que, para bases de dados grandes e reais, pode envolver uma considerável redução de tempo.

Dessa forma, a principal contribuição deste trabalho é um novo método para indução de árvore e regras de decisão, em que o principal diferencial do algoritmo proposto é a única passada pelo conjunto de treinamento durante o processo de indução, bem como a sua inspiração proveniente de um Sistema Multiagente. Também foi desenvolvido um protótipo, o Midas, que foi comparado com o See5 e aplicado com dados de autorização de internação hospitalar e de mortalidade da SES.

Como o modelo proposto percorre uma única vez os dados, não dividindo-os recursivamente em subconjuntos (como ocorre no C4.5), um aspecto interessante é o que diz respeito ao tempo de processamento para a geração do classificador. No Midas, quando os dados tiverem muitos atributos, a alta dimensionalidade (medida por esses valores) aumenta o tempo de processamento; enquanto que no C4.5, o tempo de processamento será aumentado pelo grande volume da base de dados.

Isto foi comprovado pela complexidade do C4.5 – $O(m.n^2)$, e do Midas – $O(n.m!)$, em que o primeiro possui um custo computacional dado pelo número de registros, enquanto que no Midas o custo computacional é dado pelo número de atributos. Dessa forma, apesar da ordem de complexidade do Midas ser exponencial, ou seja, pior que a do C4.5, o Midas apresenta melhores resultados, pois nas bases de dados reais tem-se uma maior quantidade de registros (milhões) do que de atributos (dezenas).

Assim, o Midas é mais adequado para grandes conjuntos de dados e que contenham uma baixa dimensionalidade, para obter-se uma visão genérica e resumida dos mesmos.

O uso da mesma tabela de ocorrências para realizar a indução de toda a árvore, apesar de possibilitar uma generalização dos dados, faz com que não se tenha a informação em nível de nodo-partição, pois a tabela considera as ocorrências dos relacionamentos dois a dois de todos os dados e não apenas dos exemplos da partição em questão. Isso é um problema delicado, pois assim perde-se a acurácia preditiva do classificador, ao mesmo tempo em que se avança no número de níveis da árvore ou no número de condições.

Entretanto, se o objetivo for de generalizar os dados, o algoritmo proposto para a indução de regras, consegue obter uma boa relação entre a simplicidade das regras, o tempo de indução e a taxa de erro. Já o modelo de indução de AD, embora apresentasse resultados aceitáveis em alguns exemplos, percebe-se que é inferior ao modelo de indução de regras apresentado.

Nos testes realizados com dados de repositório, utilizando o See5 para realizar comparações com o Midas, percebeu-se que o nível de confiança das regras do Midas é bastante alto. Outro benefício do protótipo é o número de regras obtidas, que, em média, foi menor. Isso é muito importante para grandes bases de dados reais, em que, normalmente, são encontradas muitas regras, o que dificulta a compreensão de todo o conhecimento descoberto.

É interessante notar também que a indução de regras pelo Midas forneceu uma taxa de erro 80,53% inferior que a indução de regras pelo See5, quando foram desconsiderados como erro os exemplos não classificados.

Já a aplicação de dados reais da Secretaria da Saúde no Midas comprovou a simplicidade do modelo e o processamento ágil, além da grande modularidade dos resultados. O Midas induziu as regras de decisão muito rapidamente, generalizando grandes bases de dados através de poucas regras.

O protótipo apresentado neste trabalho não pode ser considerado terminado, necessitando de alguns aperfeiçoamentos, de modo a permitir o melhoramento do trabalho desenvolvido até o momento. Entre as limitações presentes, pode-se destacar a falta de tratamento para atributos contínuos e desconhecidos, de prioridades nos atributos e de um mecanismo de controle para regras não ordenadas, além da alta taxa de erro na indução da AD.

Tendo em vista estas deficiências, alguns pontos que poderiam ser futuramente desenvolvidos para obter-se classificadores mais precisos, seriam: a implementação de técnicas automáticas de discretização para tratamento de atributos contínuos, a implementação de métodos de tratamento para atributos desconhecidos, a inclusão de técnicas que permitam a seleção e/ou atribuição de prioridades para atributos, o desenvolvimento de técnicas avançadas de pós-poda, a implementação de ferramentas gráficas de análise de árvores de decisão (permitindo a obtenção do máximo de informação), e também a inclusão de um mecanismo para escolha de regras não ordenadas e de um método mais avançado de simplificação e redução de regras.

Outras técnicas também poderiam ser incluídas, como as de escolha do nodo e definição da partição, de forma a possibilitar comparações e avaliações da técnica mais adequada ao modelo proposto. O tempo de indução também poderia ser reduzido pela otimização no acesso e manipulação das bases de dados – utilizando arquivos do tipo texto, por exemplo, bem como poderiam ser realizadas otimizações de implementação. Além disso, outros testes devem ser realizados de forma a determinar, mais exatamente, em que situações e/ou domínios, o Midas apresenta melhores resultados.

A aplicação futura, mais aprofundada e detalhada, dos dados de saúde da SES, através do projeto "Desenvolvimento de Metodologia para Extração de Conhecimento de Bases de Dados da Saúde do Estado para Avaliação e Planejamento", também se faz necessária – em conjunto com os especialistas, de modo a identificar regras que possam realmente ser utilizados pela SES na otimização de recursos públicos, além de permitir uma melhor avaliação do protótipo desenvolvido em bases de dados reais.

Finalizando, o modelo proposto para indução de regras generaliza de uma forma adequada os dados, retendo somente pequenas frações dos dados, obtendo regras simples e evitando o *overfitting* dos mesmos. Assim, obteve-se um modelo para generalização dos dados e não para a realização de futuras classificações. O algoritmo proposto para indução de árvores e regras de decisão é mais adequado para bases de dados grandes e que contenham uma baixa dimensionalidade, como são as bases de dados reais.

Anexo 1

Descrição das Bases de Dados

As bases de dados utilizadas para validação e comparação entre os algoritmos são de domínio público e foram obtidas do *UCI Machine Learning Repository,* disponível em <http://www.ics.uci.edu/~mlearn/MLSummary.html> [UCI 2001]. Abaixo (Tabela A.1), estão apresentadas as principais características de cada uma delas:

TABELA A.1 – Dados do repositório UCI

Base	Exemplos	Atributos	Classes	Perdidos
Balance	625	4	3	0
Breast Cancer	699	9	2	2%
Car	1.728	6	4	0
Chess	28.056	6	17	0
Genetics	3.190	60	3	0
Mushrooms	8.124	22	2	30%
Nursery	12.960	8	5	0
Voting	435	16	2	0
Zoo	101	16	7	0

a) *Balance*

Este conjunto de dados foi gerado por Tim Hume para modelar resultados de experimentos psicológicos. Cada exemplo é classificado com uma medida de inclinação do equilíbrio: para a *direita* (R), para a *esquerda* (L) ou *equilibrado* (B). Os atributos presentes são: o *peso direito*, a *distância direita*, o *peso esquerdo* e a *distância esquerda*. Uma forma de encontrar cada classe correta é o maior valor entre (distância esquerda * peso esquerdo) e (distância direita * peso direito). Se o primeiro resultado for maior, a classe do exemplo é esquerda, e se for menor, a classe é direita. Caso os valores sejam iguais, está balanceado. O conjunto de dados contém 625 registros e 3 classes.

b) *Breast Cancer*

Esta é uma das bases de dados de câncer de mamas disponíveis no UCI, que foi obtida dos hospitais da *University of Wisconsin,* por William H. Wolberg. O problema consiste em predizer se uma amostra de tecido de uma determinada paciente é *maligno* (4) ou *benigno* (2). Possui 699 amostras e a cada amostra fora atribuídos nove atributos: *densidade de massa*, *uniformidade do tamanho da célula, uniformidade da forma da célula, aderência da margem, tamanho da célula, núcleo, cor, nucléolo* e *mitose*.

c) *Car*

Doada por Marko Bohanec e Blaz Zupan, esta base com 1.728 instâncias tem o objetivo de realizar uma avaliação de carros em termos de preço e conforto, distribuindo as instâncias em quatro classes de aceitação: *não aceitável* (unacc), *aceitável* (acc), *bem aceitável* (good) e *muito bem aceitável* (vgood). Contém seis atributos: *preço de compra*, *preço de manutenção*, *portas*, *capacidade/pessoas*, *porta-malas* e *segurança*.

d) *Chess*

Este conjunto de dados objetiva definir, em um jogo de xadrez, o número mínimo de movimentos para o branco ganhar: de 0 a 16 movimentos, em que o rei branco e a torre branca estão contra o rei preto. Este conjunto de exemplos, inicialmente desenvolvido por Michael Bain e Arthur van Hoff, contém 28.056 registros, 17 classes e 6 atributos, que representam as posições das peças dadas pelas coordenadas do tabuleiro: *coluna do rei branco*, *linha do rei branco*, *coluna da torre branca*, *linha da torre branca*, *coluna do rei preto* e *linha do rei preto*.

e) *Genetics*

O domínio deste conjunto de 3.190 exemplos, desenvolvido por Geoffrey Towell, é referente à área de biologia molecular. Juntas de união são pontos na seqüência de DNA nas quais o DNA supérfluo é removido durante a criação da proteína. O problema consiste em reconhecer as seguintes classes: fronteiras *exon/intron* (EI), fronteiras *intron/exon* (IE) ou *nenhuma delas* (n), durante o processo de criação de proteína. Cada registro contém 60 nucleotídeos (elemento da seqüência do DNA), cada um representado pelos valores A, C, G ou T.

f) *Mushrooms*

Este conjunto de dados, originário da Audobon Society Field Guide, inclui descrições de exemplos hipotéticos correspondentes a 23 espécies de cogumelos da família Agaricus e Lepiota. Cada espécie é identificada como *comestível* (e) e *não comestível* (p). Para cada um dos 8.124 casos existem 22 atributos que definem suas principais características, como: *forma, superfície, cor, manchas, odor, espaçamento, tamanho, forma e origem do talo, número e tipo dos anéis, população, habitat,* etc. Não existe uma simples regra para determinar se o cogumelo é comestível. Entretanto, o odor pode predizer uma classe com 98% de precisão.

g) *Nursery*

Este conjunto de dados foi gerado por Marko Bohanec e Blaz Zupan para avaliar a aceitação de crianças em creches, de acordo com a necessidade e aspecto social da família, classificando entre as três classes: *não recomendado* (not_recom)*, recomendado* (recommend) e *prioritário* (priority). O conjunto de dados contém 12.960 registros e 8 atributos, que são: *ocupação dos pais*, *berçário de criança, forma da família*, *número de filhos*, *condições de moradia*, *estrutura financeira, condições sociais* e *condições de saúde*.

h) *Voting*

Fornecido por Jeff Schlinnrer, em 1987, este conjunto de dados composto por 435 registros corresponde à caracterização dos participantes *republicanos* (republican) e *democratas* (democrat) no 98° Congresso dos Estados Unidos, em 1984, através de suas respostas em relação a 16 problemas (atributos) distintos: dificuldades infantis, água compartilhada, adoção da resolução de orçamento, taxa de doutor, ajuda a El Salvador, grupo religioso nas escolas, proibição de teste anti-satélite, ajuda à Nicarágua, míssil MX, imigração, redução de corporação, gasto com educação, fundo para processos, crime, exportações isentos de direitos aduaneiros e ato de administração de exportação para África do Sul.

i) *Zoo*

Esta é uma base de dados disponibilizada por Richard Forsyth, que classifica cada registro em uma das sete classes de animais: *mamífero, pássaro, réptil, peixe, anfíbio, inseto* e *molusco*. O conjunto de dados possui 101 registros e 16 atributos, identificando os seguintes atributos: *pelos, penas, ovos, leite, nadadeira, patas, rabo, dentes, espinha dorsal, veneno, asa, aquático, predador, respiração, doméstico e tamanho.*

Anexo 2

Funcionamento do protótipo

A figura abaixo apresenta o fluxo de funcionamento do Midas, em que os principais procedimentos desenvolvidos estão apresentados em cada retângulo.

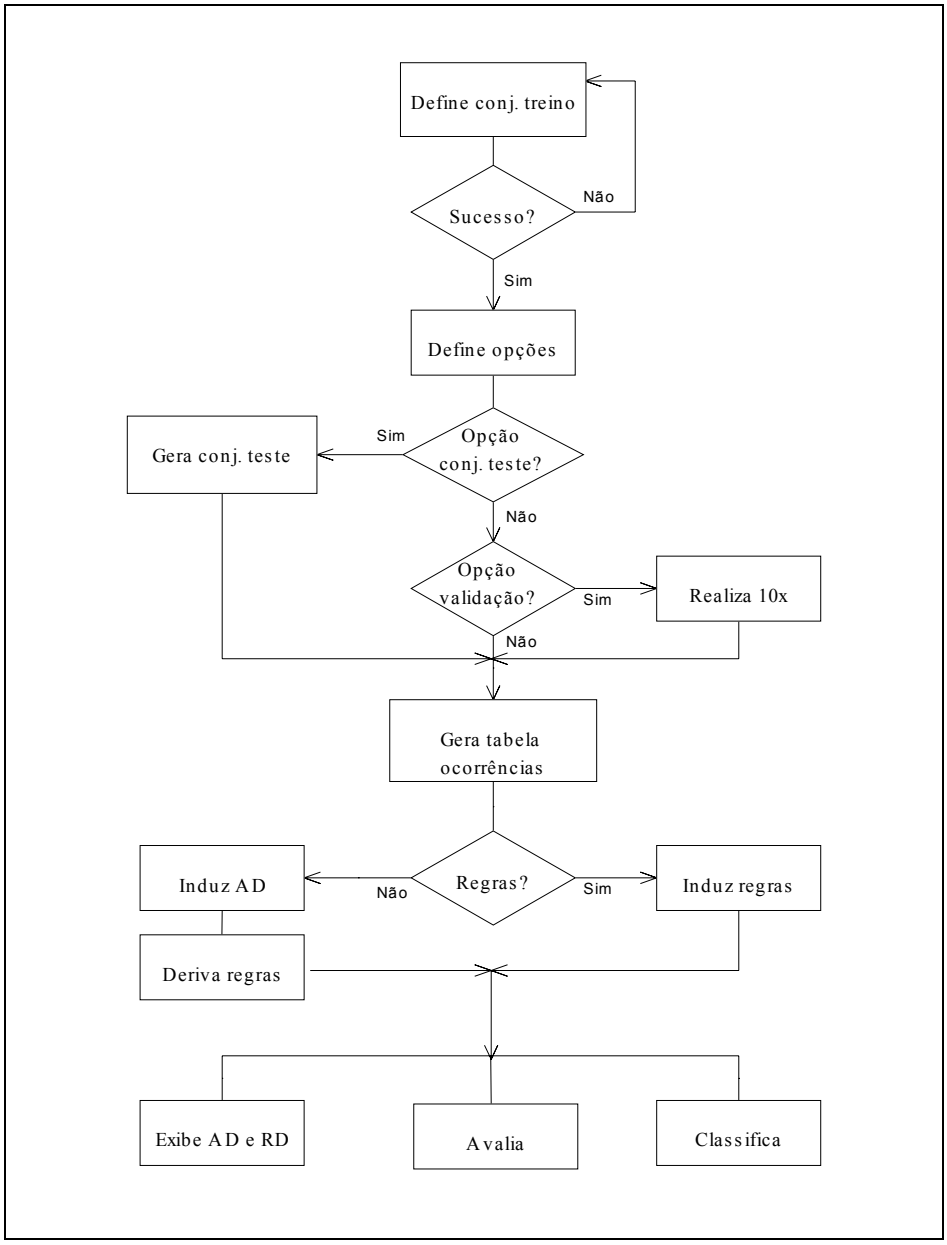

FIGURA A.1 – Funcionamento do modelo proposto

Bibliografia

[ADR 97] ADRIAANS, Pieter; ZANTINGE, Dolf. **Data Mining.** Harlow: Addison-Wesley, 1997. 158p.

[ALS 98] ALSABTI, Khaled; RANLA, Sanjay; SINGH, Vineet. **Clouds:** a decision tree classifier for large databases. Florida: University of Florida, 1998. 34p. Disponível em: <http://citeseer.nj.nec.com/alsabti98clouds.html>. Acesso em: 20 nov. 2001.

[ALV 97] ALVARES, Luis Otavio; SICHMAN, Jaime Simão. Introdução aos sistemas multiagentes. In: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA, 16., 1997, Brasília. **Anais...** Brasília: SBC, 1997. p.1-37.

[AVI 98] ÁVILA, Bráulio Coelho. Data Mining. In: ESCOLA REGIONAL DE INFORMÁTICA DA SBC, 6., 1998, Blumenau. **Anais...** Blumenau: SBC, 1998. p.87-106.

[BAR 2000a] BARANAUSKAS, José Augusto; MONARD, Maria Carolina. **Reviewing some Machine Learning concepts and methods.** São Carlos: USP, 2000, 52p. (Relatórios Técnicos do ICMC, n. 102).

[BAR 2000b] BARANAUSKAS, José Augusto; MONARD, Maria Carolina. **An unified overview of six supervised symbolic machine learning inducers.** São Carlos: USP, 2000, 63p. (Relatórios Técnicos do ICMC, n. 103).

[BAR 2001] BARANAUSKAS, José Augusto; MONARD, Maria Carolina. **Extração automática de conhecimento por múltiplos indutores.** 2001. 181p. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

[CAN 2000] CANTÚ, Marco. **Dominando o Delphi 5:** a bíblia. São Paulo: Makron Books, 2000. 967p.

[CEN 88] CENDROWSKA, Jadzia. PRISM: an algorithm for inducing modular rules. In: GAINES, B. R.; BOOSE, J. H. **Knowledge Based Systems.** London: Academic Press, 1988. v. 1, p. 255-276.

[CLA 89] CLARK, Peter; NIBLETT, Tim. The CN2 induction algorithm. **Machine Learning Journal**, [S.l.], v. 3, n. 4, p. 261-283, 1989.

[DAT 2001] DATA Mining Tools See5 and C5.0. Disponível em: <http://www.rulequest.com/ See5>. Acesso em: 20 mar. 2001.

[ESP 97] ESPOSITO, Floriana; MALERBA, Donato; SEMERARO, Giovanni. A comparative analysis of methods for pruning decision trees. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, New York, v. 19, n. 5, p. 476-491, May 1997.

[FAY 96] FAYYAD, Usama M.; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic et al. **Advances in Knowledge Discovery and Data Mining.** Califórnia: American Association for Artificial Intelligence, 1996. 611p.

[FEL 97] FELDENS, Miguel Artur. **Engenharia da Descoberta de Conhecimento em Bases de Dados:** estudo e aplicação na área da saúde. 1997. 90f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[FON 94] FONSECA, José Manuel Matos Ribeiro da. **Indução de árvores de decisão**: HistClass – proposta de um algoritmo não paramétrico. 1994. 140p. Dissertação (Mestrado em Engenharia Informática) – Departamento de Informática, Universidade Nova de Lisboa, Lisboa.

[FRO 97] FROZZA, Rejane. **Simula:** ambiente para desenvolvimento de sistemas multiagentes reativos. 1997. 116f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[GAM 99] GAMA, João Manuel Portela da. **Combining classification algorithms.** 1999. 195p. Tese (Doutorado) – Departamento de Ciência de Computadores, Universidade do Porto, Lisboa.

[HAL 2000] HALMENSCHLAGER, Carine. **Utilização de agentes na Descoberta de Conhecimento.** 2000. 55f. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[HAL 2001] HALMENSCHLAGER, Carine; ALVARES, Luis Otavio Campos. Siad: um Sistema de Indução de Árvores de Decisão baseado em agentes. In: CONFERENCIA LATINOAMERICANA DE INFORMATICA, CLEI, 27., 2001, Mérida, Venezuela. **[Articulos].** Mérida: Centro Latinoamericano de Estudios en Informatica, 2001. 1 CD.

[JOH 97] JOHN, George H. **Enhancements to the Data Mining process.** 1997. 194f. Tese (Doctor of Philosopfy) – Department of Computer Science, Stanford University, Stanford.

[LEW 91] LEWIS, David D. **Evaluating text categorization:** computer and information science. Massachusetts: University of Massachusetts, 1991. Disponível em: <http://www.research.att.com/~lewis/chronobib.html>. Acesso em: 22 nov. 2001.

[MIC 69] MICHALSKI, R. S. On the quasi-minimal solution of the general covering problem. In: INTERNATIONAL SYMPOSIUM ON INFORMATION PROCESSING, 5., 1969, Yugoslavia. **Proceedings...** Yugoslavia: FCIP, 1969. p. 125-128.

[MOR 93] MORIK, Katharina et al. **Knowledge Acquisition and Machine Learning:** theory, methods and applications. London: Academic Press, 1993. 305p.

[MUR 97] MURTHY, Kolluru Venkata Sreerama. **On growning better decision trees for data.** 1997. 163f. Tese (Doctor of Philosophy) – The Johns Hopkins University, Baltimore.

[NOG 2000] NOGUEZ, José Hiram. **Técnicas de Mineração de Dados no processo de Descoberta de Conhecimento em Bases de Dados.** 2000. 47f. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[PRA 2001] PRADO, Hércules Antonio do. **Orpheo:** uma estrutura de trabalho para integração dos paradigmas de aprendizado supervisionado e não-supervisionado. 2001. 170f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[QUI 88] QUINLAN, J. R. Simplifying decision trees. In: GAINES, B. R.; BOOSE, J. H. **Knowledge Based Systems.** London: Academic Press, 1988. v. 1, p. 241-254.

[QUI 93] QUILAN, J. Ross. **C4.5**: programs for Machine Learning. San Mateo: Morgan Kaufmann, 1993. 302p.

[QUI 94] QUINLAN, J. R. **Comparing connectionist and symbolic methods**. Australia: University of Sydney, 1994. Disponível em: <http://citeseer.nj.nec.com/quinlan94comparing.html>. Acesso em: 12 nov. 2001.

[SEE 2001] SEE5: an informal tutorial. Disponível em: <http://www.rulequest.com/see5-win.html>. Acesso em: 22 mar. 2001.

[SIM 2001] SIM: Sistema de Informações sobre Mortalidade. Disponível em: <http://www.funasa.gov.br/sis/sis01_sim.htm>. Acesso em: 02. dez. 2001.

[SMI 80] SMITH, R. G. The contract net protocol: high-level communication and control in a distributed problem solver. **IEEE Transactions on Computers**, New York, v. 29, n. 12, p. 1104-1113, Dec. 1980.

[SOU 98] SOUSA, M. S. et al. Data Mining: a database perspective. In: INTERNATIONAL CONFERENCE ON DATA MINING, 1998, Rio de Janeiro. **Data Mining.** Boston: WIT, 1998. p. 413-431.

[UCI 2001] UCI Machine Learning Repository. Disponível em: <http://www.ics.uci.edu/~mlearn/ MLSummary.html>. Acesso em: 04 mar. 2001.

[WIT 2000] WITTEN, Ian H.; FRANK, Eibe. **Data Mining**: practical machine learning tools and techniques with Java implementations. San Mateo: Morgan Kaufmann, 2000. 349p.