Pré-processamento de Dados em Aprendizado de Máquina Supervisionado

Gustavo Enrique de Almeida Prado Alves Batista

Pré-processamento de Dados em Aprendizado de Máquina Supervisionado

Gustavo Enrique de Almeida Prado Alves Batista

Orientadora: *Profa. Dra. Maria Carolina Monard*

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências — Ciências de Computação e Matemática Computacional.

USP - São Carlos
Março/2003

Resumo

A qualidade de dados é uma das principais preocupações em Aprendizado de Máquina — AM — cujos algoritmos são freqüentemente utilizados para extrair conhecimento durante a fase de Mineração de Dados — MD — da nova área de pesquisa chamada Descoberta de Conhecimento de Bancos de Dados. Uma vez que a maioria dos algoritmos de aprendizado induz conhecimento estritamente a partir de dados, a qualidade do conhecimento extraído é amplamente determinada pela qualidade dos dados de entrada.

Diversos aspectos podem influenciar no desempenho de um sistema de aprendizado devido à qualidade dos dados. Em bases de dados reais, dois desses aspectos estão relacionados com (i) a presença de valores desconhecidos, os quais são tratados de uma forma bastante simplista por diversos algoritmos de AM, e; (ii) a diferença entre o número de exemplos, ou registros de um banco de dados, que pertencem a diferentes classes, uma vez que quando essa diferença é expressiva, sistemas de aprendizado podem ter dificuldades em aprender o conceito relacionado com a classe minoritária.

O problema de tratamento de valores desconhecidos é de grande interesse prático e teórico. Em diversas aplicações é importante saber como proceder quando as informações disponíveis estão incompletas ou quando as fontes de informações se tornam indisponíveis. O tratamento de valores desconhecidos deve ser cuidadosamente planejado, caso contrário, distorções podem ser introduzidas no conhecimento induzido. Neste trabalho é proposta a utilização do algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação. Imputação é um termo que denota um procedimento que substitui os valores desconhecidos de um conjunto de dados por valores plausíveis. As análises conduzidas neste trabalho indicam que a imputação de valores desconhecidos com base no algoritmo K-VIZINHOS MAIS PRÓXIMOS pode superar o desempenho das estratégias internas utilizadas para tratar valores desconhecidos pelos sistemas C4.5 e CN2, bem como a IMPUTAÇÃO PELA MÉDIA OU MODA, um método amplamente utilizado para tratar valores desconhecidos.

O problema de aprender a partir de conjuntos de dados com classes desbalanceadas é de crucial importância, um vez que esses conjuntos de dados podem ser encontrados em diversos domínios. Classes com distribuições desbalanceadas podem se constituir em um gargalo significante no desempenho obtido por sistemas de aprendizado que assumem uma distribuição balanceada das classes. Uma solução para o problema de aprendizado com distribuições desbalanceadas de classes é balancear artificialmente o conjunto de dados. Neste trabalho é avaliado o uso do método de seleção unilateral, o qual realiza uma remoção cuidadosa dos casos que pertencem à classe majoritária, mantendo os casos da classe minoritária. Essa remoção cuidadosa consiste em detectar e remover casos considerados menos confiáveis, por meio do uso de algumas heurísticas.

Uma vez que não existe uma análise matemática capaz de predizer se o desempenho de um método é superior aos demais, análises experimentais possuem um papel importante na avaliação de sistema de aprendizado. Neste trabalho é proposto e implementado o ambiente computacional DISCOVER LEARNING ENVIRONMNET — DLE — o qual é um *framework* para desenvolver e avaliar novos métodos de pré-processamento de dados. O ambiente DLE é integrado ao projeto DISCOVER, um projeto de pesquisa em desenvolvimento em nosso laboratório para planejamento e execução de experimentos relacionados com o uso de sistemas de aprendizado durante a fase de Mineração de dados do processo de KDD.

Abstract

Data quality is a major concern in Machine Learning, which is frequently used to extract knowledge during the Data Mining phase of the relatively new research area called Knowledge Discovery from Databases — KDD. As most Machine Learning algorithms induce knowledge strictly from data, the quality of the knowledge extracted is largely determined by the quality of the underlying data.

Several aspects may influence the performance of a learning system due to data quality. In real world databases, two of these aspects are related to (i) the presence of missing data, which is handled in a rather naive way by many Machine Learning algorithms; (ii) the difference between the number of examples, or database records, that belong to different classes since, when this difference is large, learning systems may have difficulties to learn the concept related to the minority class.

The problem of missing data is of great practical and theoretical interest. In many applications it is important to know how to react if the available information is incomplete or if sources of information become unavailable. Missing data treatment should be carefully thought, otherwise bias might be introduced into the knowledge induced. In this work, we propose the use of the K-NEAREST NEIGHBOUR algorithm as an imputation method. Imputation is a term that denotes a procedure that replaces the missing values in a data set by some plausible values. Our analysis indicates that missing data imputation based on the K-NEAREST NEIGHBOUR algorithm can outperform the internal missing data treatment strategies used by C4.5 and CN2, and the MEAN OR MODE IMPUTATION, a widely used method for treating missing values.

The problem of learning from imbalanced data sets is of crucial importance since it is encountered in a large number of domains. Imbalanced class distributions might cause a significant bottleneck in the performance obtained by standard learning methods, which assume a balanced distribution of the classes. One solution to the problem of learning with skewed class distributions is to artificially balance the data set. In this work we propose the use of the one-sided selection method, which performs a careful removal of cases belonging to the majority class while leaving untouched all cases from the minority class. Such careful removal consists of detecting and removing cases considered less reliable, using some heuristics. An experimental application confirmed the efficiency of the proposed method.

As there is not a mathematical analysis able to predict whether the performance of a learning system is better than others, experimentation plays an important role for evaluating learning systems. In this work we propose and implement a computational environment, the DISCOVER LEARNING ENVIRONMENT — DLE — which is a framework to develop and evaluate new data pre-processing methods. The DLE is integrated into the DISCOVER project, a major research project under development in our laboratory for planning and execution of experiments related to the use of learning systems during the Data Mining phase of the KDD process.

Aos meus pais,
Joselito e Margarida,

Às minhas irmãs,
Anapaula e Analúcia,

À Maria Carolina Monard.

Agradecimentos

Ainda me lembro do dia que conheci a professora Carolina. Parece até mesmo que não faz muito tempo. Eu ainda estava cursando o segundo grau e estava passeando de férias pela USP. Um dos meus primos, José Pacheco, que sempre me incentivou a ingressar na carreira acadêmica e que na época era aluno de mestrado da profa. Carolina, disse-me que iria me apresentar à sua orientadora. Chegando à sala da profa. Carolina, ele me apresentou. Ela me cumprimentou, foi extremamente gentil, como sempre, e os dois começaram a tratar dos assuntos da dissertação de mestrado dele.

Naquela época, eu jamais poderia imaginar o quanto aquela senhora, de sotaque espanhol carregado, poderia representar na minha vida. Nos últimos oito anos ela não foi somente a minha orientadora de mestrado e doutorado, mas também a minha orientadora na vida. Ela me ensinou grande parte do que eu sei sobre Aprendizado de Máquina. Inúmeras foram as nossas discussões sobre o tema, e inúmeras vezes eu ouvi um doce "não é bem assim, filhinho" quando eu estava errado. Mas a profa. Carolina também me ensinou outras coisas tão valiosas quanto o conhecimento acadêmico. Uma delas, eu pretendo carregar comigo para sempre: a postura ética no trabalho e na vida.

Carolina esteve comigo nos bons momentos, como nas festas de aniversário do Labic. Nos momentos chatos do trabalho quando nos reuníamos à noite e aos finais de semana para escrever artigos, e nos momentos difíceis, como no falecimento de minha avó.

Minha avó, Margarida, da qual nunca posso esquecer, sempre acolhedora, ensinou-me muito nos anos em que morei com ela na graduação.

Importante, também, foi o apoio que sempre recebi da minha família. Agradeço aos meus pais por sempre terem me apoiado a estudar. À minha mãe por sempre me apoiar nas decisões que tomei em minha carreira. E, em especial, ao meu pai por freqüentemente me dizer que fazer uma pós-graduação era o caminho certo.

Agradeço, também, à Claudia, minha namorada e ao mesmo tempo minha colega de doutorado. Esses anos de doutorado teriam sido muito mais difíceis se você não estivesse comigo todos os dias. Obrigado por me apoiar sempre, seja qual for a coisa que eu resolva fazer.

Durante esses anos, dois amigos me ajudaram muito com o meu trabalho. Primeiro, José Augusto pelas diversas conversas sobre Mineração de Dados, KDD e a criação do

projeto DISCOVER. Eu tenho utilizado como referência de qualidade muitos trabalhos desenvolvidos por você. E o Ronaldo, um pesquisador com uma capacidade incrível, com quem tenho desenvolvido muitas das idéias que têm dado vida ao DISCOVER.

Agradeço, também aos amigos Daniel, Kaminski, Valmir e Rodrigo pelos momentos de bagunça nas viagens e nas reuniões de confraternização. Um obrigado especial ao Valmir por suportar as nossas brincadeiras. E aos amigos Huei e Wu pelas viagens que fizemos juntos.

Aos amigos do Labic: Jaque, Claudinha, Patrícia, Gedson, José Flávio, Cris, Walter, Adriano, Marcos Geromini e Marcos Paula. Obrigado também à Talita por nos ajudar na parte de Engenharia de *Software* do projeto DISCOVER.

Agradeço ao pessoal da pós-graduação do ICMC, e em especial à Beth, à Laura, à Marília e à Ana Paula, por terem respondido às minhas inúmeras perguntas sobre o funcionamento da pós. Ao pessoal da biblioteca por serem todos sempre prestativos.

Gostaria de lembrar também da Alice e da Sofia por estarem comigo nas madrugadas que estive redigindo esta tese.

Um agradecimento especial a dois professores do ICMC que me ajudaram muito durante esses anos. Ao André que sempre me impressionou com o seu dinamismo, capacidade, bom humor e cordialidade. À Solange, por ter me ajudado diversas vezes com os meus problemas na pós, e por ter me ajudado em várias oportunidades. Muito do que eu aprendi em contato com outros pesquisadores externos ao ICMC eu devo à você, Solange.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas

10-NNI . . . . . *10-Nearest Neighbour Imputation*
AM . . . . . Aprendizado de Máquina
API . . . . . *Application Programming Interface*
ARFF . . . . . *Attribute Relation Format File*
AUC . . . . . *Area under the ROC curve*
DLE . . . . . *Discover Learning Environment*
DOL . . . . . *Discover Object Library*
DSX . . . . . *Discover Dataset Sintax*
EM . . . . . *Expectation-Maximization*
HEOM . . . . . *Heterogeneous Euclidian-Overlap Metric*
HVDM . . . . . *Heterogeneous Value Difference Metric*
IA . . . . . Inteligência Artificial
KDD . . . . . *Knowledge Discover from Databases*
MAD . . . . . *Mean Absolute Difference*
MAR . . . . . *Missing at Random*
MCAR . . . . . *Missing Completely at Random*
MD . . . . . Mineração de Dados
ML . . . . . *Maximum Likelihood*
$\mathcal{MLC}++$ . . . . . *Machine Learning Library in C++*
MSE . . . . . *Mean Squared Error*
NMAR . . . . . *Not Missing at Random*
Perl . . . . . *Practical Extraction and Report Language*
PLI . . . . . Programação Lógica Indutiva
ROC . . . . . *Receiver Operating Characteristic*
SGBD . . . . . Sistema Gerenciador de Bancos de Dados
SQL . . . . . *Structured Query Language*
VDM . . . . . *Value Difference Metric*

Weka . . . . . *Waikato Environment for Knowledge Analysis*
XML . . . . . *Extensible Markup Language*
Yale . . . . . *Yet Another Learning Environment*

Capítulo 1

Introdução

1.1 Considerações Iniciais

Neste capítulo é apresentada uma descrição geral desta tese, com o objetivo de fornecer ao leitor uma visão global dos problemas tratados e dos objetivos principais, bem como uma descrição da organização deste trabalho.

Este capítulo está organizado da seguinte forma: na Seção 1.2 é apresentada uma discussão a respeito da importância da qualidade dos dados para a extração automática de conhecimento; na Seção 1.3 é apresentada brevemente a fase de pré-processamento de dados do processo de *Descoberta de Conhecimento em Bancos de Dados*, que tem como principal objetivo melhorar a qualidade dos dados a serem utilizados na extração de conhecimento; na Seção 1.4 são apresentados os objetivos desta tese, incluindo uma breve descrição dos problemas que são tratados neste trabalho; na Seção 1.5 são brevemente mostradas as principais contribuições desta tese, as quais são apresentadas e discutidas nos capítulos seguintes; por fim, na Seção 1.6 é apresentada a organização deste trabalho, com uma breve descrição do conteúdo de cada capítulo.

1.2 Qualidade de Dados

Por anos a comunidade científica que pesquisa na área de Aprendizado de Máquina — AM — tem utilizado repositórios de dados, tal como o repositório da Universidade da Califórnia em Irvine (UCI) (Blake & Merz, 1998), para avaliar propostas de melhorias e novos sistemas de aprendizado. Embora a utilidade desses repositórios seja reconhecida

pelos pesquisadores da área (Soares, 2002), os repositórios de dados normalmente apresentam dados previamente pré-processados, sem muitos dos problemas que podem ser encontrados em dados do "mundo real[1]".

Recentemente foi fundada uma nova área de pesquisa conhecida como *Descoberta de Conhecimento em Bancos de Dados* — KDD[2] — a qual visa extrair conhecimento de grandes bases de dados. Ao contrário dos conjuntos de dados presentes em repositórios de dados, dados extraídos diretamente de sistemas de gerenciamento de dados freqüentemente apresentam diversos problemas, tais como: grande quantidade de ruído e inconsistências; excesso de valores desconhecidos; classes *desbalanceadas*, ou seja, uma grande desproporção entre as distribuições das classes; entre outros.

A qualidade dos dados é uma das principais preocupações em AM e KDD. Esse fato ocorre pois a maioria dos métodos utilizados nessas áreas, tais como os algoritmos de aprendizado proposicional tratados neste trabalho, induz conhecimento estritamente a partir dos dados, sem utilizar outro conhecimento externo[3]. Dessa forma, a qualidade do conhecimento extraído é amplamente determinada pela qualidade dos dados fornecidos como entrada.

Apesar de que muitos dos algoritmos desenvolvidos pela comunidade de AM, e demais algoritmos utilizados em KDD, serem robustos à problemas existentes nos dados, muitos pesquisadores têm reportado a extrema falta de qualidade dos dados extraídos de sistemas de gerenciamento de dados (Garner, Cunningham, Holmes, Nevill-Manning & Witten, 1995; Devaney & Ram, 1997; Provost & Danyluk, 1995; Lakshminarayan, Harp & Samad, 1999). Se os problemas presentes nos dados forem identificados e tratados antes dos dados serem fornecidos a um algoritmo de extração de conhecimento, então espera-se que o conhecimento extraído seja mais representativo e mais preditivo.

Por outro lado, é necessário verificar se os métodos de tratamento de dados utilizados pela comunidade de AM são válidos no contexto de KDD. Pode-se dizer que KDD tem uma necessidade urgente pela validação dos métodos de tratamento de dados amplamente utilizados pela comunidade de AM e, possivelmente, pelo desenvolvimento e avaliação de

[1] Neste trabalho os termos *dados reais* e *dados do mundo real* referem-se a dados colhidos diretamente de sistemas de armazenamento de dados. Dados provenientes de repositórios de dados são chamados de *naturais* (Saitta, Giordana & Neri, 1995; Kohavi & Kunz, 1997) para distinguir dos conjuntos de dados *artificiais*.

[2] *Knowledge Discovery from Databases*. A sigla em inglês KDD é utilizada neste trabalho por ser amplamente difundida na literatura.

[3] Uma das exceções são os sistemas de aprendizado relacional, tais como os sistemas de *Programação Lógica Indutiva* — PLI, os quais utilizam *conhecimento de fundo* no processo de indução.

novos métodos, os quais não foram foco de pesquisa na área de AM. Este trabalho tem como principal objetivo ajudar a suprir essas necessidades.

No processo de KDD existe uma fase que tem como finalidade melhorar a qualidade dos dados. Essa fase é conhecida como *pré-processamento de dados* e seu objetivo principal é a identificação e remoção de problemas presentes nos dados antes que os métodos de extração de conhecimento sejam aplicados. Na próxima seção são discutidas algumas das características dessa fase.

1.3 Pré-processamento de Dados

A fase de pré-processamento inicia tão logo os dados são coletados e organizados na forma de um conjunto de dados. Podem existir diversos objetivos na fase de pré-processamento. Um deles é solucionar problemas nos dados, tais como identificar e tratar dados corrompidos, atributos irrelevantes e valores desconhecidos. Pode-se também estar interessado em aprender mais a respeito dos dados, o que pode ser feito, por exemplo, por meio de visualizações. Ou ainda, pode-se estar interessado em alterar a estrutura dos dados, por exemplo, por meio da alteração do grau de granularidade dos dados. As ações realizadas na fase de pré-processamento visam preparar os dados para que a fase seguinte, a fase de extração de conhecimento, seja mais efetiva.

De uma forma geral, pré-processamento de dados é um processo semi-automático. Por semi-automático entende-se que essa fase depende da capacidade da pessoa que a conduz em identificar os problemas presentes nos dados, além da natureza desses problemas, e utilizar os métodos mais apropriados para solucionar cada um dos problemas.

Este trabalho propõe classificar as tarefas realizadas por métodos empregados na fase de pré-processamento em dois grupos:

Tarefas fortemente dependentes de conhecimento de domínio
: Essas tarefas somente podem ser efetivamente realizadas com o uso de conhecimento específico de domínio. Um método automático pode eventualmente ser empregado para realizar uma tarefa fortemente dependente de conhecimento de domínio, entretanto, esse método depende de que um conhecimento específico seja fornecido.

Tarefas fracamente dependentes de conhecimento de domínio
: Essas tarefas podem ser realizadas por métodos que extraem dos próprios dados as informações necessárias para tratar o problema de pré-processamento de dados.

Se por um lado essas tarefas ainda dependem de conhecimento de domínio, pois é necessário, por exemplo, selecionar o método mais adequado para tratar o problema de pré-processamento de dados, por outro lado, essas tarefas podem ser realizadas por métodos com um grau de automação maior do que aquelas que dependem fortemente de conhecimento de domínio.

Pode-se citar como exemplo de tarefa fortemente dependente de domínio a verificação de integridade dos dados. Para realizar tais verificações é freqüentemente necessário conhecer as restrições de valores aplicáveis a cada atributo. Por exemplo, o valor do crédito fornecido por uma empresa financeira a seus clientes deve ser obrigatoriamente um valor positivo. Relações de integridade entre dois ou mais atributos também são comuns. Por exemplo, o valor máximo do seguro de um carro depende do ano e do modelo do carro. Dessa forma, a partir do conhecimento das restrições, é possível utilizar um método automático que encontre problemas de integridade dos dados. Mesmo sendo automático, tal método depende de conhecimento de domínio, ou seja, do conhecimento das restrições de valores, no caso de verificação de integridade. Por esse motivo, essas tarefas são denominadas fortemente dependentes de conhecimento de domínio.

Das tarefas fracamente dependentes de domínio pode-se citar o tratamento de valores desconhecidos, a seleção de atributos, a identificação de valores extremos[4], o tratamento de conjuntos de dados com classes desbalanceadas, entre outras.

Freqüentemente, as tarefas fracamente dependentes de domínio podem ser tratadas com o uso de conhecimento de domínio. Por exemplo, as falhas no processo de aquisição de dados que geram valores desconhecidos para uma determinada aplicação podem ser identificadas e corrigidas, de forma a recuperar os valores ausentes. Entretanto, na ausência de conhecimento de domínio, essas tarefas de pré-processamento podem ser realizadas com o uso de métodos automáticos. De uma forma geral, esses métodos utilizam informações presentes nos dados para tratar o problema de pré-processamento. Os métodos automáticos podem ser de grande valor para aplicações de KDD, nas quais existe um grande volume de dados que dificilmente pode ser tratado manualmente.

[4] *Outliers.*

1.4 Objetivos

A pesquisa realizada neste trabalho tem como objetivo fazer um estudo sobre a fase de pré-processamento de dados do processo de Descoberta de Conhecimento de Bancos de Dados, visando identificar os principais desafios dessa fase.

Sabe-se que a fase de pré-processamento de dados é muito extensa e envolve a identificação e tratamento de diversos tipos de problemas que podem se manifestar nos dados. Dessa forma, é necessário ter um bom ambiente computacional para pré-processamento de dados que ajude na identificação e tratamento dos dados. Esse ambiente computacional deve servir como um *framework* para a implementação de novos métodos de pré-processamento de dados.

Para manter esta pesquisa objetiva, dois problemas de pré-processamento de dados foram escolhidos para serem pesquisados mais detalhadamente: o tratamento de valores desconhecidos e o tratamento de conjuntos de dados com classes desbalanceadas.

O tratamento de valores desconhecidos é um problema conhecido da comunidade de AM e outras áreas de pesquisa que compõem a área de KDD. Entretanto, diversos pesquisadores têm declarado encontrar bases de dados com mais de 50% dos valores ausentes em diversos atributos. Além disso, existe uma preocupação sobre a distribuição dos valores desconhecidos. O tratamento de valores desconhecidos não aleatoriamente distribuídos pode introduzir distorções nos dados, e essas distorções podem ser refletidas no conhecimento extraído.

O tratamento de conjuntos de dados com classes desbalanceadas é um problema recente. Diversos algoritmos utilizados com freqüência em KDD, como os algoritmos que induzem árvores de decisão e regras de decisão, e outros algoritmos, como por exemplo, os algoritmos utilizados no treinamento de redes neurais, possuem dificuldades em aprender na presença de classes desbalanceadas, ou seja, quando existe uma grande diferença no número de exemplos pertencentes a cada classe. Em diversos domínios de aplicação existe uma diferença intrínseca na freqüência em que ocorrem os eventos relacionados a cada classe, e esses domínios acabam por gerar conjuntos de dados desbalanceados. Alguns exemplos são o diagnóstico de doenças raras, a identificação de transações fraudulentas, a identificação de intrusões em sistemas de segurança, entre outros.

Portanto, o tratamento de valores desconhecidos e de conjuntos de dados com classes desbalanceadas são problemas atuais e importantes em KDD. O tratamento de valores desconhecidos, pela necessidade de avaliar os métodos de tratamento de valores desconhe-

cidos propostos pela comunidade de AM no contexto de KDD. O tratamento de conjuntos de dados com classes desbalanceadas, por ser um problema recente e freqüentemente encontrado em dados do mundo real, o qual precisa ser superado para que os algoritmos de AM utilizados em KDD possam ser aplicados nos mais diversos domínios de aplicação.

1.5 Principais Contribuições desta Tese

Uma das principais contribuições deste trabalho é o projeto e implementação de um ambiente para pré-processamento de dados. A esse ambiente foi dado o nome de Discover Learning Environment — DLE. O ambiente DLE é composto por um ambiente computacional para gerenciamento de avaliações experimentais chamado Sniffer, e por uma biblioteca para implementação de métodos de pré-processamento de dados, a qual recebeu o nome de Discover Object Library — DOL. Além disso, foi proposta uma sintaxe para conjuntos de dados que oferece suporte a diversos tipos de dado e à indução construtiva apoiada pelo usuário, entre outras características. A sintaxe recebeu o nome de Discover Dataset Sintax – DSX.

A biblioteca DOL foi implementada para ser uma base sólida para a construção de novos métodos de pré-processamento de dados, enquanto que o ambiente Sniffer é utilizado para avaliar esses novos métodos experimentalmente.

Uma vez que AM e KDD são áreas de pesquisa altamente dinâmicas, nas quais novos métodos e aplicações são propostos a cada dia, as implementações realizadas neste trabalho utilizaram os conceitos de *padrões de projeto*[5] (Shalloway & Trott, 2002; Gamma, Helm, Johnson & Vlissides, 1995) para estarem preparadas para futuras modificações. Tais modificações podem ser de diferentes formas, como a adição de novos sistemas de aprendizado ao ambiente Sniffer, novos tipos de dado à sintaxe DSX, ou novos métodos de pré-processamento de dados à biblioteca DOL, entre outras possibilidades.

A partir das implementações deste trabalho foram projetados e implementados métodos de pré-processamento de dados para os dois problemas eleitos para serem pesquisados neste trabalho: o tratamento de valores desconhecidos e o tratamento de conjuntos de dados com classes desbalanceadas.

Para o tratamento de valores desconhecidos foi pesquisado e avaliado o algoritmo k-vizinhos mais próximos como método de *imputação*[6]. Esse algoritmo foi comparado

[5] *Design patterns.*
[6] *Imputation.*

com outros métodos amplamente utilizados na comunidade, como a IMPUTAÇÃO PELA MÉDIA OU MODA e as estratégias internas utilizadas pelos indutores C4.5 (Quinlan, 1988) e CN2 (Clark & Boswell, 1991) para aprender na presença de valores desconhecidos.

Algumas perguntas podem ser respondidas como resultado da pesquisa realizada sobre tratamento de valores desconhecidos:

1. *Como os valores desconhecidos de um conjunto de dados devem ser tratados?*

   Provavelmente não existe um método que seja ótimo para todos os conjuntos de dados. Dessa forma, cada conjunto de dados deve ser analisado para que se encontre o método mais adequado. De uma forma geral, deve-se evitar utilizar alguns métodos mais simples, como a IMPUTAÇÃO PELA MÉDIA OU MODA, por serem métodos que podem distorcer os dados.

2. *O método de imputação com base no algoritmo* K-VIZINHOS MAIS PRÓXIMOS *são efetivos para o tratamento de valores desconhecidos?*

   Nos experimentos realizados, o método de imputação com base no algoritmo K-VIZINHOS MAIS PRÓXIMOS obteve resultados que foram, na maioria das vezes, superiores aos demais métodos analisados.

3. *Os métodos de tratamento são efetivos mesmo com grandes quantidades de valores desconhecidos?*

   Nos experimentos realizados, os métodos de tratamento de valores desconhecidos obtiveram bons resultados mesmo com 50% ou 60% de valores desconhecidos. Na realidade, com freqüência, as taxas de erro obtidas com grandes proporções de valores desconhecidos foram levemente superiores e, em alguns casos, inferiores às taxas de erro obtidas com os dados completos. Entretanto, esse fato pode decorrer dos valores desconhecidos terem sido inseridos de forma aleatória.

4. *Quais são as limitações dos métodos de imputação?*

   Os métodos de imputação normalmente predizem valores mais bem comportados do que os valores reais (não conhecidos) seriam. Dessa forma, os classificadores induzidos tendem a se tornar mais simples quanto maior for a quantidade de valores desconhecidos tratados. Esse fato pode levar ao risco de simplificar excessivamente o problema que está sendo estudado.

   Sendo os valores imputados aproximações dos valores reais deve-se, antes de utilizar um método de imputação, procurar verificar se não é possível coletar os dados

ausentes ou, até mesmo, verificar se não existe um outro atributo com informações similares, isto é, alta correlação, no conjunto de dados. Nos experimentos realizados, a presença de um ou mais atributos com alta correlação com os atributos com valores desconhecidos fez com que o indutor C4.5 obtivesse, sem tratamento dos valores desconhecidos, bons resultados, freqüentemente superiores aos obtidos pelos métodos de imputação.

Para o tratamento de conjuntos de dados com classes desbalanceadas, foi pesquisado e utilizado o método de *seleção unilateral*. A seleção unilateral é um método de *under-sampling*, ou seja, um método que reduz o número de exemplos da classe majoritária com o objetivo de melhorar o balanceamento das classes e, conseqüentemente, melhorar a classificação da classe minoritária.

Algumas perguntas podem ser respondidas como resultado da pesquisa realizada sobre tratamento de conjuntos de dados com classes desbalanceadas:

1. *Como os conjuntos de dados com classes desbalanceadas devem ser tratados?*

   Diversos métodos têm sido propostos para solucionar o problema de aprender com conjuntos de dados com classes desbalanceadas. Uma forma bastante direta de tratar esse problema com métodos de pré-processamento de dados é balancear artificialmente as classes.

2. *O método de seleção unilateral é efetivo para melhorar o desempenho de classificação da classe minoritária?*

   Nos experimentos realizados, o método de seleção unilateral obteve bons resultados. O método de seleção unilateral reduziu a taxa de falso negativo, ou seja, o número de exemplo da classe minoritária classificados incorretamente, para menos da metade da taxa de falso negativo obtida com o treinamento realizado com todos os exemplos.

3. *Quais são as limitações dos métodos de tratamento de conjuntos com classes desbalanceadas?*

   Para a maioria dos problemas reais existe uma relação de perda e ganho entre as taxas de falso positivo e falso negativo. Dessa forma, uma redução na taxa de falso negativo pode ser acompanhada de um aumento da taxa de falso positivo. Nesse caso, é necessário verificar se houve uma redução no custo total de classificação incorreta.

1.6 Organização deste Trabalho

Esta tese está organizada da seguinte forma:

Capítulo 2: Aprendizado de Máquina

Nesse capítulo é feito um estudo sobre o aprendizado em Inteligência Artificial, com ênfase no aprendizado indutivo. É introduzida, também, a nomenclatura que é utilizada no decorrer deste trabalho. Por fim, é apresentado o projeto DISCOVER, do qual o ambiente DLE, implementado neste trabalho, é parte integrante;

Capítulo 3: Pré-processamento de Dados

Nesse capítulo é realizado um estudo sobre os desafios que podem ser encontrados na fase de pré-processamento de dados do processo de KDD. O processo de KDD é introduzido e cada uma de suas fases é comentada. As fases de coleta de dados e transformação de dados são analisadas mais detalhadamente, por estarem mais diretamente relacionadas com a fase de pré-processamento de dados.

Capítulo 4: O Ambiente DISCOVER LEARNING ENVIRONMENT — DLE

O ambiente computacional DISCOVER LEARNING ENVIRONMENT — DLE é apresentado nesse capítulo. São discutidos o projeto, a arquitetura e a implementação tanto da biblioteca DISCOVER OBJECT LIBRARY — DOL, quanto do ambiente computacional para gerenciamento de experimentos SNIFFER.

Capítulo 5: Tratamento de Valores Desconhecidos

O tratamento de valores desconhecidos é analisado nesse capítulo. É realizado um estudo que discute os principais métodos de tratamento de valores desconhecidos utilizados na literatura. São realizados, também, experimentos envolvendo diversos métodos de tratamento de valores desconhecidos, incluindo o método de imputação com o algoritmo K-VIZINHOS MAIS PRÓXIMOS.

Capítulo 6: Aprendizado com Classes Desbalanceadas

Nesse capítulo é feito um estudo sobre o problema de aprender quando um conjunto de dados possui uma grande diferença no número de exemplos pertencentes à cada classe. É discutido e avaliado experimentalmente o método de *seleção unilateral*, utilizado para reduzir o número de casos da classe majoritária.

Capítulo 7: Conclusão

Nesse capítulo são apresentadas as conclusões deste trabalho e propostas para trabalhos futuros.

Apêndice A: A Sintaxe DISCOVER DATASET SINTAX

A sintaxe DISCOVER DATASET SINTAX — DSX — utilizada como sintaxe padrão

para conjuntos de dados no ambiente DLE, é apresentada em detalhes nesse apêndice. Essa sintaxe permite declarar arquivos com a descrição dos dados e atributos. A sintaxe DSX possui diversos recursos e pode ser utilizada em conjunto com a biblioteca DOL.

Apêndice B: Relatórios do Ambiente Sniffer

Nesse apêndice são apresentados exemplos de diversos relatórios gerados pelo ambiente Sniffer, o qual é parte integrante do ambiente DLE.

Capítulo 2

Aprendizado de Máquina

2.1 Considerações Iniciais

Neste capítulo são apresentados alguns conceitos introdutórios sobre aprendizado em Inteligência Artificial e, mais especificamente, sobre Aprendizado de Máquina. É dado ênfase ao aprendizado indutivo por exemplos que consiste em aprender conceitos a partir de exemplos e contra-exemplos desses conceitos.

Este capítulo está organizado da seguinte forma: na Seção 2.2 é introduzido o aprendizado em Inteligência Artificial; na Seção 2.3 são apresentados os conceitos básicos de aprendizado indutivo e, logo em seguida, na Seção 2.4 os conceitos de aprendizado indutivo por exemplos; na Seção 2.5 é descrito o Aprendizado de Máquina Indutivo por Exemplos, que é um dos focos deste trabalho. Nessa seção também são introduzidas algumas definições e a notação que é utilizada nos próximos capítulos; na Seção 2.6 é discutido brevemente o processo de Descoberta de Conhecimento em Bancos de Dados. Esse tema volta a ser objeto de estudo no Capítulo 3; na Seção 2.7 é apresentado o projeto DISCOVER que visa dar suporte a diversas etapas do processo de Descoberta de Conhecimento de Bancos de Dados; por fim, na Seção 2.8 são apresentadas as considerações finais a respeito deste capítulo.

2.2 O Aprendizado em Inteligência Artificial

Aprendizado de Máquina — AM — é uma sub-área de pesquisa muito importante em Inteligência Artificial — IA — pois a capacidade de aprender é essencial para um compor-

tamento inteligente. AM estuda métodos computacionais para adquirir novos conhecimentos, novas habilidades e novos meios de organizar o conhecimento já existente (Mitchell, 1997). O estudo de técnicas de aprendizado baseado em computador também pode fornecer um melhor entendimento de nosso próprio processo de raciocínio (Monard, Batista, Kawamoto & Pugliesi, 1997).

Uma das críticas mais comuns à IA é que as máquinas só podem ser consideradas inteligentes quando forem capazes de aprender novos conceitos e se adaptarem a novas situações, em vez de simplesmente fazer o que lhes for mandado. Não há muita dúvida de que uma importante característica das entidades inteligentes é a capacidade de adaptar-se a novos ambientes e de resolver novos problemas. É possível incorporar tais habilidades em programas? Ada Augusta, uma das primeiras filósofas em computação, escreveu

> *A Máquina Analítica*[1] *não tem qualquer pretensão de* originar *nada. Ela pode fazer qualquer coisa desde que nós* saibamos como mandá-la *executar.*

Esse comentário foi interpretado por vários críticos de IA como uma indicação de que os computadores não são capazes de aprender. Entretanto, nada impede que digamos a um computador como interpretar as informações recebidas, de uma maneira que melhore gradualmente seu desempenho.

Como veremos nas próximas seções, sob algumas restrições, é possível criar um sistema computacional que seja capaz de aprender e melhorar o seu desempenho por meio da observação. Existem várias abordagens de aprendizado que podem ser utilizadas por um sistema computacional como, por exemplo, o aprendizado por *hábito*, por *instrução*, por *dedução*, por *analogia* e por *indução*. O aprendizado indutivo é um dos mais úteis pois permite obter novos conhecimentos a partir de exemplos, ou casos, particulares previamente observados. Entretanto, o aprendizado indutivo é também um dos mais desafiadores, pois o conhecimento gerado ultrapassa os limites das premissas, e não existem garantias de que esse conhecimento seja verdadeiro.

2.3 Aprendizado Indutivo

Indução é a forma de inferência lógica que permite que conclusões gerais sejam obtidas de exemplos particulares. É caracterizada como o raciocínio que parte do específico para

[1] *Analytical Engine.*

o geral, do particular para o universal, da parte para o todo. Hipóteses geradas pela inferência indutiva podem ou não preservar a verdade, ou seja, as hipóteses levam a conclusões cujos conteúdos excedem os das premissas. É esse traço característico da indução que torna os argumentos indutivos indispensáveis para a fundamentação de uma significativa porção dos nossos conhecimentos. Entretanto, é esse mesmo fato que levanta questões extremamente complicadas, dificultando a análise dos resultados obtidos com auxílio de métodos indutivos.

Ao contrário do que sucede com um argumento dedutivo e válido, um argumento indutivo e correto pode, perfeitamente, admitir uma conclusão falsa, ainda que suas premissas sejam verdadeiras. Mesmo não podendo garantir que a conclusão de um argumento seja verdadeira quando as premissas são verdadeiras, pode-se afirmar que as premissas de um argumento indutivo correto sustentam ou atribuem certa verossimilhança à sua conclusão. Quando as premissas de um argumento indutivo são verdadeiras, o melhor que pode ser dito é que a sua conclusão é *provavelmente* verdadeira. Uma exceção disso é a indução matemática. Em um argumento matemático indutivo correto, partindo de premissas verdadeiras obtém-se, invariavelmente, conclusões verdadeiras.

Há certos enganos que podem tornar os argumentos indutivos completamente inúteis ou inúteis de um ponto de vista prático. Enganos desse gênero são denominados *falácias indutivas*. Quando um argumento indutivo é falaz, as premissas não sustentam a conclusão. Entre os argumentos indutivos corretos, porém, pode-se cogitar um grau de sustentação ou de apoio. As premissas de um argumento indutivo correto podem tornar a conclusão extremamente provável, moderadamente provável ou provável com certo grau de certeza. Por exemplo, imagine a seguinte declaração

"Ninguém gosta de óleo de fígado de bacalhau."

Essa declaração é tipicamente resultado de um raciocínio indutivo. Para provar que ela é falsa, basta encontrar uma única pessoa que goste de óleo de fígado de bacalhau. Na realidade, uma declaração, ou hipótese, mais sustentável que poderia ser feita a respeito desse assunto seria

"Ninguém que eu conheço gosta de óleo de fígado de bacalhau."

Entretanto, pode-se tornar essa declaração mais ou menos provável, dependendo das premissas que a suportam. Por exemplo, imagine as seguintes premissas

"Eu entrevistei 5 pessoas e nenhuma delas gosta de óleo de fígado de bacalhau."

"Eu entrevistei 100 pessoas e todas elas declararam não gostar de óleo de fígado de bacalhau."

"Eu entrevistei 100.000 pessoas e não consegui encontrar uma única que gostasse de óleo de fígado de bacalhau."

Todas essas premissas sustentam a hipótese das pessoas não gostarem de óleo de fígado de bacalhau, mas cada uma delas com um grau de sustentação diferente.

Há uma segunda diferença entre os argumentos indutivos e os dedutivos. Dado um argumento dedutivo válido, é possível acrescentar novas premissas, colocando-as com as já existentes, sem afetar a validade do argumento. Em contraste, o grau de sustentação que as premissas de um argumento indutivo conferem à conclusão pode ser alterado por evidências adicionais, acrescentadas ao argumento sob a forma de novas premissas que figurem ao lado das premissas inicialmente consideradas. Como a conclusão de um argumento indutivo pode ser falsa mesmo quando as premissas forem verdadeiras, a evidência adicional, admitindo-se que seja relevante, pode nos capacitar a determinar, com maior precisão, se a conclusão é verdadeira. A evidência adicional pode afetar o grau de sustentação da conclusão.

A inferência indutiva é um dos principais meios de criar novos conhecimentos e prever eventos futuros. O processo de indução é indispensável na obtenção de novos conhecimentos pelo ser humano. Foi por meio de induções que Kepler descobriu as leis do movimento planetário, que Mendel descobriu as leis da genética e que Arquimedes descobriu o princípio da alavanca. Pode-se ousar em afirmar que a indução é o recurso mais utilizado pelos seres humanos para obter novos conhecimentos. Apesar disso, esse recurso deve ser utilizado com os devidos cuidados, pois se o número de observações for insuficiente ou se os dados relevantes forem mal escolhidos, as hipóteses induzidas podem ser de pouco ou nenhum valor.

2.4 Aprendizado Indutivo por Exemplos

Como já mencionado, aprendizado indutivo é o processo de inferência indutiva realizada sobre fatos, situações ou casos observados, os quais são fornecidos ao aprendiz por um professor ou oráculo. Um tipo especial de aprendizado indutivo é o *aprendizado indutivo por exemplos*, cuja tarefa é induzir descrições gerais de conceitos utilizando exemplos

específicos desses conceitos (Michalski, Carbonell & Mitchell, 1983).

Para introduzir o aprendizado indutivo por exemplos de forma informal, imagine uma tarefa de aprendizado na qual se deseja aprender a diferenciar seres humanos de outros animais, com base em apenas duas características: altura e peso. Pode-se então medir a altura e o peso de diversos animais e rotular cada um desses casos como humanos e não humanos. Por simplicidade, é utilizado o símbolo + para identificar os seres humanos e, diz-se que esses são *exemplos positivos* de seres humanos; e o símbolo − para identificar os exemplos de outros animais, esses exemplos são utilizados como *exemplos negativos* ou *contra-exemplos* de seres humanos.

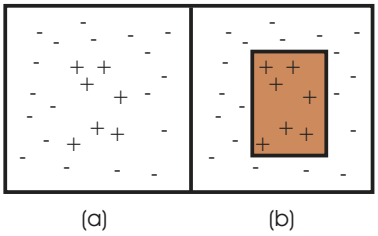

Figura 2.1: Representação gráfica de um conjunto de exemplos *(a)* e uma possível hipótese para o conceito representado por esses exemplos *(b)*.

Na Figura 2.1-*a* é ilustrada essa situação na qual os exemplos positivos e negativos estão dispostos conforme os seus respectivos valores de altura e peso. Torna-se necessário responder a seguinte pergunta

> *É possível aprender um conceito que diferencie os seres humanos dos demais animais com base apenas nas características e nos exemplos fornecidos?*

Uma possível hipótese para o conceito representado pelos exemplos está ilustrada na Figura 2.1-*b*. O retângulo restringe os valores das características de forma que somente algumas combinações de altura e peso levem a concluir que se trata de um ser humano. Os exemplos internos ao retângulo são todos positivos e os externos negativos, dessa forma diz-se que essa hipótese é *consistente* com os exemplos dados, pois os separa perfeitamente sem cometer enganos.

Pode-se formalizar o problema de aprendizado de conceitos utilizando exemplos da seguinte forma (Bratko, 1990):

Definição 2.1 *Seja $\mathcal{U}$ o conjunto universal dos objetos, isto é, todos os objetos que o aprendiz pode encontrar. Não existe limite, a princípio, para a cardinalidade de $\mathcal{U}$. Um conceito $\mathcal{C}$ pode ser formalizado como sendo um subconjunto de objetos em $\mathcal{U}$, assim*

$$\mathcal{C} \subset \mathcal{U}$$

Aprender um conceito $\mathcal{C}$ significa aprender a reconhecer objetos em $\mathcal{C}$. Ou seja, uma vez que o conceito $\mathcal{C}$ é aprendido, para qualquer objeto $\boldsymbol{x} \in \mathcal{U}$, o sistema é capaz de reconhecer se $\boldsymbol{x} \in \mathcal{C}$.

É importante notar que, pela Definição 2.1, o conceito aprendido deve ser útil não apenas para reconhecer corretamente os exemplos utilizados para aprender o conceito $\mathcal{C}$, mas também para reconhecer corretamente se qualquer outro exemplo pertence ou não ao conceito aprendido.

A inferência indutiva e a estrutura básica para guiar a busca em aprendizado indutivo são descritas em Shaw & Gentry (1990) da seguinte forma

> *... inferência indutiva é um processo de solução de problemas que obtém soluções — descrições do conceito induzido — por meio de busca e de uma seqüência de transformações. Generalização e especialização são passos essenciais quando se faz inferência indutiva. Se a descrição do conceito $\mathcal{Q}$ é mais geral que a descrição do conceito $\mathcal{P}$, a transformação de $\mathcal{P}$ para $\mathcal{Q}$ é chamada generalização, e a transformação de $\mathcal{Q}$ para $\mathcal{P}$ é chamada especialização. $\mathcal{P}$ é dito ser mais geral que $\mathcal{Q}$ se (e somente se) $\mathcal{P}$* cobre[2] *mais exemplos que $\mathcal{Q}$. Inferência indutiva pode ser vista como um processo que faz iterações sucessivas de generalização e especialização nas descrições do conceito, e é consistente com todos os exemplos. Então relações generalização/especialização entre descrições de um conceito fornecem a estrutura básica para guiar a busca em aprendizado indutivo.*

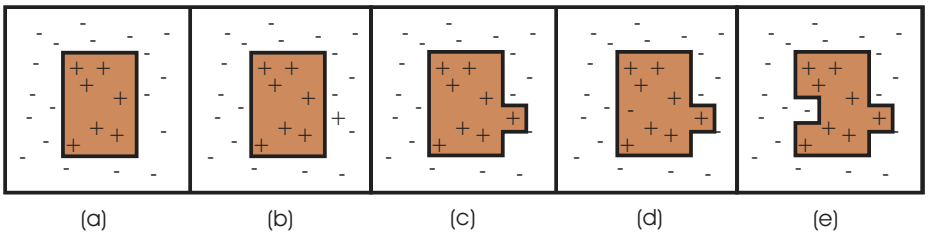

Figura 2.2: Atualização de uma hipótese. Hipótese consistente *(a)*. Falso negativo *(b)*. Hipótese generalizada *(c)*. Falso positivo *(d)*. Hipótese especializada *(e)*.

[2]Um conceito ou uma hipótese cobre um exemplo quando reconhece o exemplo como pertencente ao conceito.

As relações de generalização e especialização podem ser melhor entendidas por meio de um exemplo (Russel & Norvig, 2003). Imagine se a hipótese apresentada na Figura 2.1 na página 15 precisasse ser atualizada para se tornar consistente com novos exemplos. A Figura 2.2 na página anterior ilustra esse processo de atualização de uma hipótese. A Figura 2.2-*a* apresenta uma hipótese consistente com todos os exemplos. Essa hipótese é capaz de separar corretamente todos os exemplos rotulados como + dos exemplos rotulados como −. Na Figura 2.2-*b* um exemplo *falso negativo* é adicionado. Esse exemplo é chamado de *falso negativo*, pois a hipótese classifica esse exemplo como negativo, mas na realidade ele é positivo. Nessa situação, pode-se considerar que a hipótese é muito especializada, assim, é necessário generalizá-la para incluir esse novo exemplo. A Figura 2.2-*c* ilustra uma possível generalização para a hipótese. Na Figura 2.2-*d* é adicionado mais um exemplo, esse novo exemplo é classificado como positivo pela hipótese, mas na realidade ele é negativo, portanto ele é chamado de *falso positivo*. A presença desse exemplo indica que a hipótese é muito genérica e precisa ser especializada. A Figura 2.2-*e* ilustra uma possível especialização para a hipótese.

Por meio de generalizações e especializações é possível criar um algoritmo que induz uma hipótese de um conceito consistente com todos os exemplos. Esse algoritmo inicia induzindo uma hipótese consistente para um único exemplo positivo e adiciona novos exemplos, procurando manter a consistência da hipótese com os novos exemplos. Esse algoritmo foi primeiramente definido por Mill (1943) e é apresentado no Algoritmo 2.1 na página seguinte.

Diversos sistemas de AM utilizam operações de generalização e especialização para criar hipóteses a partir de exemplos. Em especial, os algoritmos capazes de representar a hipótese do conceito a ser aprendido utilizando como linguagem de representação *regras de decisão* (Fürnkranz, 1999; Monard & Baranauskas, 2003b). Regras podem ser especializadas/generalizadas por meio da inclusão/remoção de condições no antecedente da regra.

Na prática, os desafios de aprender um conceito por meio da indução a partir de exemplos são muito maiores. Relembrando a Definição 2.1 na página anterior, uma hipótese somente pode ser útil se puder ser utilizada para reconhecer corretamente exemplos além dos utilizados na indução da hipótese. Portanto, deve-se ter cuidado ao induzir uma hipótese para que ela não seja excessivamente especializada aos exemplos utilizados para criá-la, um problema chamado de *overfitting* ou *overspecialization*[3].

[3]Esses termos são utilizados em inglês por serem amplamente difundidos na comunidade.

Algoritmo 2.1 Algoritmo que procura por uma hipótese consistente com os exemplos por meio de operações de generalização e especialização.

Require: $E = \{E_1, E_2, \ldots E_N\}$, um conjunto de exemplos e contra-exemplos do conceito a ser aprendido
Ensure: **h** = uma hipótese do conceito consistente com todos os exemplos em E
  h $\leftarrow$ uma hipótese consistente com um exemplo positivo qualquer $E_i \in E$
  $V \leftarrow \emptyset$
  for all $E_i \in E$ **do**
    $V \leftarrow V \bigcup \{E_i\}$
    if E_i é um falso positivo para **h then**
      h $\leftarrow$ **escolha** uma especialização de **h** consistente com V
    else if E_i é um falso negativo para **h then**
      h $\leftarrow$ **escolha** uma generalização de **h** consistente com V
    end if
    if nenhuma especialização/generalização consistente pode ser encontrada **then**
      fail
    end if
  end for
  return h

Diretamente relacionado ao problema de *overfitting* está o problema de aprender um conceito mesmo quando os dados possuem *ruído*, ou seja, mesmo quando alguns dados são incorretos. Na presença de ruído, aprender uma hipótese consistente com todos os exemplos pode fazer com que a hipótese se especialize em alguns dados incorretos, e seja de pouco proveito para outros exemplos que não foram utilizados na indução. Nessas situações é preferível induzir uma hipótese mais simples e não consistente com todos os dados, mas que seja mais útil para exemplos ainda não vistos.

2.5 Aprendizado de Máquina Indutivo por Exemplos

Como mencionado anteriormente, o aprendizado indutivo é efetuado a partir do raciocínio sobre exemplos fornecidos por um processo externo ao aprendiz. Em AM, o aprendiz é um sistema computacional freqüentemente denotado por *sistema de aprendizado*, *algoritmo de aprendizado*, ou simplesmente *indutor*. Um sistema de aprendizado é um sistema computacional que toma decisões baseado em experiências acumuladas contidas em casos resolvidos com sucesso (Weiss & Kulikowski, 1991).

O aprendizado indutivo por exemplos pode ser dividido em *aprendizado supervisionado* e *não supervisionado*, descritos a seguir (Monard & Baranauskas, 2003a).

Aprendizado supervisionado

No aprendizado supervisionado é fornecido ao sistema de aprendizado um conjunto de exemplos $E = \{E_1, E_2, \ldots E_N\}$, sendo que cada exemplo $E_i \in E$ possui um rótulo associado. Esse rótulo define a *classe* a qual o exemplo pertence. Um pouco mais formalmente, pode-se dizer que cada exemplo $E_i \in E$ é uma tupla

$$E_i = (\vec{x_i}, y_i) \tag{2.1}$$

na qual $\vec{x_i}$ é um vetor de valores que representam as características, ou *atributos*, do exemplo E_i, e y_i é o valor da classe desse exemplo. O objetivo do aprendizado supervisionado é induzir um mapeamento geral dos vetores $\vec{x}$ para valores y. Portanto, o sistema de aprendizado deve construir um modelo, $y = f(\vec{x})$, de uma função desconhecida, f, também chamada de *função conceito*[4], que permite predizer valores y para exemplos previamente não vistos.

Entretanto, o número de exemplos utilizados para a criação do modelo não é, na maioria dos casos, suficiente para caracterizar completamente essa função f. Na realidade, os sistemas de aprendizado são capazes de induzir uma função $\mathbf{h}$ que aproxima f, ou seja, $\mathbf{h}(\vec{x}) \approx f(\vec{x})$. Nesse caso, $\mathbf{h}$ é chamada de *hipótese* sobre a função conceito f.

Aprendizado não supervisionado

No aprendizado não supervisionado é fornecido ao sistema de aprendizado um conjunto de exemplos E, no qual cada exemplo consiste somente de vetores $\vec{x}$, não incluindo a informação sobre a classe y. O objetivo no aprendizado não supervisionado é construir um modelo que procura por regularidades nos exemplos, formando agrupamentos ou *clusters* de exemplos com características similares.

Assim, um *conjunto de exemplos* ou *conjunto de dados* $E = \{E_1, E_2, \ldots E_N\}$ é um conjunto de vetores $\vec{x_1}, \ldots \vec{x_N}$, com ou sem a classe associada y. A Tabela 2.1 na página seguinte mostra a forma geral de um conjunto de exemplos E com N exemplos e M atributos. Essa tabela está no formato *atributo-valor*[5], o qual é utilizado como entrada pela maioria dos algoritmos de aprendizado. Na forma atributo-valor as colunas (A_1, $\ldots A_M$) da tabela representam os diferentes atributos, e as linhas ($E_1, \ldots E_N$) os diferentes exemplos. Assim, a linha i na Tabela 2.1 refere-se ao *i-ésimo* exemplo e a entrada x_{ij} refere-se ao valor do *j-ésimo* atributo A_j do exemplo i, ou seja, $\vec{x_i} = (x_{i1}, x_{i2}, \ldots x_{iM})$.

[4] *Concept function.*
[5] Também chamada de tabela *flat*.

O atributo Y é o atributo que assume os valores da classe de cada exemplo E_i. Esse atributo é freqüentemente chamado de *atributo classe* e, como já mencionado, somente está presente em conjuntos de dados para aprendizado supervisionado.

	A_1	A_2	$\cdots$	A_M	Y
E_1	x_{11}	x_{12}	$\cdots$	x_{1M}	y_1
E_2	x_{12}	x_{22}	$\cdots$	x_{2M}	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
E_N	x_{N1}	x_{N2}	$\ldots$	x_{NM}	y_N

Tabela 2.1: Conjunto de exemplos no formato atributo-valor.

Em aprendizado supervisionado, o atributo classe Y pode ser um atributo qualitativo que assume um conjunto de valores discretos $C = \{C_1, C_2, \ldots C_{Ncl}\}$ ou um atributo quantitativo que assume um conjunto de valores reais.

No primeiro caso, assumindo que os vetores $\vec{x}$ correspondem a pontos em um espaço M-dimensional $\Re^M$, o objetivo do aprendizado é encontrar uma função $\mathbf{h}$ que aproxima a função $f : \Re^M \rightarrow C$. Nesse caso, a hipótese $\mathbf{h}$ é denominada *classificador*, e a tarefa de aprendizado é denotada *classificação*.

No segundo caso, o atributo classe é quantitativo, o qual pode assumir um conjunto de valores reais. O objetivo do aprendizado é encontrar uma função $\mathbf{h}$ que aproxima a função $f : \Re^M \rightarrow \Re$. Nesse caso, a hipótese $\mathbf{h}$ é denominada *regressor*, e a tarefa de aprendizado é denotada *regressão*.

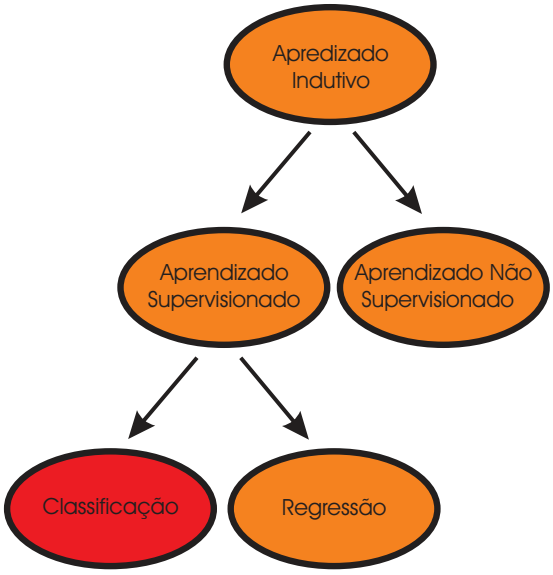

Figura 2.3: A hierarquia do aprendizado.

Na Figura 2.3 na página anterior é mostrada a hierarquia do aprendizado indutivo. O foco deste trabalho é o aprendizado supervisionado. Uma ênfase maior é dada aos problemas de classificação, embora alguns métodos propostos neste trabalho possam também ser utilizados para problemas de regressão.

2.5.1 Os Paradigmas de Aprendizado de Máquina Supervisionado

Dentro da área de AM foram propostos vários paradigmas capazes de aprender a partir de um conjunto de exemplos. Um requisito básico para todos os paradigmas de AM supervisionado é que o conceito a ser aprendido deve estar relacionado com casos observados, isto é, exemplos, e cada exemplo deve estar rotulado com a classe a qual pertence. Os paradigmas de aprendizado mais conhecidos são brevemente discutidos nas próximas seções.

2.5.1.1 Paradigma Simbólico

Os sistemas de aprendizado simbólico buscam aprender construindo representações simbólicas de um conceito por meio da análise de exemplos e contra-exemplos desse conceito. As representações simbólicas estão tipicamente na forma de alguma expressão lógica: árvores de decisão; regras de decisão ou redes semânticas.

Atualmente, entre as representações simbólicas mais estudadas estão as árvores e regras de decisão. É atribuído a Morgan & Messenger (1973) o desenvolvimento original do programa para a indução de árvores de decisão. O método de indução de árvores de decisão a partir de dados empíricos, conhecido como *particionamento recursivo*, foi estudado por pesquisadores da área de IA e Estatística. Os sistemas ID3 (Quinlan, 1986) e C4.5 (Quinlan, 1987b) para indução de árvores de decisão tiveram uma importante contribuição sobre a pesquisa em IA. É interessante observar que sistemas de árvores de classificação e regressão (Breiman, Friedman, Olshen & Stone, 1984) foram desenvolvidos independentemente por estatísticos durante praticamente o mesmo período que o ID3, no final dos anos 70.

Os trabalhos com indução de regras de decisão surgiram com a simples tradução das árvores de decisão para regras, com a poda realizada sobre as regras, tal abordagem surgiu no trabalho de (Quinlan, 1987a). Posteriormente, foram criados métodos que induziam regras diretamente a partir dos dados, um exemplo desse trabalho pode ser encontrado

em (Michalski, Mozetic, Hong & Lavrac, 1986). Um excelente levantamento dos principais sistemas indutores de regras de decisão pode ser encontrado em (Fürnkranz, 1999).

2.5.1.2 Paradigma Estatístico

Pesquisadores em estatística têm criado diversos métodos de classificação e regressão, muitos deles semelhantes aos métodos empregados em AM. Por exemplo, o método CART (Breiman, Friedman, Olshen & Stone, 1984), um sistema muito conhecido para construir árvores de decisão, foi desenvolvido por estatísticos. Como regra geral, técnicas estatísticas tendem a focar tarefas em que todos os atributos têm valores contínuos ou ordinais. Muitos deles também são paramétricos, assumindo alguma forma de modelo ou distribuição, e então encontrando valores apropriados para os parâmetros do modelo a partir de dados. Por exemplo, um classificador linear assume que as classes podem ser expressas como combinação linear dos valores dos atributos, e então procura uma combinação linear particular que forneça a melhor aproximação sobre o conjunto de dados. Os classificadores estatísticos freqüentemente assumem que os valores de atributos estão normalmente distribuídos, e então usam os dados fornecidos para determinar média, variância e co-variância da distribuição.

2.5.1.3 Paradigma *Instance-based*

Uma forma de classificar um caso é lembrar de um caso similar cuja classe é conhecida e assumir que o novo caso terá a mesma classe. Essa filosofia exemplifica os sistemas *instance-based*, os quais classificam casos nunca vistos utilizando casos similares conhecidos (Aha, Kibler & Albert, 1991).

As características principais dos sistemas *instanced-based* são:

Os casos de treinamento devem ser lembrados

Se todos os casos forem memorizados, o classificador pode se tornar lento e difícil de manusear. O ideal é reter casos prototípicos que juntos resumam toda a informação importante. Essa abordagem pode ser observada em livros médicos e legais. Aha, Kibler & Albert (1991) descrevem algumas estratégias para decidir quando um novo caso deve ser memorizado. Uma segunda solução reside em construir estruturas capazes de indexar os exemplos e responder consultas sobre os exemplos mais semelhantes de forma mais rápida. Exemplos dessas estruturas são as M-trees (Ciaccia, Patella & Zezula, 1997) e as Slim-trees (Jr., Traina, Seeger & Faloutsos, 2000);

A medida de similaridade entre os casos

Se todos os atributos forem quantitativos, pode-se calcular a distância entre dois casos utilizando a distância euclidiana, por exemplo. Quando alguns atributos não são qualitativos, essa interpretação de distância se torna mais problemática. Além do mais, se existem muitos atributos irrelevantes, dois casos similares podem aparentar serem muito diferentes pois eles podem possuir valores diferentes em atributos sem importância. Stanfill & Waltz (1986) desenvolveram um método sensível ao contexto para alterar a escala dos atributos de forma que as medidas de distância fiquem mais robustas;

A relação entre um novo caso e os casos armazenados

Para classificar um novo caso existem diversas alternativas. Uma delas consiste em usar um único caso, o qual é o mais próximo do novo caso para classificar esse novo caso. Uma segunda alternativa consiste em usar vários casos. Nessa alternativa, pode-se levar em consideração os diferentes graus de similaridade entre cada caso e o novo caso na determinação da classe do novo caso. A segunda alternativa é geralmente mais utilizada por ser mais robusta a erros nos dados.

2.5.1.4 Paradigma Conexionista

Redes neurais são construções matemáticas relativamente simples que foram inspiradas no modelo biológico do sistema nervoso. Sua representação envolve unidades altamente inter-conectadas, e o nome *conexionismo* é utilizado para descrever a área de estudo.

A metáfora biológica com as conexões neurais do sistema nervoso tem interessado muitos pesquisadores, e tem fornecido diversas discussões sobre os méritos e as limitações dessa abordagem de aprendizado. Em particular, as analogias com a biologia têm levado muitos pesquisadores a acreditar que as redes neurais possuem um grande potencial na resolução de problemas que requerem intenso processamento sensorial humano, tal como visão e reconhecimento de voz.

As pesquisas em redes neurais foram iniciadas com o trabalho pioneiro de McCulloch & Pitts (1943). McCulloch era um psiquiatra e pesquisou por 20 anos uma forma de representar um evento no sistema nervoso. Pitts era um jovem pesquisador e começou a trabalhar com McCulloch em 1942. Praticamente 15 anos após a publicação de McCulloch e Pitts, Rosenblatt (1958) apresentou o *Perceptron*, cuja grande contribuição foi a prova do teorema de convergência. Mas, no livro *Perceptrons*, Minsky & Papert (1969) demonstraram a existência de limites fundamentais nos perceptrons de uma camada. A pesquina

na área ficou praticamente estática até que Hopfield (1982) utilizou a idéia de uma função de energia para formular uma nova forma de compreender os cálculos realizados em redes recorrentes com conexões sinápticas simétricas.

Talvez mais do que qualquer outra publicação, o artigo de Hopfield em 1982 e o livro de Rumelhart & McClelland (1998), foram as publicações que mais influenciaram para o resurgimento do interesse sobre redes neurais na década de 80.

2.6 Descoberta de Conhecimento em Bancos de Dados

Tradicionalmente, o processo de análise de dados para a identificação de informações úteis baseia-se em um trabalho manual. Por exemplo, especialistas em concessão de crédito de um banco podem periodicamente verificar determinadas estatísticas nos dados, como por exemplo, a média de idade dos clientes que conseguiram e dos que não conseguiram pagar um empréstimo, a inadimplência no período atual e no mesmo período dos anos anteriores, entre outras. A partir dessas análises, os especialistas podem gerar relatórios que serão utilizados para tomar futuras decisões de concessão de crédito para novos clientes.

Em geral, o método clássico de análise de dados reside em um ou mais analistas humanos familiares com os dados e atuando como uma interface entre os dados e o usuário (Baranauskas, 2001). Nota-se, facilmente, que a forma manual de análise de dados é lenta, cara e altamente subjetiva. É comum que tais dados residam em uma base de dados digital, facilitando a extração de muitos relatórios e consultas. Mas, ainda assim, o analista humano deve comparar relatórios, cruzar informações e utilizar seu conhecimento prévio sobre a área de atuação para tentar extrair *algo novo* dos dados existentes.

Na medida em que as bases de dados atuais crescem cada vez mais, essa abordagem torna-se impraticável em vários domínios, principalmente na descoberta de informações úteis. Bases de dados contendo registros da ordem de 10^9 exemplos estão se tornando cada vez mais comuns. De forma similar, o número de atributos dos registros pode facilmente atingir a ordem de 10^2 ou mesmo 10^3 (Teller & Veloso, 1995). É muito improvável que um ser humano consiga analisar e inferir informações em bases de dados com essas dimensões.

Existe, portanto, a necessidade por uma nova geração de técnicas e ferramentas com a habilidade de assistir os analistas humanos de uma forma "inteligente" e automática na procura de informações úteis previamente desconhecidas nos dados. Tais técnicas e

ferramentas são objeto de estudo de um nova área de pesquisa chamada de *Descoberta de Conhecimento em Bancos de Dados* – KDD.

KDD é um processo que engloba diversas fases, as quais são apresentadas em mais detalhes no Capítulo 3. Normalmente, o processo de KDD inicia com um estudo do problema a ser tratado, passa por uma identificação dos dados que serão analisados, posteriormente esses dados são coletados do banco de dados, pré-processados e transformados. Por fim, esses dados são utilizados na fase do processo de KDD, denominada *Mineração de Dados* — MD — para extrair conhecimento que é posteriormente pós-processado e analisado. Nessa fase de Mineração de Dados são utilizados algoritmos capazes de receber dados como entrada e extrair conhecimento desses dados. Freqüentemente, mas não obrigatoriamente, esses algoritmos de extração de conhecimento são algoritmos de AM indutivo por exemplos. Este trabalho tem como foco principal o uso de algoritmos de AM no processo de KDD.

KDD é uma área de pesquisa multidisciplinar que envolve outras áreas de pesquisa como Inteligência Artificial, Bando de Dados, Estatística e Visualização Gráfica (Piatetsky-Shapiro, 1989). Cada uma dessas áreas contribui com métodos e ferramentas para a consolidação de KDD.

2.7 O Projeto DISCOVER

A Descoberta de Conhecimento em Bancos de Dados, tanto como uma aplicação efetiva, quanto como área de pesquisa, é um processo de engenharia. Ela não é algoritmica, ou seja, não existe nenhum tipo de receita ou roteiro que deve ser seguido. Ela requer experimentação, análise e comparação de diversos modelos na busca por um resultado satisfatório.

Na condução desses experimentos, atividades como transformações de formato, adaptações, execução de algoritmos, medições, entre outras, devem ser executadas um grande número de vezes.

Quando se trata de uma aplicação de descoberta de conhecimento, muitas dessas tarefas podem ser automatizadas com o uso de sistemas integrados comerciais. Geralmente, essas ferramentas comerciais tem um caráter mais exploratório e fazem uso de algoritmos e ferramentas proprietários, o que dificulta o seu uso por pesquisadores que pretendem analisar e desenvolver novos algoritmos e ferramentas.

Uma alternativa é a utilização de ferramentas de domínio público, tais como: o $\mathcal{MLC}$++[6] (Kohavi, Sommerfield & Dougherty, 1994, 1997); WEKA[7] (Witten & Frank, 2000); YALE[8] (Fischer, Klinkenberg, Mierswa & Ritthoff, 2002), entre outras.

$\mathcal{MLC}$++ apresenta diversas facilidades, tais como interfaces para os principais algoritmos de aprendizado, formato padrão para os dados de entrada, obtenção de estatísticas de desempenho e visualização gráfica das estruturas simbólicas obtidas por alguns algoritmos. Entretanto, essa biblioteca trata os classificadores como "*caixas pretas*", não fornecendo uma visão única dos classificadores simbólicos que podem ser extraídos utilizando a biblioteca. A incorporação de novos aplicativos à $\mathcal{MLC}$++ também não é uma tarefa trivial, já que é necessário recompilar a biblioteca a cada novo aplicativo adicionado, além da dificuldade de adaptação do novo aplicativo aos padrões da $\mathcal{MLC}$++. Além disso, a partir de 1995 a $\mathcal{MLC}$++ passou a ser de propriedade da SGI[9], e a última versão disponibilizada data de 1997.

WEKA, ao contrário da biblioteca $\mathcal{MLC}$++, reimplementa os algoritmos de aprendizado na linguagem Java[10] (Horstmann & Cornell, 1997). Essa abordagem padroniza as interfaces e produz código uniforme, facilitando a inclusão de novos aplicativos. Entretanto, as novas versões dos algoritmos originais, ou até mesmo de novos algoritmos propostos pela comunidade, podem não ser disponibilizados no WEKA, pois exigem a sua conversão em código Java. Além disso, a recodificação de algoritmos sempre está sujeita a falhas, as quais podem causar um comportamento anômalo do algoritmo reimplementado que não ocorre no algoritmo original.

O ambiente YALE também está sendo implementado em Java. Esse ambiente utiliza alguns dos algoritmos reimplementados no ambiente WEKA, mas também é capaz de executar alguns algoritmos em suas implementações originais. O ambiente YALE possui diversas semelhanças com o ambiente computacional para descoberta de conhecimento, do qual este trabalho faz parte. O ambiente do qual este trabalho faz parte é discutido logo adiante.

Eventuais problemas decorrentes da utilização de algoritmos de aprendizado reimplementados conduzem, muitas vezes, à necessidade de utilização dos algoritmos de aprendizado tal como foram implementados pelos seus idealizadores e, por conseguinte, todas

[6] *Machine Learning Library in* C++. http://www.sgi.com/tech/mlc.
[7] *Waikato Environment for Knowledge Analysis*. http://www.cs.waikato.ac.nz/ml/weka.
[8] *Yet Another Learning Environment*.http://yale.cs.uni-dortmund.de.
[9] http://www.sgi.com.
[10] http://java.sun.com.

as atividades necessárias para a execução dos experimentos devem ser feitas para cada algoritmo. Essa necessidade implica no desenvolvimento de programas para automatizar essas tarefas.

Muitos pesquisadores em nosso laboratório de pesquisa — LABIC[11] — utilizam esses algoritmos em suas pesquisas e, muitas vezes, reimplementam algum tipo de código semelhante para a realização de experimentos.

Esses fatores levaram alguns pesquisadores do nosso laboratório de pesquisa a propor e implementar, na forma de *scripts* na linguagem Perl[12] (Wall, Christiansen & Schwartz, 1996), uma série de ferramentas para facilitar a configuração e execução de experimentos. Surgiu então a proposta (Baranauskas & Batista, 2000) de criar um projeto no qual todos os membros do laboratório estariam envolvidos. A esse projeto foi dado o nome de DISCOVER.

Um dos principais objetivos do projeto DISCOVER é tentar diminuir o esforço, por parte dos membros do projeto, necessário para implementar um experimento. Muitas vezes, um programa semelhante é implementado diversas vezes, por membros diferentes, por falta de comunicação entre os membros ou por falta de documentação dos programas já implementados. Ainda, é comum que diversas implementações sejam perdidas quando seus autores se desligam do laboratório, após finalizar o curso.

No projeto DISCOVER são utilizados os algoritmos de aprendizado implementados pela comunidade e ferramentas com finalidades específicas, desenvolvidas pelos pesquisadores relacionados ao projeto, tais como ferramentas de pré-processamento de dados e textos, amostragem, avaliação de erro, mesclagem de regras de diversos classificadores, análise de cobertura de regras, entre outros.

De uma forma geral, o projeto DISCOVER pode ser entendido como um conjunto de métodos que são aplicados sobre os dados ou sobre o conhecimento extraído a partir dos dados. Dessa forma, é muito importante que o projeto DISCOVER ofereça uma base sólida para a manipulação de dados e conhecimento. Essa base é composta por sintaxes padrões para a representação de dados e conhecimento, e por bibliotecas que oferecem um conjunto de funcionalidades básicas de manipulação de arquivos nessas sintaxes. Atualmente, existem definidas sintaxes padrão para a representação de dados e para a representação de conhecimento extraído de diversos indutores simbólicos, bem como bibliotecas que oferecem funcionalidades sobre essas sintaxes padrão (Batista & Monard,

[11] Laboratório de Inteligência Computacional. http://labic.icmc.usp.br.
[12] *Practical Extraction and Report Language*. http://www.perl.com/.

2003b; Prati, Baranauskas & Monard, 2001b,a). Novas sintaxes estão sendo especificadas, principalmente para a representação de regras de regressão (Dosualdo, 2002), regras de associação (Melanda, 2002) e *clusters* (Martins, 2001).

2.7.1 O Ambiente DISCOVER

A principio, o projeto DISCOVER consistiria apenas de um repositório de *scripts*. De uma forma geral, *scripts* são pequenos programas que realizam tarefas atômicas. Por meio da combinação desses *scripts* independentes seria possível a realização de tarefas mais complexas. Dessa forma, o projeto consistiria de uma série de ferramentas independentes que poderiam ser compostas conforme as necessidades dos usuários, e reunidas em um repositório compartilhado entre esses usuários.

Posteriormente surgiu a proposta de criar um ambiente integrado, ao qual foi dado o nome de ambiente DISCOVER, em que os *scripts* seriam substituídos por bibliotecas de classes e essas bibliotecas *empacotadas* como componentes, com a composição dos componentes sendo feita por meio de uma interface gráfica (Geromini, 2002).

Essa abordagem, além de benefícios tais como uma maior facilidade de utilização e melhor performance (com a utilização de *scripts* independentes não é possível o compartilhamento de memória, causando uma sobrecarga com operações de leitura/escrita) trouxe também novos desafios para a execução do projeto. Para se tornar um ambiente integrado, várias questões que dizem respeito ao gerenciamento do processo, comunicação entre os participantes e interação entre os diferentes componentes devem ser respondidas. Para ajudar a responder essas questões foi feito um estudo para fazer do processo de implementação do ambiente DISCOVER um processo de Engenharia de *Software* (Rozante, 2003)

Além disso, um outro ponto importante diz respeito à arquitetura do ambiente. Várias considerações devem ser feitas quanto à integração dos componentes, além de um conjunto de processos que suportam e criam um ambiente de execução apropriado para o sistema. Prati (2003) propõe um *framework* para a integração dos componentes do ambiente DISCOVER baseado em *software patterns*, no qual os componentes são integrados por meio de uma linguagem baseada em XML, a qual foi dada o nome de xDML.

A princípio, o ambiente DISCOVER terá como principal objetivo fornecer um campo de prova para os pesquisadores do nosso laboratório. A vantagem do ambiente DISCOVER como ferramenta de apoio à pesquisa em KDD, em relação a outros sistemas, é a

visão unificada que os formatos padrão oferecem para quem está desenvolvendo novos componentes, além de um conjunto de ferramentas de manipulação dos mesmos.

No caso de pré-processamento de dados, atividade fundamental no processo de KDD, na qual é gasto a maior parte do tempo e do esforço despendido em todo o processo de KDD, foi projetado e implementado o ambiente DISCOVER LEARNING ENVIRONMENT — DLE, o qual faz parte deste trabalho. O ambiente DLE, apresentado no Capítulo 4, é integrado ao projeto DISCOVER, e provê as funcionalidades necessárias para pré-processamento de dados nesse ambiente.

O ambiente DLE é composto por dois módulos, a biblioteca de classes DISCOVER OBJECT LIBRARY — DOL (Batista & Monard, 2003d) e o ambiente para gerenciamento de experimentos SNIFFER (Batista & Monard, 2003b). Ambos os módulos foram projetados e implementados neste trabalho e também são descritos no Capítulo 4.

A biblioteca DOL provê um conjunto de métodos de pré-processamento de dados. Entre as principais funcionalidades estão diversas abordagens para tratamento de valores desconhecidos, métodos para balanceamento de conjuntos de dados com grandes diferenças entre o número de exemplos de cada classe, além de prover métodos com funcionalidades mais básicas como o cálculo de estatísticas descritivas básicas, remoção, troca de posição e criação de novos atributos a partir de expressões lógicas e aritméticas, embaralhamento, amostragem aleatória e complementar, filtros de exemplos a partir de expressões lógicas, entre outros.

O ambiente de gerenciamento de experimentos SNIFFER é um ambiente capaz de automatizar a execução de experimentos, facilitando a comparação de diferentes métodos e a publicação de resultados.

2.7.2 Outros Trabalhos Realizados e em Desenvolvimento

Como já mencionado, diversas ferramentas e sistemas que compõem o projeto DISCOVER já estão implementadas, utilizando a linguagem Perl. Uma biblioteca para a manipulação de conjuntos de regras (classificadores) no formato padrão do projeto DISCOVER foi desenvolvida (Prati, Baranauskas & Monard, 2002), além de uma série de ferramentas que transformam os classificadores simbólicos induzidos por diversos sistemas de aprendizado para o formato padrão de regras (Prati, Baranauskas & Monard, 2001a).

Uma vez convertido no formato padrão, um classificador simbólico pode ser avaliado por uma outra ferramenta também disponível (Prati, Baranauskas & Monard, 2001b), com

um conjunto de dados, obtendo um conjunto de medidas, também em um formato padrão. Essas medidas incluem os valores da matriz de contingência, calculada para cada regra, de forma que métricas de qualidade de regras possam ser facilmente obtidas (Lavrač, Flach & Zupan, 1999). Também estão disponíveis ferramentas para obtenção de estatísticas de desempenho de classificadores utilizando métodos de reamostragem (Kemp, Batista & Monard, 2001).

Uma outra ferramenta implementada é o sistema XRULER (Baranauskas & Monard, 2000b, 2003; Baranauskas, 2001), no qual um conjunto de exemplos é dividido em amostras de treinamento e teste. As amostras de treinamento são submetidas a indutores diferentes e os classificadores obtidos são convertidos para o formato padrão de regras. As regras no formato padrão são avaliadas e um algoritmo de cobertura é utilizado, gerando um subconjunto das regras que cobrem os exemplos obtendo, dessa maneira, um classificador simbólico final. Algumas outras pesquisas realizadas e em curso em nosso laboratório também resultaram em diversos trabalhos e idéias sobre novas ferramentas para o projeto DISCOVER. Entre essas pesquisas pode-se destacar o processamento de textos (Martins, 2001; Imamura, 2001), Aprendizado de Máquina (Monard & Batista, 2002; Pila & Monard, 2002; Pila, 2001; Gomes, 2002) e processamento de padrões (Baranauskas & Monard, 2003; Lee & Monard, 2000; Milaré et al., 2002; Martins et al., 2002; Baranauskas & Monard, 2000a; Baranauskas, 2001; Milaré, 2000; Paula, 2003; Melanda, 2002; Pugliesi, 2001).

2.8 Considerações Finais

Aprendizado de Máquina é uma das áreas de pesquisa mais ativas em Inteligência Artificial. Nos últimos anos foram propostos diversos paradigmas, métodos e aplicações de AM. A aplicação desses métodos em problemas reais tem demonstrado o amadurecimento da pesquisa realizada na área. Com o surgimento da área de pesquisa de Descoberta de Conhecimento em Bancos de Dados, muito interesse têm sido depositado na área de AM.

O projeto DISCOVER tem como um de seus objetivos principais ser uma ferramenta que integre as implementações e algoritmos de aprendizado mais utilizados pela comunidade de AM, servindo como um campo de prova para pesquisas em descoberta de conhecimento.

Neste trabalho o maior interesse está na aplicação dos algoritmos de AM no processo de KDD. O processo de KDD foi brevemente apresentado neste capítulo, e esse mesmo processo é o principal tema do capítulo seguinte.

Capítulo 3

Pré-processamento de Dados

3.1 Considerações Iniciais

Neste capítulo é descrito o processo de Descoberta de Conhecimento de Bancos de Dados — KDD — e suas fases. Dentre todas as fases é dada maior ênfase às fases de coleta, pré-processamento e transformação de dados, pois essas fases estão mais diretamente relacionados com o tema deste trabalho.

Este capítulo está organizado da seguinte forma: na Seção 3.2 é descrito o processo de KDD e seus principais passos. Em seguida, são discutidas mais detalhadamente as fases do processo de KDD de coleta de dados na Seção 3.3, de pré-processamento de dados na Seção 3.4, e de transformação de dados na Seção 3.5. Por fim, na Seção 3.6 são apresentadas as considerações finais deste capítulo.

3.2 O Processo de Descoberta de Conhecimento em Bancos de Dados

Uma das primeiras aplicações dos computadores foi gerenciar dados. Desde então, as instituições que utilizam computadores têm armazenado dados em grandes volumes, e com uma velocidade de aquisição crescente. Avanços nas tecnologias de armazenamento de dados tais como dispositivos de armazenamento mais rápidos, com maior capacidade de armazenamento e mais baratos, além de sistemas de gerenciamento de bancos de dados mais eficientes, da tecnologia de *Data Warehousing* (Seção 3.3) e do *World Wide Web* têm

contribuído para fazer com que existam enormes volumes de dados disponíveis a todos.

É alarmante também a distância crescente entre a geração de dados e a capacidade de analisar e compreender esses dados. Conforme o volume de dados aumenta, a proporção dos dados que é analisada e entendida pelas pessoas diminui. Escondido entre todo esse volume de dados está a informação potencialmente útil, a qual dificilmente pode ser identificada e utilizada.

Existe, portanto, a necessidade de uma nova geração de técnicas e ferramentas com a habilidade de assistir os analistas humanos de uma forma "inteligente" e automática na procura de informações úteis, previamente desconhecidas, nos dados. Tais técnicas e ferramentas são objeto de estudo de uma nova área de pesquisa chamada de *Descoberta de Conhecimento em Bancos de Dados* — KDD. Existem diversas definições para KDD, uma das mais utilizadas é:

Descoberta de conhecimento em bancos de dados é um processo não trivial para identificar padrões válidos, novos, potencialmente úteis e compreensíveis em dados existentes (Fayyad, Piatetsky-Shapiro & Smyth, 1996).

Segundo essa definição, KDD é um processo, isso significa que KDD é composto de diversas fases, descritas mais à frente, as quais podem ser repetidas em múltiplas iterações. Por não trivial é entendido que alguma busca ou inferência é utilizada, isto é, KDD não é um simples cálculo de quantidades pré-definidas, como calcular a média de um conjunto de valores. Esse tipo de informação pode ser obtido utilizando ferramentas tradicionais de consulta em bancos de dados e ferramentas *OLAP*[1] para *Data Warehouses*. Os padrões descobertos durante o processo de KDD devem ser válidos em novos dados com um certo grau de certeza. Os padrões devem também ser novos e potencialmente úteis, isto é, devem levar a algum benefício ao usuário ou à aplicação. Por fim, os padrões devem ser compreensíveis aos analistas humanos, se não imediatamente, ao menos após algum pós-processamento.

Freqüentemente, três *atores* estão envolvidos no processo de KDD:

O analista de dados

O analista de dados é aquele que entende das técnicas envolvidas no processo de Descoberta de Conhecimento em Bancos de Dados. Essa pessoa tem conhecimento sobre o funcionamento dos algoritmos e das ferramentas utilizadas no processo, mas não necessariamente conhece o domínio ao qual os dados pertencem;

[1] *On-line analytical processing.*

O especialista no domínio

O especialista no domínio é aquele que conhece o domínio no qual o processo de Descoberta de Conhecimento será aplicado. Por exemplo, pode-se utilizar KDD para encontrar padrões de vendas de produtos, nesse caso o especialista no domínio pode ser um especialista em *marketing*; ou encontrar padrões entre clientes que não são capazes de honrar um empréstimo, nesse caso o especialista no domínio seria um especialista em concessão de crédito;

O usuário

O usuário é aquele que irá utilizar o resultado do processo de KDD. Normalmente, o usuário não é somente uma pessoa, mas uma instituição, uma empresa ou um departamento de uma empresa.

Esses três atores denotam diferentes habilidades, mas não são, necessariamente, pessoas diferentes. Por exemplo, freqüentemente os papéis de usuário e especialista são exercidos por uma mesma pessoa quando o usuário possui conhecimento detalhado do domínio de aplicação.

Deve-se ressaltar que o usuário deve sempre estar presente na equipe envolvida em um projeto de KDD. Isso porque o problema a ser resolvido deve ser de importância para o usuário. Além disso, preferencialmente, esse problema não deve ser facilmente solucionável por alguma técnica tradicional. O usuário é quem estabelece os critérios de avaliação dos resultados, além de decidir se o conhecimento descoberto será aplicado nos processos da instituição. Uma das formas de convencer o usuário a confiar e utilizar os resultados de um processo de KDD é envolvê-lo durante todo o processo (Saitta & Neri, 1998).

Como informado anteriormente, KDD é um processo que envolve diversas fases. As fases de coleta, pré-processamento, transformação, mineração de dados e avaliação e interpretação de resultados são ilustradas na Figura 3.1 na página seguinte. A seguir são descritas as principais fases do processo de KDD.

Identificação e entendimento do problema

É necessário identificar entre as necessidades do usuário aquelas que podem ser resolvidas por meio do uso de algum método de *Mineração de Dados* — MD. De uma forma geral, os principais métodos de *MD* são *classificação*, *regressão*, *segmentação*, *sumarização* e *detecção de desvio* (Fayyad, Piatetsky-Shapiro & Smyth, 1996). Freqüentemente, o usuário descreve o seu problema informalmente e é responsabilidade do analista de dados mapeá-lo para um dos métodos de MD. Para que isso

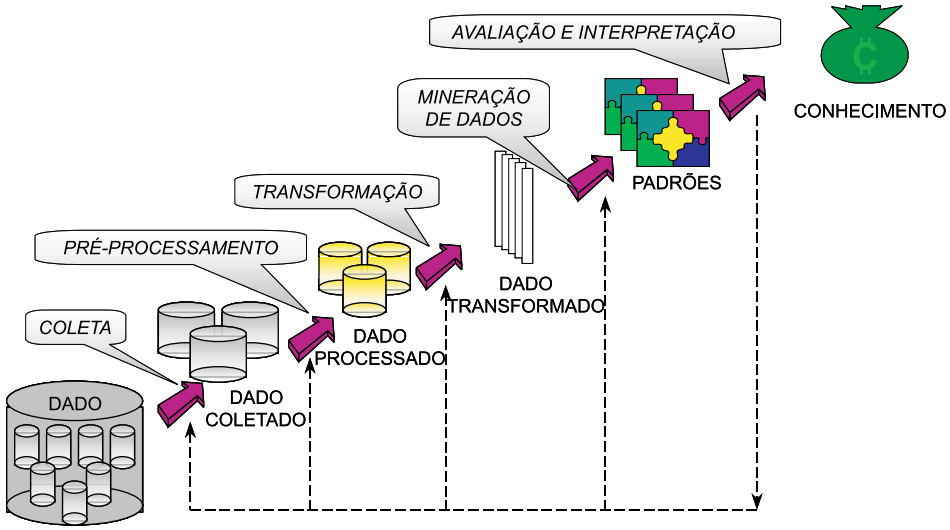

Figura 3.1: Principais fases do processo de KDD.

seja feito, o analista de dados precisa, com a ajuda do especialista no domínio, obter informações sobre o domínio de aplicação.

Uma vez que o problema do usuário foi claramente entendido, não é sempre óbvio como mapeá-lo para um dos métodos de MD. Em muitos casos, o processo de KDD não pode resolver todo o problema e, dessa forma, sub-problemas mais pertinentes precisam ser identificados. Diversos pesquisadores têm reportado essas dificuldades (Asker & Boström, 1995; Schwabacher, Hirsh & Ellman, 1995).

Identificação de dados relevantes

Uma vez que um problema foi identificado e entendido, é necessário identificar quais atributos serão utilizados na análise. Um especialista no domínio pode fornecer ao analista de dados informações sobre quais atributos são, na sua opinião, os mais relevantes para a criação do modelo. Entretanto, esse procedimento pode limitar a originalidade do conhecimento descoberto. Sempre que possível, o analista de dados deve adicionar novos atributos e verificar a importância dessas variáveis no conhecimento gerado. É importante também verificar se esses dados existem nos bancos de dados da instituição ou podem ser encontrados em fontes de dados externas.

Coleta de dados

O próximo passo é coletar os atributos que serão utilizados na análise dos bancos de dados da instituição. Coletar dados é uma atividade crítica porque os dados podem não estar disponíveis em um formato apropriado para serem utilizados no processo de KDD. Ou, mesmo se disponíveis, os dados podem precisar ser rotulados com o auxílio de um especialista, como relatado por Provost & Danyluk (1995), caso seja

necessário a aplicação de um método de classificação.

Um dos principais problemas em coletar dados é descobrir onde os dados estão armazenados nos bancos de dados. Muitos dos sistemas de gerenciamento de dados que estão funcionando hoje foram criados há muitos anos, quando as técnicas de *Engenharia de Software* (Pressman, 1992) ainda não estavam bem desenvolvidas e/ou pouco difundidas. Como resultado, muitos desses sistemas são proprietários e possuem documentação insatisfatória, o que faz com que o processo de coleta de dados seja extremamente difícil. Recentemente, empresas têm implementado novos bancos de dados direcionados para dar suporte às pessoas responsáveis por tomar decisões. Esses bancos de dados são chamados de *Data Warehouses* (Kimball, 1996). *Data Warehouses* podem em muito facilitar a coleta de dados, uma vez que esses bancos de dados tentam integrar dados de diversos sistemas transacionais da forma mais confiável possível. Ao final do passo de coleta de dados obtém-se os dados em um formato que algoritmos utilizados na fase de MD aceitam como entrada, como por exemplo, uma tabela atributo-valor. Uma descrição mais detalhada dos desafios encontrados na fase de coleta de dados do processo de KDD pode ser encontrada na Seção 3.3 na página 38.

Pré-processamento de dados

Nesta fase busca-se aprimorar a qualidade dos dados coletados. Freqüentemente, os dados apresentam diversos problemas, tais como grande quantidade de valores desconhecidos, ruído (atributos com valores incorretos), atributos de baixo valor preditivo, grande desproporção entre o número de exemplos de cada classe, entre outros. Muitos dos sistemas utilizados na fase de MD são capazes de extrair padrões de dados que apresentam esses problemas. Por exemplo, a poda de árvores de decisão permite que sejam induzidas árvores sobre dados com ruído. Entretanto, é esperado que os sistemas utilizados na fase de MD possuam um desempenho superior, caso a maioria dos problemas presentes nos dados for removida antes da extração dos padrões. Como pré-processamento de dados é o tema central deste trabalho, a Seção 3.4 na página 40 discute os principais desafios dessa etapa do processo de KDD mais detalhadamente. Ainda, os próximos capítulos deste trabalho apresentam os resultados da pesquisa realizada sobre alguns dos principais problemas encontrados nessa fase do processo de KDD.

Transformação de dados

Após os dados serem pré-processados, pode ser necessário transformar a forma em que os dados estão representados com o objetivo de superar quaisquer limitações

existentes no algoritmo de extração de padrões que será aplicado. Por exemplo, diversas implementações de algoritmos utilizados em MD não são capazes de analisar certos tipos de dado como, por exemplo, atributos do tipo data e hora. Dessa forma, freqüentemente, atributos com esses tipos de dado são transformados em um outro atributo com a mesma informação, mas com um tipo de dado que o algoritmo seja capaz de analisar. Por exemplo, um atributo do tipo data pode ser transformado em um atributo do tipo inteiro, o qual representa o número de dias decorridos a partir de uma data fixa. Existem ainda outros problemas que precisam ser contornados nessa fase do processo de KDD. Na Seção 3.5 na página 46 pode ser encontrada uma descrição mais detalhada dos problemas a serem resolvidos na fase de transformação de dados.

Mineração de dados

A fase de Mineração de Dados (Rezende, Pugliesi, Melanda & Paula, 2003) envolve decidir quais algoritmos serão aplicados aos dados. Nesta fase, pode-se utilizar algoritmos provenientes de diversas áreas de conhecimento, tais como Aprendizado de Máquina, Estatística, Redes Neurais e Banco de Dados. Se o objetivo dessa fase é criar um modelo preditivo, então, decidir qual algoritmo é ótimo para o problema que está sendo analisado não é uma tarefa trivial. Esse fato ocorre pois é sabido que nenhum algoritmo é ótimo para todas as aplicações (Kibler & Langley, 1988; Dietterich, 1997a; Schaffer, 1994). Muitos estudos empíricos têm sido realizados a fim de relacionar o algoritmo de aprendizado com a natureza do problema a ser resolvido (Michie, Spiegelhalter & Taylor, 1994). Entretanto, encontrar tal relacionamento parece ainda ser um problema em aberto. Uma possível solução, que ainda precisa ser analisada para grandes volumes de dados, é combinar os resultados de vários classificadores em vez de selecionar um único classificador. *Ensembles* (Wolpert, 1992; Breiman, 1996; Quinlan, 1996) têm obtido muito sucesso em combinar os resultados de diferentes sistemas de aprendizado. Entretanto, a utilização de *ensembles* pode dificultar a fase de interpretação dos resultados. Uma outra possibilidade é a composição de classificadores simbólicos em um classificador final também simbólico, como no sistema XRULER proposto por Baranauskas (2001) e no sistema MCE proposto por Bernardini (2002).

Avaliação e interpretação de resultados

Após a fase de MD, o processo de KDD entra na fase de *avaliação* e *interpretação* dos resultados. Esta fase envolve todos os participantes. O analista de dados tenta descobrir se o classificador atingiu as expectativas, avaliando os resultados de acordo

com algumas métricas tais como taxa de erro, tempo de CPU e complexidade do modelo. O especialista no domínio irá verificar a compatibilidade dos resultados com o conhecimento disponível do domínio. E, por fim, o usuário é responsável por dar o julgamento final sobre a aplicabilidade dos resultados.

Em KDD o resultado final do processo deve ser compreensível. Entretanto, definir *compreensibilidade* não é uma tarefa trivial. Em certos contextos, compreensibilidade pode ser estimada pela simplicidade do modelo (como, por exemplo, número de nós de uma árvore de decisão). Entretanto, até o momento, não existe um mecanismo efetivo para medir a compreensibilidade do conhecimento. De acordo com Craven & Shavlik (1995), compreensibilidade é útil para validar o conhecimento (o especialista deseja inspecionar o conhecimento para verificar se o conhecimento é confiável), para descoberta de novos padrões, para a sugestão de melhores atributos e para o refinamento do conhecimento. Como a compreensibilidade de um modelo freqüentemente não pode ser facilmente estimada, uma segunda medida chamada *interessabilidade* (Silberschatz & Tuzhilin, 1995) é freqüentemente utilizada pela comunidade de KDD. Interessabilidade mede o valor de um padrão combinando validade, novidade, utilidade e simplicidade.

Para ser utilizado, o conhecimento adquirido não precisa necessariamente ser incorporado a um sistema de tomada de decisão. Por exemplo, Evans & Fisher (1994) descrevem uma aplicação na qual o conhecimento foi simplesmente escrito em papel para uso de operadores humanos. O produto esperado do processo de KDD é, de fato, informações úteis e interessantes a serem utilizadas, por exemplo, por pessoas que tomam decisões.

Este trabalho tem como um de seus principais objetivos pesquisar e desenvolver métodos para pré-processamento de dados. Neste trabalho é entendido que os dados do mundo real apresentam diversas imperfeições, e a fase de pré-processamento de dados busca identificar e corrigir tais imperfeições antes do início da extração dos padrões. As fases de coleta de dados, pré-processamento e transformação dos dados estão mais diretamente ligadas ao problema de tratar as imperfeições presentes nos dados. Por esse motivo, essas três fases são descritas mais detalhadamente nas próximas seções.

3.3 Coleta de Dados

A fase de coleta de dados é uma das mais trabalhosas de todo o processo de KDD. Essa fase freqüentemente envolve extrair dados de sistemas computacionais *legados*, ou seja, de sistemas antigos nos quais inexiste documentação a respeito do projeto e da arquitetura do sistema. Dessa forma, por mais que o usuário e o especialista no domínio saibam que uma determinada informação está registrada nos sistemas da empresa, pode-se não saber exatamente onde e de qual forma essa informação foi armazenada.

Os desafios encontrados em coletar os dados podem ser diminuídos se a instituição em questão possuir um *Data Warehouse*. *Data Warehouse* é um repositório de dados geralmente construído para dar suporte às pessoas que tomam decisões, tais como gerentes e diretores. Essa tecnologia tem sido largamente utilizada, uma vez que os bancos de dados transacionais não são considerados adequados para fornecer respostas para análises estratégicas. Os bancos de dados transacionais, sobretudo os de projeto mais antigo, freqüentemente apresentam diversos problemas, tais como, problemas de falta de documentação de projeto, problemas com inconsistências e integridade de dados, entre outros. O *Data Warehouse* é periodicamente atualizado com dados de sistemas transacionais e/ou de fontes externas. Dados extraídos de diferentes bancos de dados são integrados e sua consistência é verificada, na medida do possível, antes de serem carregados no *Data Warehouse*. Dessa forma, o *Data Warehouse* pode ser uma boa fonte de dados para um projeto de KDD.

Independentemente se a instituição possui um *Data Warehouse* ou não, a fase de coleta de dados pode ser considerada uma das mais trabalhosas. Alguns desafios que podem ser encontrados nessa fase são (Pyle, 1999):

Problemas legais e éticos
: Podem existir barreiras legais ou éticas que impeçam que dados sejam disponibilizados para análise. Por exemplo, instituições financeiras possuem barreiras legais que, sob algumas circunstâncias, impedem que dados referentes a movimentações financeiras de clientes sejam disponibilizados. Ainda, podem existir razões éticas que restrinjam o acesso aos dados como ocorre, por exemplo, com dados que identifiquem pacientes na área médica e clientes na área legal;

Motivos estratégicos
: Podem haver motivos estratégicos que impeçam o acesso à parte dos dados ou até mesmo a algumas estatísticas sobre os dados. Por exemplo, Chan & Stolfo (1998b)

descrevem uma análise para identificação de operações fraudulentas em cartões de crédito na qual a distribuição das classes foi alterada. A proporção de operações fraudulentas e não fraudulentas é uma informação estratégica mantida em absoluto segredo pelas companhias de cartão de crédito;

Razões políticas

Alguns dados podem pertencer a pessoas ou departamentos que pelos mais diversos motivos não apóiam a iniciativa de analisar esses dados. Essas pessoas podem impor restrições de acesso aos dados, atrasando ou inviabilizando a análise;

Formato dos dados

Por décadas, dados têm sido gerados e armazenados em diferentes formatos. Até mesmo computadores modernos utilizam inúmeros formatos para a codificação de dados. Por exemplo, existem diversos formatos para mídias (disquetes e fitas de diferentes tipos, CDROM, entre outros), e para codificação de dados (ASCII, EBCDIC, etc.) que podem complicar a coleta de dados de fontes distribuídas;

Conectividade

Para que os dados sejam analisados é necessário que eles estejam disponíveis *on-line* e conectados ao sistema que será utilizado nas análises. Tal conexão pode ser feita de diversas formas, como por exemplo, rede de computadores, fitas, discos, etc. Entretanto, sistemas antigos (legados) e proprietários podem dificultar a conectividade aos dados, uma vez que esses sistemas podem utilizar tecnologias obsoletas para a troca de informações, as quais não estão disponíveis nos novos sistemas computacionais;

Bancos de Dados e Aplicações Obsoletas

Como mencionado anteriormente, vários sistemas transacionais que atualmente operam em empresas foram projetados e desenvolvidos há vários anos. Nessa época, muitos dos métodos e técnicas de Engenharia de *Software* ainda não haviam sido desenvolvidos. Como resultado, existe pouca documentação sobre como esses sistemas foram projetados, e esse fato dificulta ou impede que certos dados sejam localizados e extraídos de sistemas transacionais antigos. Além disso, algumas aplicações e sistemas gerenciadores de banco de dados podem não ter um tipo de dado equivalente em outros sistemas, e unificar a representação dos dados pode se tornar um problema complicado.

Granularidade

Um outro problema importante refere-se a *granularidade* dos dados. Granularidade é o nível de detalhe em que os dados são armazenados. Os sistemas transacionais

normalmente armazenam os dados com todo o detalhe possível (também chamados de registros de transação), para que esses sistemas possam realizar as suas tarefas. Entretanto, os *Data Warehouses* dificilmente armazenam os dados de forma detalhada, por motivos de desempenho. É comum que no processo de carga de um *Data Warehouse* os dados sejam totalizados antes de serem armazenados. Por exemplo, os dados de venda de produtos podem ser totalizados, e somente o total de vendas no dia, de um determinado produto, seja armazenado. Esse fato faz com que os *Data Warehouses* sejam capazes de armazenar, e serem consultados, sobre dados referentes a vários anos de movimento. A decisão da granularidade em que os dados são armazenados é de extrema importância, pois dados armazenados de forma sumarizada não podem ser transformados em dados detalhados novamente. Se por um lado as agregações podem tornar um *Data Warehouse* mais ágil e compacto, por outro lado, certas agregações podem esconder detalhes dos dados que poderiam ser importantes em uma análise de MD.

3.4 Pré-processamento de Dados

O pré-processamento de dados em um processo de KDD é freqüentemente tido como sendo uma fase que envolve uma grande quantidade de conhecimento de domínio. Muitos pesquisadores têm citado que dados coletados diretamente de bancos de dados são de má qualidade, ou seja, possuem informações incorretas e imprecisas, além de uma grande quantidade de valores desconhecidos. Embora muitos dos algoritmos utilizados na fase de MD tenham sido projetados para manipular dados em tais situações, pode-se esperar que esses algoritmos gerem resultados mais precisos caso a maioria dos problemas presentes nos dados tenha sido, ou removida ou corrigida.

De uma forma geral, pré-processamento de dados é um processo semi-automático. Por semi-automático entende-se que essa fase depende da capacidade do analista de dados em identificar os problemas presentes nos dados, além da natureza desses problemas, e utilizar os métodos mais apropriados para solucionar cada um dos problemas.

Este trabalho propõe classificar as tarefas realizadas por métodos empregados na fase de pré-processamento em dois grupos:

Tarefas fortemente dependentes de conhecimento de domínio

Essas tarefas somente podem ser efetivamente realizadas com o uso de conhecimento específico ao domínio. Um método automático pode ser empregado para realizar

uma tarefa fortemente dependente de conhecimento de domínio, entretanto, esse método depende de que um conhecimento específico seja fornecido. Um exemplo são as verificações de integridade de dados. Por exemplo, em uma aplicação para concessão de crédito, um atributo `crédito`, o qual informa o valor emprestado, não pode assumir valores negativos. Ou ainda, caso existam informações a respeito do bem adquirido por meio de um empréstimo, esse atributo pode possuir faixas de valores permitidas, as quais dependem do bem adquirido. Com o uso de um conjunto de regras dependentes de domínio é possível verificar a integridade dos atributo presentes em um conjunto de dados;

Tarefas fracamente dependentes de conhecimento de domínio

Essas tarefas podem ser realizadas por métodos que extraem dos próprios dados as informações necessárias para tratar o problema de pré-processamento de dados. Se por um lado essas tarefas ainda dependem de conhecimento de domínio, pois é necessário selecionar o método correto para tratar o problema, por outro lado, essas tarefas podem ser realizadas por métodos que são mais automáticos do que aqueles utilizados em tarefas que dependem fortemente de conhecimento de domínio. São exemplos de tarefas fracamente dependentes de domínio o tratamento de valores desconhecidos e a identificação de valores extremos[2], entre outras.

As tarefas de pré-processamento de dados fortemente dependentes de domínio vêm sendo estudadas há alguns anos. Essas tarefas são muito semelhantes às tarefas que são encontradas no processo de carga de um *Data Warehouse*. Como mencionado anteriormente, os *Data Warehouses* são tipicamente alimentados com dados provenientes dos sistemas transacionais. Esses dados freqüentemente apresentam diversos problemas, e para auxiliar na solução de alguns desses problemas foi criada uma classe de ferramentas chamada *ETL*[3]. Algumas das principais tarefas de pré-processamento de dados fortemente dependentes de domínio são:

Identificação de inconsistências

Inconsistências podem ocorrem quando dados diferentes são representados pelo mesmo rótulo, ou quando o mesmo dado é representado por rótulos diferentes. Um exemplo de inconsistência ocorre quando um atributo assume diferentes valores, os quais representam, na verdade, uma mesma informação. Por exemplo, um atributo `nome_empresa`, que armazena nomes de empresas, assume os valores `USP`, `Usp`,

[2] *Outliers*.
[3] *Extraction, Transformation and Load*.

`Universidade de São Paulo`, etc, sendo que todos esses valores representam uma mesma instituição.

Identificação de poluição

Existem diversas fontes de poluição de dados. De certa forma, pode-se entender por poluição a presença de dados distorcidos, os quais não representam os valores verdadeiros.

Uma possível fonte de poluição de dados é a tentativa, por parte dos usuários do sistema que coletam os dados, de utilizar esse sistema além da sua funcionalidade original. Por exemplo, Pyle (1999) cita o caso de uma empresa de cartão de crédito cujo banco de dados possuía um campo `gender` para armazenar o sexo de seus clientes. Entretanto, alguns registros assumiam o valor `B` para esse atributo, o qual, posteriormente, descobriu-se que correspondia à informação `Business`. Originalmente, o sistema tinha sido projetado somente para cadastrar cartões para pessoas físicas, porém, quando cartões para empresas foram permitidos, não havia um campo específico para indicar que o cadastrado era uma empresa. Essa informação foi então armazenada no campo `gender`.

Um segundo motivo que pode gerar poluição nos dados é a resistência humana em entrar com os dados corretamente. Enquanto que campos em um banco de dados podem ser incluídos para capturar informações valiosas, esses campos podem ser deixados em branco, incompletos ou simplesmente com informações incorretas. Pyle (1999) menciona uma empresa que possuía, à primeira vista, um banco de dados bastante promissor. Informações demográficas como tamanho da família e *hobbies*, entre outras, tinham sido incluídas no projeto do banco de dados. Apesar de que essas informações eram de muito valor para a equipe de *marketing*, os vendedores viam esse processo como um impecílio para o processo de venda. Posteriormente, os vendedores descobriram algumas combinações de valores que satisfaziam o sistema e as usaram para alimentá-lo. Assim, além de não entrar nenhuma informação de valor, os vendedores acabaram por inserir falsos padrões que poderiam ter sido descobertos por um algoritmo utilizado na fase de MD.

Verificação de integridade

Analisar a integridade dos dados freqüentemente envolve uma análise das relações permitidas entre os atributos. Por exemplo, um empregado pode possuir vários carros, entretanto, um mesmo empregado não pode possuir mais de um número funcional em um dado sistema. Dessa forma, é possível analisar os atributos por meio de faixa de valores válidos.

Uma caso especial de verificação de integridade de dados é a identificação de casos extremos. Casos extremos são casos em que a combinação dos valores é válida, pois os atributos estão dentro de faixas de valores aceitáveis, entretanto, a combinação dos valores dos atributos é muito improvável. A identificação de casos extremos pode ser considerada uma tarefa fracamente dependente de domínio, pois a probabilidade das combinações de valores de atributos pode ser feita a partir dos dados disponíveis. Uma discussão mais detalhada sobre a identificação de casos extremos é feita mais adiante nesta seção.

Identificação de atributos duplicados e redundantes

Redundância ocorre quando uma informação essencialmente idêntica é armazenada em diversos atributos. Um exemplo é possuir atributos em uma mesma tabela tais como `preço por unidade`, `número comprado` e `preço total`. O maior dano causado pela redundância para a maioria dos algoritmos utilizados na fase de MD é um aumento no tempo de processamento. Entretanto, alguns métodos são especialmente sensíveis ao número de atributos, e variáveis redundantes podem comprometer seus desempenhos. Se o problema de coletar atributos redundantes não for solucionado durante a fase de coleta de dados, existe a possibilidade de utilizar métodos de pré-processamento de dados, conhecidos como métodos de seleção de atributos, para tentar identificar e remover os atributos redundantes.

Defaults

A maioria dos sistemas gerenciadores de banco de dados permitem valores *defaults* para alguns atributos. Esses valores podem causar algumas confusões, especialmente se o analista de dados não está informado a respeito. Um valor *default* pode estar ligado condicionalmente a outros atributos, o que pode criar padrões significantes à primeira vista. Na realidade, tais valores *defaults* condicionais simplesmente representam falta de informações, em vez de informações relevantes. Um exemplo é a área médica na qual os valores de um atributo, como por exemplo, `período de gravidez`, está condicionalmente ligado a valores de outros atributos, como por exemplo, `sexo`. Valores *defaults* podem ser especialmente perigosos quando o usuário está interessado em uma análise preditiva.

Além das tarefas listadas, existe uma outra classe de tarefas de pré-processamento de dados, as quais são fracamente dependente do domínio de aplicação. Essas tarefas de pré-processamento de dados podem ser tipicamente solucionadas por métodos que extraem do próprio conjunto de dados as informações necessárias para tratar o problema. Essas tarefas são o principal foco de estudo deste trabalho e são brevemente descritas a seguir.

Algumas delas são tratadas mais detalhadamente nos próximos capítulos.

Tratamento de valores desconhecidos

Um problema comum em pré-processamento de dados é o tratamento de valores desconhecidos. Muitas técnicas têm sido aplicadas, sendo algumas delas bastante simples, como a substituição dos valores desconhecidos pela média ou moda do atributo. Entretanto, outras técnicas mais elaboradas podem ser implementadas e avaliadas experimentalmente. Por exemplo, pode-se substituir os valores desconhecidos por valores preditos utilizando um algoritmo de aprendizado.

Identificação e descrição de valores extremos

Valores extremos são dados que aparentemente não seguem o mesmo padrão dos demais. Estatísticos têm pesquisado por métodos de identificação de valores extremos, uma vez que esses valores podem distorcer os resultados obtidos por diversos métodos paramétricos (Barnett & Lewis, 1994).

Entretanto, valores extremos precisam ser tratados com cuidado, uma vez que casos que possuem valores extremos que, a principio, parecem ser dados incorretos, podem ser dados válidos. Na realidade, os casos com valores extremos podem representar a informação mais interessante, pela qual o analista de dados está procurando.

Diversos métodos para identificar valores extremos foram propostos por pesquisadores de AM, tais como, o método de *filtro* (John, 1995; Brodley & Friedl, 1999), e de Aprendizado *Instance-based*, como por exemplo, *Ligações Tomek* (Tomek, 1976).

Tratamento de conjuntos de dados com classes desbalanceadas

Conjuntos de dados com classes desbalanceadas são aqueles que possuem uma grande diferença entre o número de exemplos pertencentes a cada valor de um atributo classe qualitativo. A maioria dos algoritmos de AM tem dificuldades em criar um modelo que classifique com precisão os exemplos da classe minoritária. Uma forma de solucionar esse problema é procurar por uma distribuição da classe que forneça um desempenho aceitável de classificação para a classe minoritária. *Seleção Unilateral* foi utilizada por Kubat & Matwin (1997) para balancear um conjunto de dados contendo informações colhidas de fotos de satélites. Métodos de seleção unilateral foram estudados e aprimorados em (Batista & Monard, 1998; Batista, Carvalho & Monard, 1999).

Seleção de atributos

Seleção de atributos é um problema muito importante em KDD (Blum & Langley, 1997; Kohavi, 1997; John, Kohavi & Pfleger, 1994). Ele consiste em encontrar

um subconjunto de atributos no qual o algoritmo de AM utilizado em MD irá se concentrar. Existem diversas razões que justificam o uso de métodos para seleção de atributos. As principais são (Lee, Monard & Baranauskas, 1999):

1. Muitos algoritmos de AM não funcionam bem com uma grande quantidade de atributos, dessa forma, seleção de atributos pode melhorar o desempenho desses algoritmos;
2. Com um número menor de atributos o conhecimento induzido por algoritmos de AM simbólico é, freqüentemente, mais compreensível;
3. Alguns domínios possuem um alto custo de coletar dados, nesses casos, métodos de seleção de atributos podem diminuir o custo da aplicação.

Existem diversas abordagens propostas para selecionar um subconjunto de atributos. De uma forma geral, pode-se dividir as abordagens mais utilizadas em pré-processamento de dados em três grupos (Kohavi, 1997; Baranauskas & Monard, 1998):

Embutida

A abordagem embutida consiste na seleção de atributos realizada como parte do processo de criação do modelo por parte de um algoritmo de AM;

Filtro

A abordagem filtro consiste em aplicar um método de seleção de atributos anterior à aplicação do algoritmo de AM, geralmente analisando características do conjunto de exemplos que podem levar a selecionar alguns atributos e excluir outros;

Wrappers

A abordagem *wrapper* consiste em selecionar um subconjunto de atributos e medir a precisão do classificador induzido sobre esse subconjunto de atributos. É realizada uma busca pelo subconjunto que gera o classificador com menor erro. Essa busca avalia cada subconjunto candidato, até que o critério de parada, relacionado com a precisão do classificador induzido, seja satisfeito.

Construção de atributos

Os atributos podem ser considerados inadequados para a tarefa de aprendizado quando são fracamente ou indiretamente relevantes, condicionalmente relevantes ou medidos de modo inapropriado (Baranauskas, Monard & Horst, 1999; Baranauskas & Monard, 1999; Lee, 2000). Se os atributos utilizados para a descrição do

conjunto de dados são inadequados, os algoritmos de AM utilizados em MD provavelmente criarão classificadores imprecisos ou excessivamente complexos (Bloedorn & Michalski, 1998). Atributos fracamente, indiretamente ou condicionalmente relevantes podem ser individualmente inadequados, entretanto, esses atributos podem ser convenientemente combinados gerando novos atributos que podem mostrar-se altamente representativos para a descrição de um conceito. O processo de construção de novos atributos é conhecido como *construção de atributos* ou *indução construtiva*[4] (Michalski, 1978; Bloedorn & Michalski, 1998).

Assim, construção de atributos é o processo de composição de atributos ditos primitivos[5], produzindo-se novos atributos possivelmente relevantes para a descrição de um conceito.

De uma forma bastante ampla, o processo de indução construtiva pode ser dividido em duas abordagens: a automática e a guiada pelo usuário. A *indução construtiva automática* consiste em um processo de construção de atributos guiada automaticamente pelo método de construção. Geralmente, os atributos construídos são avaliados em relação aos dados, e podem ser descartados ou integrados ao conjunto de dados. A *indução construtiva guiada pelo usuário* utiliza o conhecimento do usuário ou do especialista no domínio para guiar a composição dos atributos.

Neste trabalho é proposta uma sintaxe para declaração de dados e atributos, chamada DSX[6], descrita no Apêndice A na página 169. A sintaxe DSX, integrada ao ambiente computacional *Discover*, descrito na Seção 2.7 na página 25, fornece, entre outros recursos, suporte à indução construtiva guiada pelo usuário.

3.5 Transformação de Dados

O principal objetivo desta fase é transformar a representação dos dados a fim de superar quaisquer limitações existentes nos algoritmos que serão empregados para a extração de padrões. De uma forma geral, a decisão de quais transformações são necessárias depende do algoritmo que será utilizado na fase de MD. Algumas das transformações mais comuns são:

[4] Os termos *aprendizado construtivo* ou *transformação de atributos* também são utilizados na literatura.
[5] Atributos pertencentes ao conjunto de dados original.
[6] *Discover Dataset Sintax*.

Normalização

Consiste em transformar os valores dos atributos de seus intervalos originais para um intervalo específico, como, por exemplo, $[-1, 1]$ ou $[0, 1]$. Esse tipo de transformação é especialmente valiosa para os métodos que calculam distâncias entre atributos. Por exemplo, um método como o K-VIZINHOS MAIS PRÓXIMOS tende a dar mais importância para os atributos que possuem um intervalo maior de valores. Outros métodos como redes neurais são reconhecidamente melhor treinadas quando os valores dos atributos são pequenos. Entretanto, normalização não é de grande utilidade para a maioria dos métodos que induzem representações simbólicas, tais como árvores de decisão e regras de decisão, uma vez que a normalização tende a diminuir a compreensibilidade do modelo gerado por tais algoritmos.

Discretização de atributos quantitativos

Muitos algoritmos possuem a limitação de trabalhar somente com atributos qualitativos. Entretanto, muitos conjuntos de dados possuem atributos quantitativos, e para que esses algoritmos possam ser aplicados é necessário utilizar algum método que transforma um atributo quantitativo em um atributo qualitativo, ou seja, em faixas de valores. Diversos métodos de discretização de atributos foram propostos pela comunidade. Uma descrição geral desses métodos pode ser encontrada em (Dougherty, Kohavi & Sahami, 1995; Kohavi & Sahami, 1996).

Transformação de atributos qualitativos em quantitativos

Alguns algoritmos não são capazes de manipular atributos qualitativos. Dessa forma, é necessário converter os atributos qualitativos em atributos quantitativos. Existem diversas abordagens para realizar essa transformação dependendo das características e limitações de cada algoritmo. De uma forma geral, atributos qualitativos sem ordem inerente, tal como `verde`, `amarelo` e `vermelho`, podem ser mapeados arbitrariamente para números. Entretanto, esse mapeamento acaba por criar uma ordem nos valores do atributo que não é real. Atributos qualitativos com ordem, tal como `pequeno`, `médio` e `grande`, podem ser mapeados para valores numéricos de forma a manter a ordem dos valores, por exemplo `pequeno` = `1`, `médio` = `2` e `grande` = `3`.

Alguns especialistas em redes neurais aconselham criar um nó de entrada para cada valor de um atributo qualitativo, ou seja, um atributo qualitativo com p valores diferentes deve ser desmembrado em p atributos binários. Cada novo atributo binário representa um único valor do atributo original. Esse mapeamento é feito sempre que o valor de p não seja muito grande, tipicamente $2 < p < 5$. Caso $p = 2$, então

é feito um mapeamento de tal forma que um dos valores do atributo quantitativo represente um valor baixo (tipicamente 0) e o outro valor represente um valor alto (tipicamente 1).

Atributos de tipos de dado complexos

A maioria dos algoritmos utilizados para extrair padrões não consegue trabalhar com tipos de dado mais complexos. Por exemplo, como já mencionado, atributos do tipo data e hora não são normalmente analisados pela maioria dos algoritmos utilizados na fase de MD. Dessa forma, é necessário converter esses atributos para algum outro tipo de dado com o qual esses algoritmos possam trabalhar. No caso específico dos tipos de dado data e hora, a escolha mais simples é pela conversão para o tipo inteiro. Isso pode ser feito calculando-se a diferença em dias, meses, ou qualquer outra unidade de tempo, entre os valores das datas do atributo em questão e uma data fixa. Por exemplo, um atributo `data de nascimento` pode ser convertido para `idade` calculando-se a diferença em anos entre os valores do atributo `data de nascimento` e a data atual.

3.6 Considerações Finais

Pré-processamento de dados é tido como uma das tarefas mais trabalhosas e demoradas de KDD. É considerado que aproximadamente 80% do tempo despendido no processo de KDD seja utilizado para pré-processar os dados (Pyle, 1999). Isso se deve, em parte, à existência de um grupo de problemas que são específicos para cada aplicação e que, dessa forma, precisam ser resolvidos com soluções específicas. Quanto ao outro grupo de tarefas independentes de domínio, existe ainda uma falta de ferramentas que englobem uma gama de soluções de pré-processamento que possam ser testadas pelo analista de dados.

No próximo capítulo é apresentado o projeto do ambiente Discover Learning Environment — DLE. O ambiente DLE visa dar suporte a implementação de métodos de pré-processamento de dados. Nos capítulos seguintes, o ambiente DLE é utilizado para analisar dois problemas de pré-processamento fracamente dependentes de conhecimento de domínio: o tratamento de valores desconhecidos, e de conjuntos de dados com classes desbalanceadas.

Capítulo 4

O Ambiente Discover Learning Environment — DLE

4.1 Considerações Iniciais

Neste capítulo é apresentada uma descrição do ambiente computacional Discover Learning Environment — DLE, o qual é integrado ao projeto Discover. O ambiente computacional DLE tem como principal objetivo prover um *framework* para que novos métodos de pré-processamento de dados possam ser rapidamente implementados e avaliados experimentalmente.

Este capítulo está organizado da seguinte forma: na Seção 4.2 é apresentada a biblioteca de classes Discover Object Library — DOL e o ambiente para gerenciamento de experimentos Sniffer, os quais integram o ambiente DLE; na Seção 4.3 é explicada a arquitetura, o projeto e os principais módulos da biblioteca DOL; na Seção 4.4 é discutido o funcionamento do ambiente computacional Sniffer, além do seu projeto e sua arquitetura interna; por fim, na Seção 4.5 são apresentadas as considerações finais deste capítulo.

4.2 Os Módulos do Ambiente DLE

O ambiente computacional DLE é composto por dois módulos: a biblioteca de classes Discover Object Library, e o ambiente para gerenciamento de experimentos Sniffer, os quais são descritos a seguir.

A biblioteca de classes Discover Object Library — DOL

A biblioteca DOL é uma biblioteca orientada a objeto baseada em *padrões de projeto*[1] (Gamma, Helm, Johnson & Vlissides, 1995; Shalloway & Trott, 2002), na qual novos métodos de pré-processamento de dados podem ser implementados. As classes da biblioteca DOL implementam as tarefas de manipulação e gerenciamento de dados mais comuns em pré-processamento, tais como gerenciamento de diferentes sintaxes de arquivos de dados e atributos, amostragens, métodos de reamostragem, estatísticas descritivas, normalizações de dados, etc.

Um objetivo futuro da biblioteca DOL é ser uma biblioteca de métodos de pré-processamento de dados. Ou seja, prover diversos métodos de pré-processamento de dados amplamente difundidos na literatura. Com isso, será possível fornecer uma base para a implementação de melhorias para os métodos de pré-processamento de dados amplamente difundidos, além da utilização desses métodos como referência em análises experimentais. Dessa forma, será possível responder à pergunta mais freqüente nas pesquisas na área de AM:

> *O método de pré-processamento de dados recém desenvolvido, ou a melhoria proposta a um método conhecido, é capaz de superar o método amplamente utilizado na literatura?*

O ambiente computacional Sniffer

O ambiente computacional Sniffer é um ambiente de gerenciamento de avaliações e comparações experimentais de algoritmos de AM. Uma vez que um novo método ou uma variante de um método conhecido de pré-processamento é implementado, torna-se necessário avaliá-lo experimentalmente com o objetivo de identificar sob quais circunstâncias esse método provê bons resultados e, ainda, se tal método pode superar os métodos presentes na literatura.

O ambiente computacional Sniffer automatiza a avaliação experimental, e é integrado com diversos sistemas de aprendizado, tais como C4.5 (Quinlan, 1988), C4.5 Rules (Quinlan, 1987a), ID3 (Quinlan, 1986), CN2 (Clark & Boswell, 1991) e NewId (Boswell, 1990). Dessa forma, é possível comparar o desempenho de novos métodos de pré-processamento de dados aplicados a diversos sistemas de aprendizado. As comparações são realizadas utilizando testes estatísticos de significância, e permitem identificar quando um método é capaz de superar seu concorrente com

[1] *Design patterns.*

95% ou 99% de confiança. O ambiente computacional Sniffer é integrado com o restante deste trabalho de três formas principais:

1. O ambiente Sniffer complementa a biblioteca DOL, pois permite que métodos de pré-processamento de dados desenvolvidos sejam avaliados e comparados experimentalmente de uma forma rápida e segura;
2. O ambiente Sniffer foi desenvolvido utilizando a biblioteca DOL. Dessa forma, esse ambiente serve também como *prova de conceito*, ou seja, o ambiente Sniffer é uma aplicação funcional que mostra que a biblioteca DOL pode ser utilizada na prática;
3. O ambiente Sniffer provê uma *API* (*Application Programming Interface*[2]) que permite estender as funcionalidades do ambiente. Essa API pode ser utilizada, por exemplo, para avaliar os sistemas de aprendizado sob novas medidas de desempenho (Provost, Fawcett & Kohavi, 1998; Provost & Fawcett, 2001) ou implementar novos testes estatísticos de significância (Dietterich, 1997b; Salzberg, 1997). Dessa forma, o ambiente Sniffer complementa as funcionalidades da biblioteca DOL, fazendo com que o usuário tenha total controle, desde a implementação até a avaliação de técnicas de pré-processamento de dados.

4.3 A Biblioteca de Classes Discover Object Library — DOL

A biblioteca DOL é uma biblioteca orientada a objeto que tem como objetivo dar suporte à criação de novos métodos de pré-processamento de dados. Futuramente, tem-se como objetivo que a biblioteca DOL seja mais que uma base para a criação de novos métodos, mas também uma biblioteca de métodos de pré-processamento de dados. Com isso, será possível não somente desenvolver novos métodos, mas também comparar o desempenho desses novos métodos com outros métodos já reconhecidamente eficazes.

Famili, Shen, Weber & Simoudis (1997) definem pré-processamento de dados como sendo um grupo de ações realizadas antes do início do processo de extração de padrões. Uma ação de pré-processamento de dados pode ser entendida como sendo uma transfor-

[2] O termo em inglês *Application Programming Interface* e a sigla *API* são utilizados neste trabalho por serem amplamente difundidos na literatura.

mação T que transforma um conjunto de dados E em um novo conjunto de dados E', ou seja

$$E' = T(E) \tag{4.1}$$

tal que:

1. E' preserva as informações valiosas em E;
2. E' elimina pelo menos um dos problemas existentes em E, e;
3. E' é mais útil que E.

Informações valiosas incluem componentes de conhecimento que existem nos dados, e é o objetivo do processo de análise de dados descobrir e apresentar essas informações.

A dimensão do conjunto de dados pré-processado E' não é necessariamente a mesma de E. É comum que métodos de pré-processamento de dados excluam atributos, criem novos atributos por meio da composição de atributos existentes, removam exemplos, ou, até mesmo, incluam "novos" exemplos derivados de exemplos presentes no conjunto de dados E.

Com base nessa definição de pré-processamento de dados pode-se entender que um método de pré-processamento de dados é um procedimento que tem como entrada um conjunto de dados E representado por meio de uma tabela atributo-valor[3], e tem como saída um conjunto de dados E' na mesma representação.

Nesse sentido, a biblioteca DOL provê um conjunto de métodos que fornece ao desenvolvedor uma forma simples de ter acesso aos dados. Os dados podem estar tanto disponíveis em arquivos texto quanto em tabelas de bancos de dados relacionais. Os dados são carregados em uma estrutura, e uma gama de métodos fornece acesso aos dados, possibilitando que esses dados sejam manipulados. Por fim, os dados podem ser armazenados em arquivos texto em diferentes sintaxes ou, ainda, carregados em uma tabela em um banco de dados relacional.

Entre as principais funcionalidades de biblioteca de classes DOL, pode-se citar:

Manipulação de atributos e dados

Manipular atributos e dados é a tarefa fundamental da biblioteca. A biblioteca

[3]Neste trabalho restringimos a discussão a métodos de pré-processamento de dados que podem manipular dados representados em tabelas atributo-valor.

DOL provê métodos que fornecem informações sobre o conjunto de dados que está sendo pré-processado. Os dados podem ser manipulados, isto é, lidos e modificados. Entre as principais tarefas de manipulação de atributos e dados pode-se citar:

1. Informar nome, tipo e posição de atributos no conjunto de dados, além de ser capaz de alterar os nomes dos atributos, realizar conversão de tipos de atributos e modificar a posição dos atributos dentro de um conjunto de dados;
2. Informar qual atributo é o atributo classe e alterar essa informação, tornando outro atributo do conjunto de dados o atributo classe;
3. Realizar indução construtiva apoiada pelo usuário, compondo novos atributos a partir de expressões aritméticas ou lógicas que envolvem atributos do conjunto de dados. Também criar novos atributos a partir de uma lista de valores que define os valores que o atributo assume para cada exemplo do conjunto de dados;
4. Remover atributos definitivamente do conjunto de dados, ou somente ignorá-los temporariamente;
5. Ler o valor que um determinado atributo assume em um certo exemplo, e ser capaz de modificar esse valor realizando todas as verificações de integridade de tipo;
6. Adicionar novos exemplos ao conjunto de dados e ser capaz de remover determinados exemplos temporariamente ou definitivamente do conjunto de dados.

Integração com diversos sistemas de aprendizado

É essencial que a biblioteca de classes DOL possa ser integrada com os principais sistemas de aprendizado acadêmicos e comerciais. Muitos desses sistemas podem ler e gravar dados por meio de arquivos texto. Infelizmente, a grande maioria dos sistemas de aprendizado é capaz de ler somente arquivos texto em alguma sintaxe proprietária. Portanto, é necessário criar conversores capazes de armazenar os dados carregados na biblioteca em arquivos texto em diversas sintaxes. Atualmente, a biblioteca de classes DOL é capaz de armazenar informações referentes a dados e atributos nas sintaxes dos sistemas de aprendizado listados na Tabela 4.1 na página seguinte;

Integração com sistemas gerenciadores de bancos de dados

Com o surgimento da aplicação de métodos de extração de conhecimento em dados provenientes de bancos de dados, os sistemas gerenciadores de banco de dados

C4.5 e C4.5 Rules	Ripper	Weka
CN2 e NewId	RT	Cart
C5.0 e Cubist	M5	MineSet
SNNS	SVM Torch	Trepan

Tabela 4.1: Sistemas de aprendizado cujas sintaxes são suportadas atualmente pela biblioteca DOL.

têm integrado, cada vez mais, métodos de extração de conhecimento em seus *engines*. Esse fato motiva a integração da biblioteca de classes DOL com sistemas gerenciadores de bancos de dados. A biblioteca DOL utiliza a interface *Database Independent Interface for Perl — DBI*[4] (Descartes & Bunce, 2000) com o objetivo de exportar e importar conjuntos de dados na forma de tabelas de um banco de dados, com uma grande variedade de sistemas gerenciadores de bancos de dados, tais como Oracle™ (Koch & Loney, 1997), Informix™ (Flannery, 2000), Sybase™ (Rankins, Garbus, Solomon & McEwan, 1996) e MySQL™ (Axmark, Widenius & DuBois, 2000);

Filtro de exemplos

A biblioteca de classes DOL oferece um sistema de filtros que permite ocultar temporariamente parte dos exemplos de um conjunto de dados. Uma vez que o filtro é removido, os exemplos voltam a compor o conjunto de dados. Os filtros podem ser criados a partir de expressões lógicas envolvendo atributos ou utilizando uma lista que indica explicitamente quais exemplos devem ser filtrados. A biblioteca permite que vários filtros sejam definidos e compostos. O sistema de filtragem de exemplos é, por exemplo, a base para a criação de um conjunto de classes que criam amostras aleatórias e estratificadas de um conjunto de exemplos;

Estatísticas descritivas e correlações

Estatísticas descritivas podem ser utilizadas tanto pelo usuário quanto por métodos de pré-processamento de dados. Pelo usuário, estatísticas descritivas podem ser utilizadas para compreender melhor os dados e ter uma visão geral sobre cada um dos atributos. Métodos de pré-processamento de dados podem utilizar estatísticas descritivas em suas implementações. Por exemplo, os métodos de normalização de dados necessitam de informações como média, máximo, mínimo e desvio padrão dos atributos. Ainda, informações sobre correlações entre atributos, como correlação linear e covariância são a base para a construção de diversos métodos de

[4] http://dbi.perl.org. DBI é um módulo de interface para bancos de dados para a linguagem Perl. Esse módulo define um conjunto de métodos, variáveis e convenções com o objetivo de prover uma interface para bancos de dados, independente do gerenciador de bancos de dados que está sendo utilizado.

pré-processamento de dados;

Métodos de reamostragem

Os métodos de reamostragem são métodos estatísticos capazes de estimar apropriadamente a taxa de erro verdadeira, mesmo com um conjunto reduzido de dados (Batista, 1997; Weiss & Kulikowski, 1991). Os métodos de reamostragem utilizam conjuntos de teste para estimar a taxa de erro verdadeira, os quais contém exemplos que não foram utilizados no treinamento do sistema de aprendizado. Diferentes métodos de reamostragem particionam de diferentes formas o conjunto de dados original em conjuntos de treinamento e teste. Atualmente, a biblioteca de classes DOL provê dois métodos de reamostragem: o *k-fold cross-validation* e o *k-fold cross validation estratificado* — Seção 4.3.3.2.

4.3.1 O Desenvolvimento da Biblioteca de Classes DOL

A biblioteca de classes DOL foi desenvolvida utilizando o modelo de protótipos. Os protótipos foram desenvolvidos, avaliados e melhorados até o projeto final, o qual é apresentado neste capítulo. De uma forma geral, o desenvolvimento do projeto da biblioteca de classes DOL pode ser dividido em duas fases:

Arquitetura monolítica

Nessa primeira fase, o projeto da biblioteca de classes consistia em uma *arquitetura monolítica*. A arquitetura é dita monolítica uma vez que o projeto se caracterizava por ter uma única classe principal que era responsável por prover uma grande gama de funcionalidades. Esse projeto inicial da biblioteca de classes primava pelo desempenho, mas a flexibilidade do projeto, principalmente em termos de extensibilidade, era limitada. A classe principal era complexa, com um grande número de métodos. Como todos os métodos da classe podiam acessar uma estrutura de dados principal, o desempenho da biblioteca era otimizado. Por outro lado, o projeto fornecia poucas facilidades para os usuários que desejassem adicionar funcionalidades à biblioteca. A adição de novas funcionalidades à biblioteca ficava limitada a criar classes derivadas da classe principal. Entretanto, a derivação é uma forma relativamente complexa para estender as funcionalidades de uma classe, pois quase sempre requer um grande conhecimento de como a classe a ser derivada funciona internamente. Reuso utilizando herança é muitas vezes chamado de *reuso caixa branca*, uma vez que os detalhes internos da classe pai ficam freqüentemente visíveis às classes filhas, ou seja, a "*herança quebra o encapsulamento*" (Snyder, 1986).

Arquitetura orientada a objeto e baseada em padrões de projeto

Nessa segunda fase, o projeto da biblioteca foi modificado para dar suporte a uma arquitetura mais sofisticada. Foram aplicados os conceitos de padrões de projeto de Engenharia de *Software* (Gamma, Helm, Johnson & Vlissides, 1995; Shalloway & Trott, 2002). Dessa forma, as grandes classes do projeto anterior deram espaço a um grupo maior de classes, cada uma delas com funcionalidades reduzidas e mais bem definidas. Além disso, a definição da interação entre as classes passou a ser um ponto chave no projeto da biblioteca. Criou-se um mecanismo de troca de mensagens entre as classes, para que as classes tomassem conhecimento sobre mudanças do estado interno de outras classes. Adicionar novas funcionalidades à biblioteca tornou-se mais simples, uma vez que essa tarefa pode ser feita ora por composição, também conhecida como *reuso caixa preta*, ora por derivação, dependendo da necessidade do usuário. Entretanto, mesmo quando a derivação é necessária, essa se torna mais simples, pois fica restrita à derivação de uma classe menor e mais simples. Além disso, normalmente, a derivação é restrita às classes e aos métodos abstratos, os quais foram projetados para serem especializados por derivação. Por outro lado, o alto grau de interação entre as classes causa uma degradação no desempenho se comparado à arquitetura monolítica previamente implementada. Mesmo apresentando um desempenho inferior, a arquitetura orientada a objeto foi escolhida pela sua facilidade em adicionar novas funcionalidades. Essa facilidade é um requisito importante, uma vez que a biblioteca DLE é utilizada no ambiente de pesquisa cujas áreas de pesquisa são altamente dinâmicas.

4.3.2 A Arquitetura da Biblioteca DOL

A biblioteca DOL possui uma arquitetura modular. Cada módulo é constituído de uma ou mais classes e realiza um conjunto bem definido de tarefas. Existe um módulo central, chamado Core, o qual carrega um conjunto de dados em uma estrutura e disponibiliza mais de 60 métodos capazes de consultar e manipular essa estrutura. Posteriormente, a estrutura pode ser salva em arquivos texto em diversas sintaxes ou carregada em bancos de dados relacionais.

O módulo Core é o único módulo que precisa ser carregado obrigatoriamente por uma aplicação que utiliza a biblioteca DOL. Os demais módulos podem ser carregados dependendo das necessidades do usuário. Cada módulo carregado estende a interface do módulo Core disponibilizando novas funcionalidades à aplicação do usuário.

Atualmente, a biblioteca DOL é constituída de 21 módulos implementados em 66 classes. Segue uma breve descrição de cada módulo:

Core O módulo Core é o maior módulo da biblioteca DOL. Como já foi descrito, o módulo Core é responsável por carregar um conjunto de dados em uma estrutura e prover uma gama de métodos capazes de realizar dezenas de operações sobre essa estrutura. Posteriormente, os dados na estrutura podem ser armazenados em mais de uma dezena de sintaxes diferentes utilizadas por alguns dos principais sistemas de aprendizado existentes;

Filter

O módulo Filter estende a interface do módulo Core, provendo métodos que permitem filtrar exemplos por meio de expressões lógicas e aritméticas que envolvem atributos. Os exemplos filtrados não são removidos a princípio, e podem voltar a integrar o conjunto de dados caso o usuário deseje. Ainda, os exemplos filtrados podem ser removidos fisicamente, caso esses exemplos não sejam mais necessários;

BasicStats e Correlation

Esses módulos disponibilizam métodos capazes de calcular estatísticas descritivas básicas como mínimo, máximo, média, variância e desvio padrão, além de outras sete estatísticas (módulo BasicStats) e, também, índices de correlação linear e covariância (módulo Correlation);

Shuffle

Shuffle é um módulo capaz de embaralhar a ordem na qual os exemplos armazenados em uma instância do módulo Core estão disponíveis. Esse módulo é utilizado, por exemplo, para garantir que a ordem dos exemplos não irá interferir nos conjuntos de treinamento e teste criados por métodos de reamostragem;

ResamplingKFoldCV e ResamplingStratKFoldCV

Esses dois módulos são utilizados para estimar o erro de classificação de um sistema de aprendizado sobre um determinado conjunto de dados. O conjunto de dados é dividido em diversos conjuntos de treinamento, teste e, opcionalmente, validação, segundo o método estatístico de reamostragem *k-fold cross-validation*. O módulo ResamplingStratKFoldCV é semelhante, entretanto, esse módulo realiza um *k-fold cross-validation estratificado* sobre o atributo classe;

SampleRandom

Esse módulo cria uma amostra aleatória de um conjunto de dados. Essa amostra pode ser utilizada, por exemplo, para reduzir o tamanho do conjunto de treinamento, quando não existe poder computacional suficiente para processar todos dos dados;

NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD

Esses três módulos realizam normalização de dados, ou seja, transformam os valores de um certo atributo que estão em uma determinada faixa de valores para outra faixa de valores. Esses módulos de normalização utilizam diferentes funções de normalização. O primeiro aplica uma transformação linear nos dados. O segundo utiliza o desvio-padrão de um atributo, e transforma os dados desse atributo em *z scores*. O terceiro inicialmente transforma os dados em *z scores* e, posteriormente, aplica uma transformação linear (Masters, 1993, Capítulo 16);

DistanceHEOM e DistanceHVDM

Os módulos DistanceHEOM e DistanceHVDM implementam funções de distância. O primeiro módulo implementa a função de distancia *Heterogeneous Euclidean-Overlap Metric* — HEOM e o segundo a função de distância *Heterogeneous Value Difference Metric* — HVDM (Wilson & Martinez, 2000; Batista & Monard, 2003c);

MTreeRandom e MTreeMST

M-tree é uma estrutura de índice que organiza os exemplos de um conjunto de dados em um "espaço métrico". As M-trees são capazes de realizar buscas por similaridade, como por exemplo, a busca pelos exemplos mais semelhantes a um dado exemplo, ou seja, uma busca do tipo K-VIZINHOS MAIS PRÓXIMOS. As M-trees são brevemente explicadas na Seção 5.5.1.4 na página 105. Uma explicação mais detalhada, incluindo os algoritmos para realizar a inserção e busca em M-Trees é descrito em (Ciaccia, Patella & Zezula, 1997);

kNN e kNNMTree

Esses dois módulos implementam o algoritmo K-VIZINHOS MAIS PRÓXIMOS. O primeiro módulo realiza a busca pelos vizinhos mais próximos fazendo uma passagem completa por todos os exemplos. O segundo módulo utiliza uma estrutura M-tree para acelerar a busca pelos exemplos mais similares;

Tomek

O módulo Tomek fornece métodos capazes de identificar pares de exemplos que formam *ligações Tomek* (Tomek, 1976). As ligações Tomek podem ser úteis na identificação de ruído e exemplos de borda nos dados (Kubat & Matwin, 1997; Batista, Carvalho & Monard, 2000). Ligações Tomek podem ser utilizadas para *balancear* um conjunto de dados rotulados que possui uma grande desproporção entre o número de exemplos de cada classe, como é discutido no Capítulo 6 na página 141;

Imputation

Esse módulo disponibiliza alguns métodos de *imputação* (Batista & Monard, 2002, 2003c) para o tratamento de valores desconhecidos. Entre os métodos implementados está o tratamento baseado na substituição dos valores desconhecidos pela média ou moda do atributo, e na substituição dos valores desconhecidos por valores preditos pelo algoritmo K-VIZINHOS MAIS PRÓXIMOS. No capítulo 5 na página 89 é discutida a utilização de métodos de imputação para o tratamento de valores desconhecidos;

Histogram e Scatter

Os módulos Histogram e Scatter utilizam a biblioteca Perl/Tk (Lidie & Walsh, 2002) para gerar gráficos do tipo histograma e *scatter*, respectivamente.

Ao menos do ponto de vista do usuário, cada módulo é independente dos demais, com exceção do módulo Core, do qual todos os demais módulos são dependentes. Na realidade, diversos módulos dependem de outros módulos para realizar uma determinada tarefa. Entretanto, sempre que esse fato ocorre, o módulo dependente é responsável por carregar os demais módulos sem que o usuário precise tomar conhecimento desse fato. Um exemplo é o módulo que implementa o método de reamostragem *k-fold cross-validation*. Esse módulo estende a interface do módulo Core, disponibilizando métodos para a criação de pares de conjuntos de treinamento e teste, com a possibilidade de criar também conjuntos de validação. O módulo ResamplingkFoldCV, o qual implementa esse método de reamostragem depende do módulo Shuffle, o qual embaralha o conjunto de dados. Como mencionado previamente, esse passo é necessário para se certificar de que a ordem dos exemplos no conjunto de dados não irá interferir nos conjuntos de treinamento e teste criados pelo método. O módulo Shuffle é carregado pelo módulo ResamplingkFoldCV sem o conhecimento do usuário, e a interface do módulo Shuffle não fica disponível à aplicação do usuário, como mostra a Figura 4.1 na página seguinte.

A carga de módulos realizada conforme as dependências entre os módulos, e sem o controle do usuário, pode levar à degradação de desempenho por excesso de *instâncias* carregadas dos mesmos módulos. Por exemplo, o módulo BasicStats é responsável por disponibilizar diversas estatísticas descritivas sobre um conjunto de dados carregado em Core. Diversos módulos dependem do módulo BasicStats, como por exemplo, os módulos de normalização de dados, e os módulos que implementam funções de distância entre exemplos. BasicStats realiza algumas passagens sobre os dados para calcular diversas estatísticas descritivas. A existência de diversas instâncias do módulo BasicStats executadas concorrentemente pode degradar o desempenho da aplicação do usuário, mesmo porque somente uma instância poderia atender todos os módulos que dependem desse módulo.

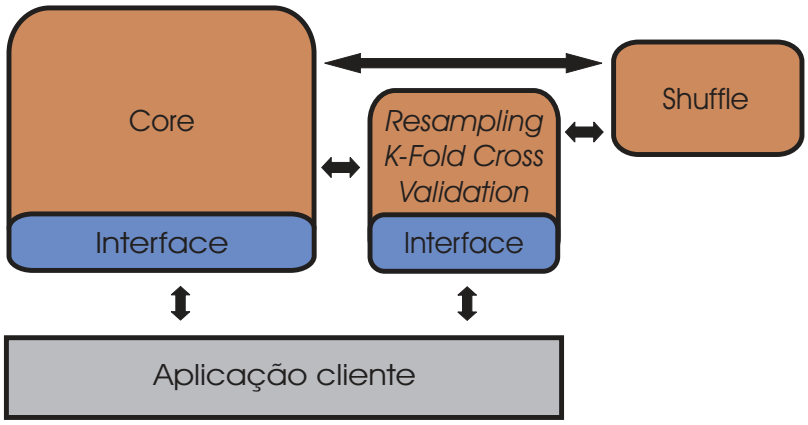

Figura 4.1: Exemplo de interação entre módulos da biblioteca DOL.

A biblioteca DOL possui um mecanismo de gerenciamento de instâncias de módulos que restringe a um o número de instâncias sendo executadas concorrentemente de determinados módulos ligados a uma mesma instância do módulo Core. Essa restrição de execução se aplica a todos os módulos cuja restrição não acarreta em perda de funcionalidades ou qualquer outro problema. O mecanismo de restrição de execução de instâncias foi projetado baseado no padrão de projeto *Singleton* (Shalloway & Trott, 2002, Capítulo 16) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 127–134).

Um outro problema intrínseco à estrutura modular da biblioteca DOL é que alterações no estado interno do módulo Core precisam ser notificadas a outros módulos. Por exemplo, o módulo BasicStats precisa ser informado de mudanças nos valores dos atributos para que as estatísticas que esse módulo provê sejam atualizadas[5]. Um outro exemplo é o módulo Filter, o qual cria filtros que removem temporariamente exemplos de um conjunto de dados. O módulo Filter deve ser informado, por exemplo, sobre a adição de um novo exemplo para que a sua estrutura de índice interna seja atualizada.

Para solucionar esse problema, a biblioteca DOL possui um sistema de envio de mensagens na qual o módulo Core avisa todos os módulos que necessitam ser informados de qualquer mudança em seu estado interno. O sistema de envio de mensagens da biblioteca DOL foi construído utilizando o padrão de projeto *Observer* (Shalloway & Trott, 2002, Capítulo 17) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 293–303). Existem diversas mensagens que podem ser enviadas pelo módulo Core, entre as principais estão alterações

[5]O usuário pode ajustar o módulo BasicStats para não atualizar as estatísticas a cada modificação nos dados gerenciados por um módulo Core. Nesse caso, as estatísticas podem ficar temporariamente desatualizadas. Fica a cargo do usuário requisitar diretamente ao módulo BasicStats que atualize as estatísticas.

nos valores de dados individuais, adição e remoção de atributos, adição ou remoção de exemplos, entre outras. Na Figura 4.2 é ilustrada a arquitetura do mecanismo de envio de mensagens da biblioteca DOL.

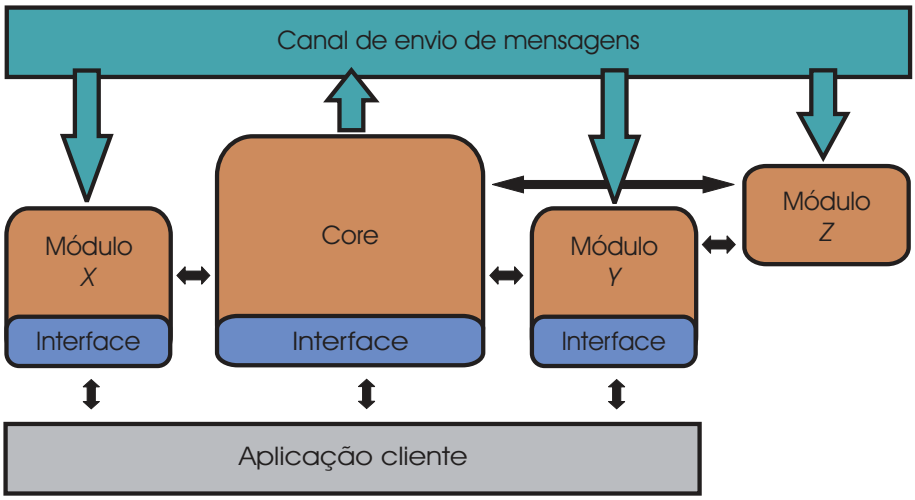

Figura 4.2: Arquitetura do mecanismo de envio de mensagens da biblioteca DOL.

4.3.3 O Projeto da Biblioteca DOL

Nesta seção é apresentada uma visão geral do projeto da biblioteca DOL. Apenas as classes principais são apresentadas, sem entrar em detalhes sobre a funcionalidade de cada método. Para ajudar na descrição dessas classes são utilizados diagramas de classe em *UML — The Unified Modelling Language* (Booch, Jacobson, Rumbaugh & Rumbaugh, 1998). Detalhes adicionais sobre o projeto da biblioteca DOL podem ser encontrados em (Batista & Monard, 2003d).

O projeto da biblioteca DOL utiliza padrões de projeto. Padrões de projeto são registros de experiência em projeto de programas orientados a objeto (Gamma, Helm, Johnson & Vlissides, 1995; Shalloway & Trott, 2002). Nenhum padrão de projeto pode ser considerado um padrão de projeto sem que tenha sido aplicado com sucesso em mais de um projeto diferente. O objetivo dos padrões de projeto é capturar a experiência em projetar sistemas existentes na comunidade e disponibilizar essa experiência de forma que outras pessoas possam utilizá-la efetivamente. Cada padrão de projeto fornece um nome, uma explicação e uma avaliação para soluções importantes e recorrentes em projetos de sistemas orientados a objeto. Padrões de projeto fazem com que o reuso de projetos e arquiteturas de sucesso se torne mais fácil. A expressão de técnicas comprovadas na

forma de padrões de projeto faz com que essas soluções se tornem mais acessíveis para os desenvolvedores de novos sistemas. Alguns padrões de projeto utilizados no projeto da biblioteca DOL são:

Bridge
: É um dos padrões de projeto mais importantes e, também, um dos mais difíceis de entender. É utilizado para separar um conjunto de implementações dos objetos que as utiliza (Shalloway & Trott, 2002, Capítulo 9) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 151–162);

Strategy
: Permite utilizar diferentes algoritmos dependendo do contexto em que eles ocorrem. Esse padrão permite que o algoritmo varie independentemente do cliente que o utiliza (Shalloway & Trott, 2002, Capítulo 14) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 315–324);

Observer
: Define uma dependência entre objetos de forma que quando um objeto muda de estado, todos os outros dependentes são notificados e atualizados automaticamente (Shalloway & Trott, 2002, Capítulo 17) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 293–303);

Singleton
: Certifica que uma classe possui somente uma instância, e provê um ponto de acesso global para essa instância (Shalloway & Trott, 2002, Capítulo 16) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 127–134);

Template
: Define o esqueleto de um algoritmo, delegando alguns passos do algoritmo para as sub-classes. Dessa forma, permite que os passos sejam redefinidos sem alterar a estrutura do algoritmo (Shalloway & Trott, 2002, Capítulo 18) (Gamma, Helm, Johnson & Vlissides, 1995, pp. 325–330).

Nas próximas seções são apresentados os projetos dos principais módulos da biblioteca DOL.

4.3.3.1 O Módulo Core

Como mencionado anteriormente, o módulo Core é o principal módulo da biblioteca de classes DOL. Esse módulo provê uma série de métodos capazes de fornecer ao usuá-

rio uma estrutura simples para acessar e manipular dados armazenados em uma tabela atributo-valor. Dados armazenados em uma instância do módulo Core podem ser gravados em diversas sintaxes utilizadas pelos principais sistemas de aprendizado, ou podem ser carregados em bancos de dados relacionais. Como informado previamente, o módulo Core disponibiliza atualmente mais de 60 métodos para acesso e manipulação de dados. Esse módulo é capaz de armazenar os dados em 15 diferentes sintaxes utilizadas por alguns dos sistemas de aprendizado mais conhecidos, além de poder carregar dados nos principais bancos de dados, tais como Oracle™, Sybase™, Informix™ e MySQL™.

Durante o projeto de módulo Core foram definidos alguns requisitos, entre os principais estão:

1. O módulo deve prover um conjunto mínimo de métodos capazes de realizar as principais manipulações sobre atributos e exemplos. Tarefas mais complexas devem ser implementadas por outros módulos, utilizando como base os métodos disponibilizados na interface do módulo Core;

2. O módulo deve ser capaz de suportar novos tipos de dado. Deve ser observado que os métodos utilizados em MD estão cada vez mais difundidos. Assim, é possível encontrar aplicações de MD em diversas áreas de conhecimento. Alguns pesquisadores têm trabalhado em adaptar os métodos utilizados em MD para diferentes domínios, dando origem a novas áreas de pesquisa como *Mineração de Textos — Text Mining* (Dörre, Gerstl & Seiffert, 1999), *Mineração de Dados Espaciais — Spatial Data Mining* (Roddick & Hornsby, 2000), e *Mineração de Dados Multimídia — Multimedia Mining* (Djeraba, 2003). Essas novas áreas de pesquisa podem vir a necessitar de novos tipos de dado, como por exemplo, os tipos *blob*[6], coordenada espacial, entre outros;

3. O módulo deve estar preparado para armazenar os dados tanto em memória principal quanto em memória auxiliar. O projeto inicial mantém os dados armazenados em memória principal, mas o módulo deve ser projetado de forma que, caso a aplicação requeira, esteja preparado para ser modificado para armazenar os dados em memória secundária. Ainda, o módulo deve ser capaz de armazenar alguns atributos em memória principal e outros atributos em memória auxiliar. Por exemplo, uma aplicação pode requerer a definição de um novo tipo de dados *blob*, o qual normalmente requer grande quantidade de memória. O módulo Core deve possuir

[6] *Binary Large Object*. São tipos de dado freqüentemente utilizados para armazenar vídeos, imagens e sons.

um projeto que permita armazenar esse atributo em memória auxiliar, enquanto que outros atributos que necessitam de uma quantidade menor de memória fiquem armazenados em memória principal;

4. O módulo deve ser capaz de ler arquivos de declaração de atributos e exemplos na sintaxe DSX[7] — DISCOVER DATASET SINTAX. O módulo Core está preparado para ser estendido e ler arquivos em outras sintaxes. Entretanto, é pouco provável que isso ocorra pois a sintaxe DSX possui um poder de representação que contempla as necessidades da grande maioria dos sistemas de aprendizado existentes atualmente. No futuro, a sintaxe DSX pode ser estendida para dar suporte a novos recursos que possam ser adicionados a novos sistemas de aprendizado;

5. O módulo deve ser capaz de gravar informações sobre atributos e dados em diversas sintaxes. Uma vez que novos sistemas de aprendizado serão propostos por pesquisadores da área, o módulo Core deve estar preparado para ser facilmente estendido para gravar dados nas sintaxes de sistemas de aprendizado que até mesmo ainda não foram propostos.

Na Figura 4.3 na próxima página é apresentado um diagrama de classes em UML do projeto do módulo Core. Esse módulo é constituído de 42 classes, sendo 11 delas *classes abstratas*. Uma classe abstrata tem como principal objetivo definir uma interface comum para as suas sub-classes. Uma classe abstrata delega a implementação de alguns ou todos os seus métodos para as suas sub-classes. De uma forma geral, a classe abstrata deve implementar tudo que há de comum entre as suas sub-classes, deixando em aberto somente as partes que variam. As partes da implementação que variam devem ser implementadas nas sub-classes conforme a necessidade de cada sub-classe. Por exemplo, a classe abstrata `TypeAbstract` define a estrutura básica que todos os tipos de dado possuem, enquanto que as classes abstratas `TypeQuantitative` e `TypeQualitative` estendem essa interface para dar suporte aos atributos quantitativos e qualitativos, respectivamente. Por fim, as demais classes realizam as implementações dos tipos específicos, por exemplo, a classe `TypeReal` implementa o armazenamento e gerenciamento de atributos do tipo de dado `real` e a classe `TypeNominal` do tipo de dado `nominal`.

[7] A sintaxe DSX é a sintaxe padrão para arquivos de declaração de dados e atributos do ambiente DISCOVER. Uma explicação detalhada sobre essa sintaxe pode ser encontrada no Apêndice A na página 169.

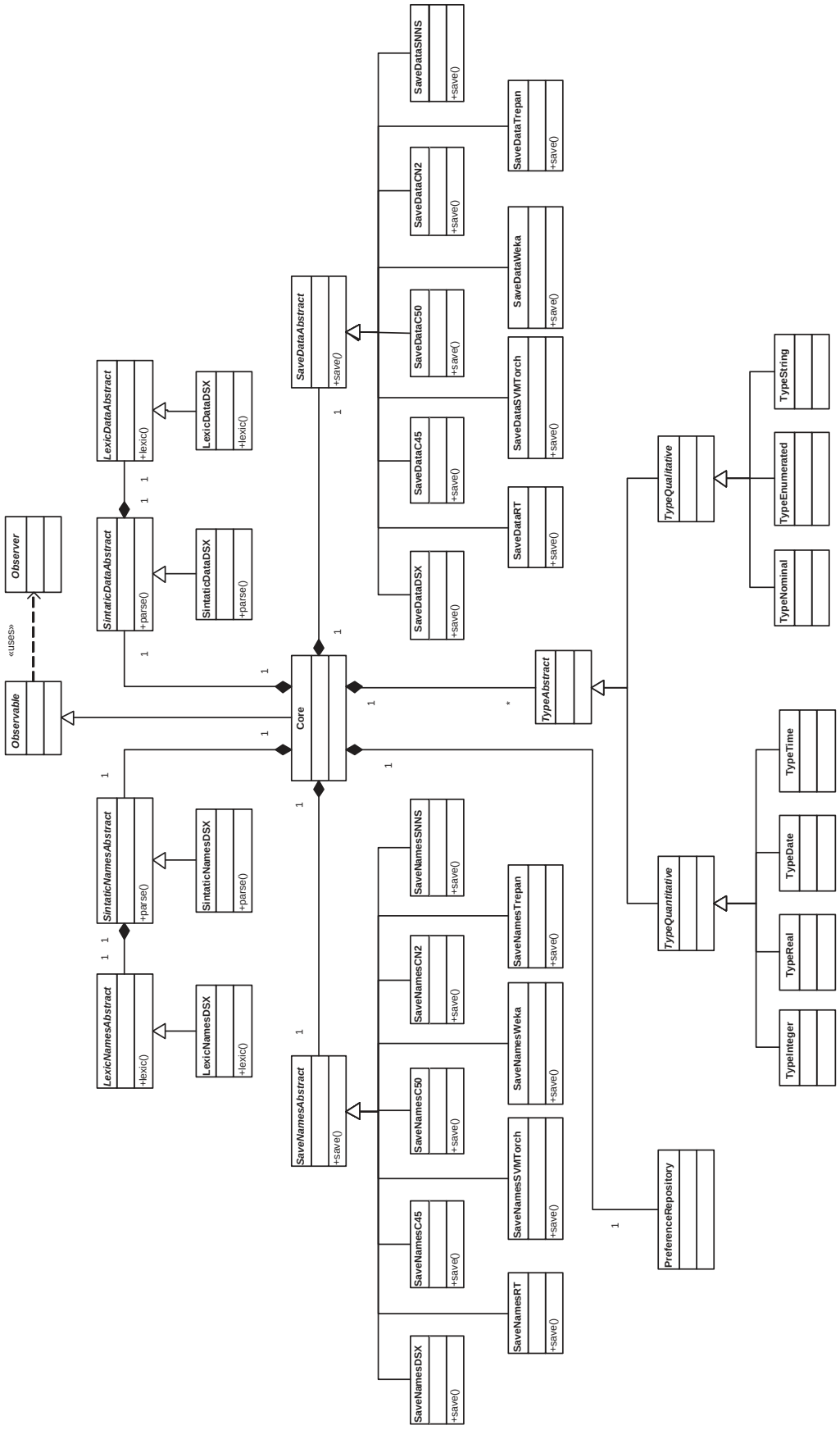

Figura 4.3: Diagrama de classes em UML do projeto do módulo Core.

A classe `Core` é a classe principal entre as classes apresentadas na Figura 4.3 na página anterior. Essa classe é responsável por definir e implementar a interface utilizada pelo usuário do módulo Core. Para isso, a classe `Core` é auxiliada pelas demais classes, as quais realizam tarefas específicas. As classes abstratas `SaveNamesAbstract` e `SaveDataAbstract` criam a estrutura para gravar os dados armazenados na classe `Core` em diferentes sintaxes. As classes abstratas `SintaticNamesAbstract` e `SintaticDataAbstract` são a base para a implementação de analisadores sintáticos para a leitura de arquivos de definição de atributos e dados, respectivamente. A classe `PreferenceRepository` mantém as preferências do usuário quanto ao comportamento do módulo Core e dos demais módulos da biblioteca DOL. A classe `PreferenceRepository` é um repositório central que gerencia a configuração de todos os módulos da biblioteca.

A classe `Core` é derivada da classe `Observable`. As classes `Observable` e `Observer` implementam o sistema de envio de mensagens que mantém as demais classes da biblioteca DOL informadas sobre mudanças no estado interno da classe `Core`. Qualquer classe derivada da classe `Observer` recebe mensagens sobre as alterações de estado da classe `Core`.

No módulo Core pode-se identificar diversos padrões de projeto que auxiliaram em seu projeto. Entre os principais, as classes `LexicAbstractNames` e `SintaticAbstractNames` e suas sub-classes, as quais implementam analisadores léxico e sintático, respectivamente, para arquivos de definição de atributos, seguem o padrão de projeto **Bridge**. O mesmo ocorre com as classes `LexicAbstractData` e `SintaticAbstractData` e suas sub-classes. As classes `Observable` e `Observer`, juntamente com as suas sub-classes, seguem o padrão de projeto **Observer**. As classes `SaveNamesAbstract` e `SaveDataAbstract` e suas sub-classes seguem o padrão de projeto **Strategy**.

4.3.3.2 Os Módulos ResaplingkFoldCV e ResamplingStratKFoldCV

A classe abstrata `ResamplingAbstract` cria a estrutura básica para a implementação de métodos estatísticos de reamostragem. Atualmente, dois métodos de reamostragem estão implementados, o *k-fold cross-validation* e o *k-fold cross-validation estratificado*, por meio das classes `ResamplingkFoldCV` e `ResamplingStratKFoldCV`, respectivamente. A classe `ResamplingAbstract` utiliza a classe `Shuffle` para embaralhar os exemplos antes de dividi-los em conjuntos de treinamento e teste, e a classe `SampleRandom` para remover do conjunto de treinamento uma amostra aleatória para a criação de um conjunto de validação. A criação de conjuntos de validação é opcional. Na Figura 4.4 é mostrado o

diagrama de classes em UML que ilustra essa interação entre as classes.

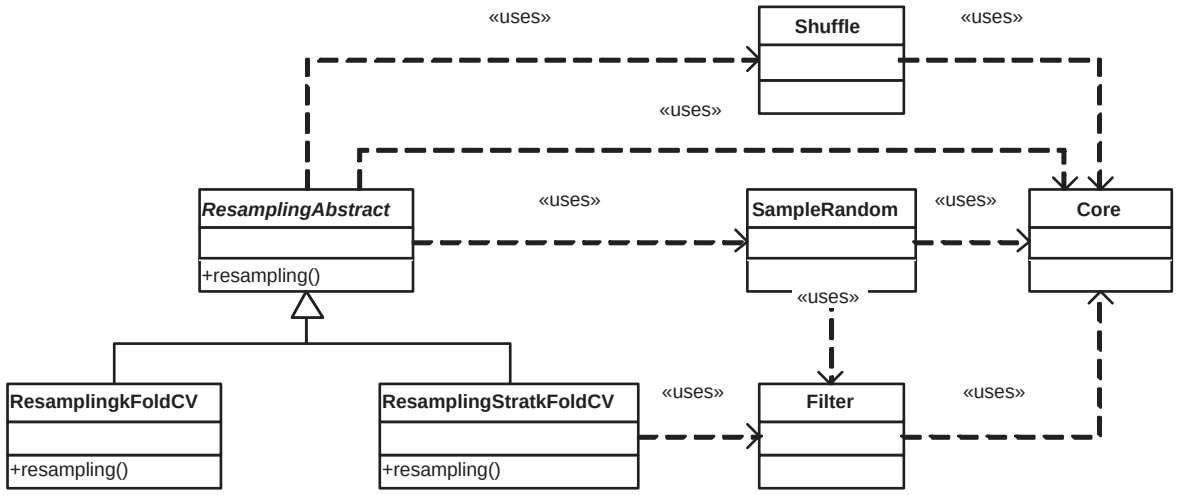

Figura 4.4: Diagrama de classes em UML do projeto dos módulos ResaplingkFoldCV e ResamplingStratKFoldCV.

O algoritmo para implementar o método *k-fold cross-validation* é descrito no Algoritmo 4.1.

Algoritmo 4.1 Algoritmo que divide um conjunto de exemplos em k pares de conjuntos de treinamento e teste segundo o método de reamostragem *k-fold cross-validation*.

Require: $E = \{E_1, E_2, \ldots E_N\}$, um conjunto de exemplos a ser dividido em conjuntos de treinamento e teste
k, número de pares de conjuntos de treinamento e teste

Ensure: (Tr_1,Te_1), (Tr_2,Te_2) $\ldots (Tr_k,Te_k)$ são k pares de conjuntos de treinamento e teste obtidos segundo o método *k-fold cross-validation*

1: Sejam os conjuntos $P_1, P_2, \ldots P_k$ partições aleatórias de E, com aproximadamente o mesmo número de exemplos, tal que se $E_l \in P_m$, então $E_l \notin P_n \; \forall \; n \neq m$
2: **for all** i tal que $1 \leq i \leq k$ **do**
3: $\quad Te_i = P_i$ {Conjunto de teste}
4: $\quad Tr_i = \bigcup_{j \neq i} P_j$ {Conjunto de treinamento}
5: **end for**
6: **return** (Tr_1, Te_1), (Tr_2,Te_2) $\ldots (Tr_k, Te_k)$

O método *k-fold cross-validation* estratificado se certifica que os conjuntos de treinamento e teste possuem, aproximadamente, a mesma proporção de exemplos para cada valor do atributo classe, se comparado com o conjunto de dados original. Para implementar esse método, a classe `ResamplingStratKFoldCV` utiliza a classe `Filter` para filtrar os exemplos segundo cada valor do atributo-classe. O algoritmo do método *k-fold cross-validation* estratificado é similar ao Algoritmo 4.1, entretanto, na linha 1 desse algoritmo

é necessário incluir a condição de que as partições $P_1, P_2, \ldots P_k$ devem possuir aproximadamente o mesmo número de exemplos de cada classe.

A classe `ResamplingAbstract` e as sub-classes `ResamplingkFoldCV` e `ResamplingStratkFoldCV` seguem o padrão de projeto **Strategy** para a implementação das variações dos algoritmos de reamostragem. Ainda, a classe `ResamplingAbstract` utiliza o padrão de projeto **Template** para definir a estrutura básica do algoritmo de reamostragem, o qual primeiro embaralha os exemplos e depois os divide em conjuntos de treinamento e teste segundo algum método específico e, por fim, caso o usuário deseje, separa uma amostra aleatória do conjunto de treinamento para ser utilizada como conjunto de validação.

4.3.3.3 Os Módulos DistanceHEOM e DistanceHVDM

A classe abstrata `DistanceAbstract` disponibiliza uma estrutura básica para a implementação de funções de distância entre exemplos. `DistanceAbstract` não assume que os exemplos estão normalizados, e utiliza a classe `BasicStats` para obter informações tais como máximo, mínimo e desvio padrão dos valores dos atributos para normalizá-los, caso assim o usuário deseje.

Atualmente, `DistanceAbstract` possui duas sub-classes, DistanceHEOM e DistanceHVDM, as quais implementam as funções de distância *Heterogeneous Euclidean-Overlap Metric* — HEOM e *Heterogeneous Value Difference Metric* — HVDM, repectivamente (Wilson & Martinez, 2000; Batista & Monard, 2003c). A função de distância HEOM utiliza a distância euclidiana para atributos quantitativos e a distância *overlap* para atributos nominais. A função de distância HVDM utiliza a distância euclidiana para atributos quantitativos e a distância VDM (Stanfill & Waltz, 1986) para atributos qualitativos. Na Seção 5.5.1.3 na página 101 as funções de distância HEOM, VDM e HVDM são discutidas mais detalhadamente.

Na Figura 4.5 na página oposta é mostrado um diagrama de classes UML que ilustra o projeto dos módulos DistanceHEOM e DistanceHVDM.

A classe `DistanceAbstract` utiliza o padrão de projeto **Strategy** para implementar as diferentes funções de distância. As classes `BasicStats` e `DistanceAbstract` são derivadas da classe `Observer`. Dessa forma, elas utilizam o padrão de projeto **Observer** para observar modificações no estado interno da classe `Core`.

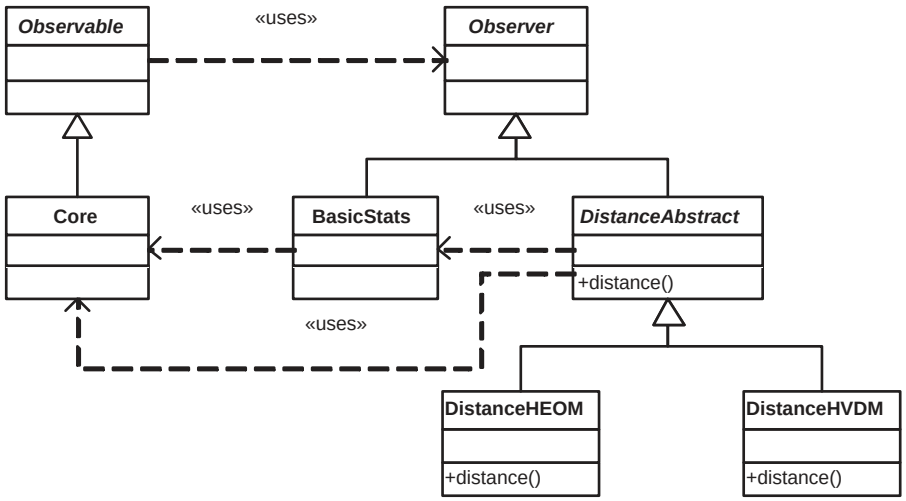

Figura 4.5: Diagrama de classes em UML do projeto dos módulos DistanceHEOM e DistanceHVDM.

4.3.3.4 Os Módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD

Os módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD implementam três funções de normalização de dados. O módulo NormalizeLinear normaliza um atributo A_r utilizando um mapeamento linear simples, no qual os valores extremos do atributo A_r, denotados por max_{A_r} e min_{A_r}, com $max_{A_r} > min_{A_r}$, são mapeados em dois valores definidos pelo usuário max'_{A_r} e min'_{A_r}, sendo $max'_{A_r} \geq min'_{A_r}$. Dado um valor x_{ir} do atributo A_r, seu valor normalizado x'_{ir} é dado pela Equação 4.2.

$$x'_{ir} = R(x_{ir} - min_{A_r}) + min'_{A_r} \tag{4.2}$$

na qual

$$R = \frac{max'_{A_r} - min'_{A_r}}{max_{A_r} - min_{A_r}} \tag{4.3}$$

Se o atributo é unimodal, possui uma distribuição quase simétrica em torno da média, e possui poucos ou nenhum valor extremo distante da média, então uma forma mais sofisticada de normalizar tal atributo é com o uso da média e do desvio padrão. O módulo NormalizeSimpleSD realiza essa normalização, na qual os valores do atributo são convertidos para *z scores*. Seja $\hat{\mu}_{A_r}$ e $\hat{\sigma}_{A_r}$ a média e o desvio padrão estimados dos valores do atributo A_r no conjunto de dados, respectivamente. A normalização realizada pelo

módulo NormalizaSimpleSD para um valor x_{ir} do atributo A_r é definida pela Equação 4.4.

$$x'_{ir} = \frac{x_{ir} - \hat{\mu}_{A_r}}{\hat{\sigma}_{A_r}} \tag{4.4}$$

Se um atributo normalmente distribuído é normalizado conforme a Equação 4.4, então é esperado que 95% dos valores desse atributo estejam no intervalo $[-1.96, 1.96]$, e que 99% dos valores estejam no intervalo $[-2.58, 2.58]$.

É possível aplicar as duas normalizações definidas pelas Equações 4.2 e 4.4 em conjunto. Primeiro, o atributo é convertido para *z scores* e, em seguida, os valores são mapeados para uma faixa de valores definida pelo usuário. O módulo NormalizeScalledSD realiza essa normalização em um único estágio. Essa normalização é definida pela Equação 4.5.

$$x'_{ir} = R\left(\frac{x_{ir} - \hat{\mu}_{A_r}}{\hat{\sigma}_{A_r}} - min_{A_r}\right) + min'_{A_r} \tag{4.5}$$

Os módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD utilizam a classe `BasicStats` para obter estatísticas sobre os atributos, tais como média, desvio padrão e valores mínimo e máximo. As classes `NormalizeLinear`, `NormalizeSimpleSD` e `NormalizeScalledSD` implementam as três normalizações realizadas pelos módulos homônimos. `NormalizeAbstract` é a classe abstrata que implementa a estrutura comum entre todos os métodos de normalização. Novos métodos de normalização podem ser adicionados à biblioteca DOL derivando a classe `NormalizeAbstract`, e implementando os métodos abstratos `normalize` e `denormalize`, os quais normalizam e desnormalizam um valor de um atributo, respectivamente. Na Figura 4.6 na próxima página é ilustrada a interação entre as classes dos módulos de normalização.

4.3.3.5 Os Módulos kNNMTree, kNNLinear, MTreeRandom e MTreeMST

A classe `kNNAbstract` cria a estrutura básica para a implementação de algoritmos de aprendizado baseados no método K-VIZINHOS MAIS PRÓXIMOS. `kNNAbstract` utiliza a classe `DistanceAbstract` para os cálculos de distância entre os exemplos. Essa relação entre essas duas classes abstratas permite que qualquer sub-classe de `kNNAbstract` (ou seja, qualquer implementação de métodos baseados no algoritmo K-VIZINHOS MAIS PRÓXIMOS) possa utilizar qualquer medida de distância definida por meio da classe `DistanceAbstract`.

Atualmente, a biblioteca DOL disponibiliza duas implementações do algoritmo K-

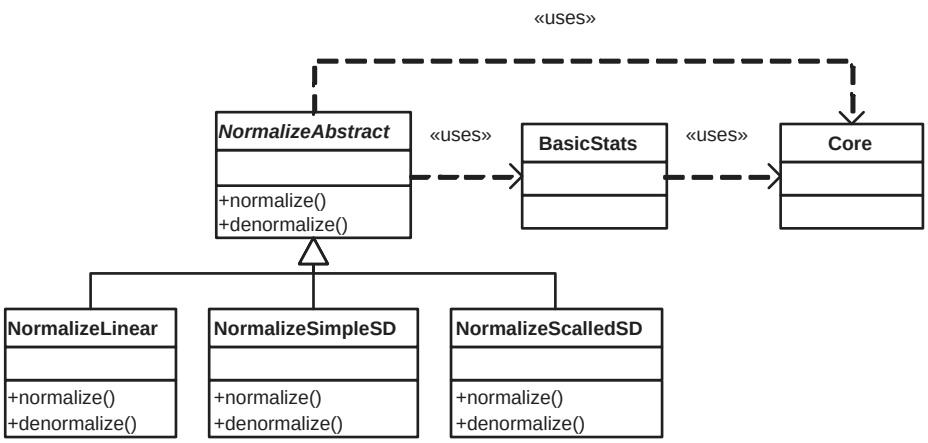

Figura 4.6: Diagrama de classes em UML do projeto dos módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD.

VIZINHOS MAIS PRÓXIMOS. A primeira, implementada por meio da classe `kNNLinear`, faz uma passagem por todos os exemplos para identificar os k exemplos mais similares ao exemplo de consulta. A segunda, implementada pela classe `kNNMTree`, utiliza um método de acesso chamado M-tree (Ciaccia, Patella & Zezula, 1997) para acelerar a busca pelos exemplos mais semelhantes.

Como mencionado previamente, as estruturas M-tree organizam os exemplos em um espaço métrico, o qual permite realizar buscas por similaridade com um número reduzido de cálculos de distância. A classe `MTreeAbstract` implementa os algoritmos de construção e busca em estruturas M-trees, deixando em aberto a implementação de dois métodos responsáveis pela divisão de um nó quando esse nó está cheio. As sub-classes `MTreeRandom` e `MTreeMST` implementam duas alternativas para esses dois métodos. A primeira é a escolha de dois exemplos a serem promovidos para o nó pai de forma aleatória, e a divisão dos exemplos restantes em dois nós considerando a distância entre cada exemplo e os exemplos promovidos. A segunda alternativa utiliza o algoritmo *Minimal Spanning Trees* — MST para realizar a escolha dos elementos a serem promovidos e a divisão dos elementos restantes. A segunda implementação está baseada na estrutura *Slim Trees* (Jr., Traina, Seeger & Faloutsos, 2000).

Na Figura 4.7 na página seguinte é mostrado o diagrama de classes em UML das classes descritas anteriormente.

Nos módulos kNNMTree, kNNLinear, MTreeRandom e MTreeMST foram utilizados diversos padrões de projeto. As classes `kNNAbstract` e suas sub-classes e a classe `DistanceAbstract` fazem uso do padrão de projeto **Bridge**. Ainda, as classes `MTreeAbstract`

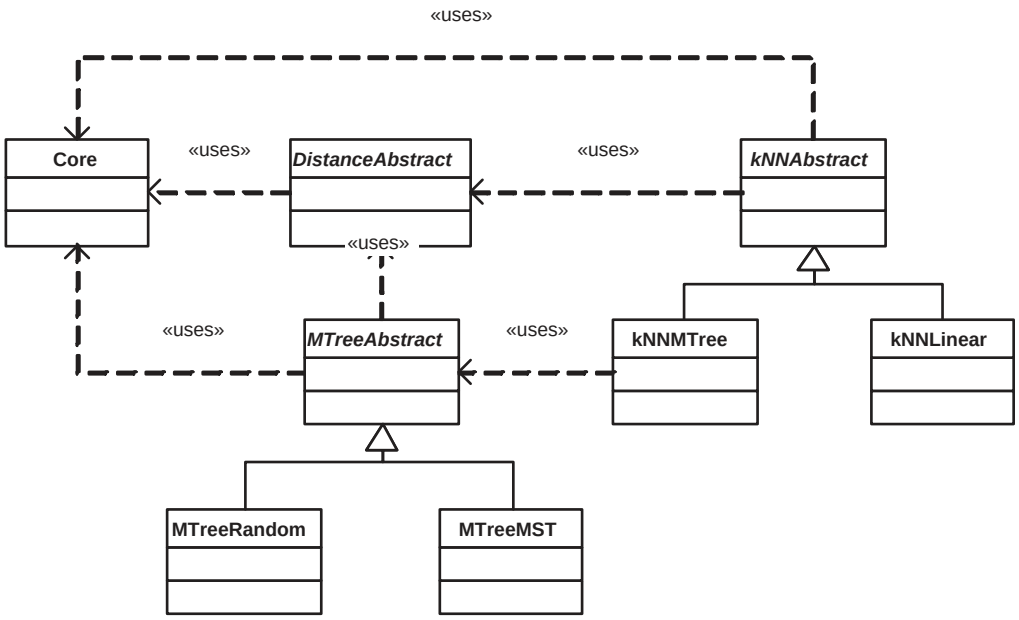

Figura 4.7: Diagrama de classes em UML do projeto dos módulos kNNMTree, kNNLinear, MTreeRandom e MTreeMST.

e `kNNAbstract` utilizam o padrão de projeto **Strategy** para implementar as variações de seus algoritmos. Por fim, a classe `MTreeAbstract` é derivada da classe `Observer` (essa relação não é mostrada no diagrama), portanto, essa classe utiliza o padrão de projeto **Observer** para observar as alterações no estado interno da classe `Core`.

4.3.4 Outras Soluções para a Implementação da Biblioteca DOL

Em AM, a grande maioria dos sistemas de aprendizado é capaz de induzir um conceito a partir de uma tabela atributo-valor. As tabelas atributo-valor são tabelas muito similares às tabelas armazenadas e manipuladas pelos *Sistemas Gerenciadores de Banco de Dados* — SGBDs. Uma alternativa à construção da biblioteca DOL é armazenar os dados em um SGBD e pré-processá-los utilizando os recursos oferecidos pelos SGBDs. Essa solução possui diversos argumentos a favor, e outros contra. Entre os principais argumentos a favor pode-se relacionar:

1. Os SGBDs são sistemas cujo desempenho é uma das preocupações centrais. Com a recente aplicação de sistemas de aprendizado em grandes bases de dados, os métodos de pré-processamento de dados, quando implementados nos SGBDs, podem obter proveito da performance desses sistemas no processamento de grandes volumes de

dados;

2. Diversas empresas que comercializam SGBDs, como a Microsoft™ e a Oracle™, têm mostrado interesse em integrar os seus SGBDs a algoritmos de aprendizado. Nesse sentido, integrar métodos de pré-processamento de dados aos SGBDs pode obter proveito dessa tendência, uma vez que novos algoritmos de aprendizado poderão estar integrados aos SGBDs;

3. A grande maioria dos SGBDs relacionais utilizam a linguagem *SQL*[8] para manipulação de dados. A linguagem SQL é padronizada (Silberschatz, Korth & Sudarchan, 1997), embora a maioria das empresas que comercializam SGBDs ofereçam versões estendidas dessa linguagem. Essa padronização pode fazer com que um sistema de pré-processamento de dados baseado em SQL padrão possa ser executado em diferentes SGBDs;

4. Existem alguns excelentes SGBDs disponíveis gratuitamente na Internet como, por exemplo, o MySQL™ (Axmark, Widenius & DuBois, 2000). Esse fato permite implementar um projeto de pesquisa sem que esse projeto seja dependente de um SGBD comercial. Dessa forma, ferramentas implementadas em um projeto de pré-processamento de dados sobre um SBGD gratuito poderiam ficar disponibilizadas livremente para uso de outros pesquisadores.

Por outro lado, existem diversos argumentos contra, entre os principais pode-se relacionar:

1. Os SGBDs são sistemas complexos. Normalmente são sistemas difíceis de instalar, configurar, utilizar e gerenciar. Dessa forma, o sistema de pré-processamento de dados pode se tornar um sistema de uso difícil. Usuários que precisam de pequenas tarefas de pré-processamento como, por exemplo, uma normalização de dados, precisariam instalar e configurar um banco de dados para realizar esse pré-processamento;

2. A linguagem SQL é uma linguagem projetada para realizar consultas e atualizações em bancos de dados. Normalmente, as tarefas de pré-processamento de dados são bem mais complexas que a maioria das consultas padrão realizadas em SQL. Isso significa que muitas das tarefas de pré-processamento de dados somente podem ser convertidas em consultas SQL extremamente complexas. Na realidade, nem todas as tarefas de pré-processamento de dados podem ser expressas em SQL, uma

[8] *Structured Query Language.*

vez que SQL não possui todo o poder de expressão de uma linguagem procedural (Silberschatz, Korth & Sudarchan, 1997, pp. 142). Dessa forma, um sistema de pré-processamento de dados criado sobre SQL seria muito complexo, além de ser necessário utilizar outra técnica capaz de substituir a linguagem SQL quando essa linguagem não fosse expressiva o suficiente para implementar uma determinada tarefa de pré-processamento de dados;

3. Muitas empresas que comercializam SGBDs definiram linguagens que mesclam SQL e comandos procedurais como, por exemplo, a linguagem PL/SQL™ da Oracle. Entretanto, essas linguagens carecem de uma padronização. A adoção de uma dessas linguagens faria com que o sistema de pré-processamento de dados ficasse dependente de um SGBD específico;

4. Existe ainda, a possibilidade de utilizar uma linguagem de programação procedural como C++ (Stroustrup, 1997), Java (Horstmann & Cornell, 1997) ou Perl (Wall, Christiansen & Schwartz, 1996) com comandos SQL embutidos. Nessa solução, a aplicação cliente escrita em linguagem procedural utiliza comandos SQL para se comunicar com o banco de dados. Os resultados dos comandos SQL são retornados na forma de uma tabela, a qual a aplicação cliente pode ter acesso de diversas formas como, por exemplo, realizando um *fetch* registro a registro. O maior problema dessa solução é que a maior parte do processamento normalmente fica com a aplicação cliente, e assim não usufrui-se das funcionalidades de um SGBD. Para que uma tarefa de pré-processamento de dados, a qual fosse demasiadamente complexa para ser implementada somente em SQL, seja realizada, seria necessário transmitir os dados do SGBD para a aplicação cliente, realizar o pré-processamento dos dados, e voltar a enviar os dados pré-processados de volta ao SGBD. Infelizmente, muitas das tarefas de pré-processamento de dados envolvem uma ou mais passagens por todos os registros, até mesmo as tarefas mais simples como calcular estatísticas descritivas e normalizar dados. Essa solução acaba fazendo do SGBD um mero repositório de dados, com pouco uso na manipulação de dados.

Concluindo, embora SQL seja uma linguagem amplamente utilizada, pré-processamento de dados possui necessidades específicas, as quais nem sempre são simples de serem codificadas diretamente em SQL. Esse fato pode fazer com que o sistema de pré-processamento de dados se torne excessivamente complexo. Certamente, muito do futuro dos sistemas de aprendizado reside na integração com SGBDs. Esse é o principal fator

que motiva a integração da biblioteca de classes DOL com SGBDs, como descrito nesta seção.

4.4 O Ambiente Computacional SNIFFER

Como mencionado na Seção 4.2 na página 49, o segundo módulo do ambiente computacional DLE implementa o ambiente computacional SNIFFER para gerenciamento de avaliações e comparações experimentais de algoritmos de aprendizado. Análises experimentais são extremamente importantes na área de AM porque os métodos empregados e os dados a serem analisados são, normalmente, muito complexos para um tratamento formal completo. Em outras palavras, para um dado problema não existem instrumentos formais para decidir qual método é ótimo (Kibler & Langley, 1988; Dietterich, 1997a; Schaffer, 1994). Mesmo existindo uma vasta literatura em Estatística, Teoria de Aprendizado Computacional e outras áreas afins, a última palavra na decisão de qual método é o melhor para um determinado problema é sempre dada por uma avaliação experimental, como em qualquer outra ciência que precisa de avaliações de suas teorias.

Como descrito anteriormente, o ambiente computacional SNIFFER automatiza a avaliação experimental, e é totalmente integrado com diversos sistemas de aprendizado, tais como C4.5 (Quinlan, 1988), C4.5 RULES (Quinlan, 1987a), ID3 (Quinlan, 1986), CN2 (Clark & Boswell, 1991) e NEWID (Boswell, 1990). Com isso, é possível comparar o desempenho de novos métodos de pré-processamento de dados aplicados a diversos sistemas de aprendizado. As comparações são realizadas utilizando testes estatísticos de significância, os quais permitem determinar quando um método é capaz de superar seu concorrente com 95% ou 99% de confiança.

O ambiente SNIFFER complementa a biblioteca DOL pois permite que métodos de pré-processamento de dados desenvolvidos sejam avaliados e comparados experimentalmente de uma forma rápida e segura.

Freqüentemente, para analisar o desempenho de um método é necessário executá-lo diversas vezes. Primeiro, porque a repetição dos experimentos, como nos métodos de reamostragem, fornece a variância dos resultados; segundo, a maior parte dos métodos possui parâmetros que precisam ser ajustados para cada conjunto de dados; e, finalmente, para que se tenha uma compreensão melhor do desempenho dos métodos analisados, esses métodos são normalmente executados em diversos conjuntos de exemplos. Dessa forma, é bastante comum que determinado método seja executado centenas de vezes.

Por exemplo, os experimentos realizados sobre tratamento de valores desconhecidos descritos no Capítulo 5 na página 89 foram realizados utilizando *10-fold cross-validation*, 7 proporções de valores desconhecidos (0%, 10%, 20%, 30%, 40%, 50% e 60%), 8 valores para o parâmetro k do algoritmo K-VIZINHOS MAIS PRÓXIMOS (1, 3, 5, 10, 20, 30, 50, 100), e experimentos executados com 2 sistemas de aprendizado diferentes (C4.5 e CN2). Portanto, no total, esses dois sistemas de aprendizado foram executados $10 \times 7 \times 8 \times 2 = 1120$ vezes para cada conjunto de dados.

Diante de tantas execuções, torna-se muito difícil gerenciar os experimentos manualmente, sendo necessário a utilização de um ambiente computacional que realize essa tarefa. Por esse motivo, o ambiente computacional SNIFFER foi projetado. Esse ambiente computacional foi desenvolvido neste trabalho para solucionar os seguintes problemas:

Gerenciamento de sintaxes dos sistemas de aprendizado
: Em experimentos que envolvem execuções de diferentes sistemas de aprendizado, é comum que cada sistema utilize sintaxes diferentes para os arquivos de declaração de dados e atributos. O ambiente SNIFFER converte, quando necessário, a sintaxe do conjunto de dados original para a sintaxe do sistema de aprendizado antes de executá-lo;

Aplicação de reamostragem para estimar a taxa de erro verdadeira
: Os conjuntos de dados devem ser divididos em conjuntos de treinamento e teste para que a taxa de erro verdadeira seja estimada no conjunto de teste. O ambiente SNIFFER realiza essa tarefa, desde que solicitado pelo usuário. O usuário pode também escolher qual método de reamostragem deve ser aplicado ao conjunto de dados;

Recuperação das taxas de erro
: Os principais sistemas de aprendizado aceitam conjuntos de treinamento e teste para a criação do modelo e a estimativa da taxa de erro verdadeira, respectivamente. Normalmente, os sistemas de aprendizado gravam a matriz de confusão[9] em um arquivo texto com sintaxe proprietária. O ambiente SNIFFER recupera a matriz de confusão no conjunto de teste, e a armazena para posterior análise dos resultados;

Cálculo de medidas de desempenho
: Recuperadas as matrizes de confusão para cada iteração do método de reamostragem, é necessário calcular estatísticas que forneçam um indicativo do desempenho do sistema de aprendizado no conjunto de dados analisado. O ambiente SNIFFER

[9] A matriz de confusão apresenta o erro cometido pelo classificador para cada um dos valores do atributo classe.

calcula medidas de desempenho para cada valor do atributo classe e, também, o desempenho geral levando em consideração todos os valores do atributo classe juntos;

Comparação de medidas de desempenho

Uma vez que as medidas de desempenho foram calculadas é necessário compará-las para verificar se existem diferenças significativas entre elas, ou seja, se um método superou outro com uma diferença de 95% ou 99% de confiabilidade.

Um outro objetivo importante do ambiente SNIFFER é automatizar, sempre que possível, a publicação dos resultados. Dessa forma, o ambiente fornece ao usuário uma segurança maior de que os resultados publicados são fiéis aos resultados obtidos nos experimentos. Como, freqüentemente, os resultados dos experimentos envolvem uma grande quantidades de valores numéricos, é muito comum a introdução de erros durante o processo de confecção de tabelas e gráficos. O ambiente SNIFFER fornece ao usuário relatórios resumidos e detalhados dos resultados obtidos, além de tabular os resultados em um formato que pode ser utilizado para gerar gráficos no utilitário Gnuplot[10] (Crawford, 1998), e tabelas no processador de textos LaTeX[11] (Kopla & Daly, 1999).

Nas próximas seções são discutidos o funcionamento do ambiente computacional SNIFFER e a arquitetura desse ambiente computacional.

4.4.1 O Funcionamento do Ambiente Computacional SNIFFER

Para organizar um experimento, o ambiente computacional SNIFFER utiliza a estrutura do sistema de arquivos do sistema operacional no qual está sendo executado. Os conjuntos de dados a serem analisados são dispostos em diretórios, sendo que alguns identificadores de diretórios possuem significado especial para o ambiente.

A seguir é utilizado um exemplo para tornar mais simples as explicações sobre o funcionamento do ambiente. Vamos imaginar que desejamos comparar o desempenho de três sistemas de aprendizado: C4.5, C4.5 RULES e CN2 sobre quatro conjuntos de dados: **Breast**, **Bupa**, **CMC** e **Pima**.

Inicialmente, o usuário deve organizar os conjuntos de dados em diretórios. A forma que os diretórios devem ser organizados depende de como o usuário deseja que os testes de hipótese sejam feitos. Por hora, vamos pensar que os conjuntos de dados foram organizados pelo usuário da forma em que são mostrados na Figura 4.8 na próxima página.

[10] http://www.gnuplot.info.
[11] http://www.latex-project.org.

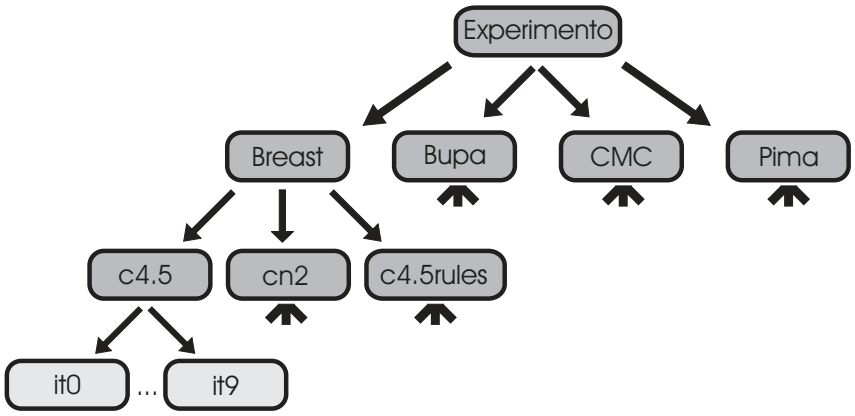

Figura 4.8: Exemplo de experimento organizado em diretórios para o ambiente SNIFFER.

Existe um diretório raiz para o experimento chamado `Experimento`. Dentro desse diretório, o qual engloba todo o experimento, foram criados quatro diretórios: `Breast`, `Bupa`, `CMC`, e `Pima`, ou seja, um para cada conjunto de dados que será analisado. Dentro de cada diretório dos conjuntos de dados foram criados outros três diretórios: `c4.5`, `cn2` e `c4.5rules` um para cada sistema de aprendizado a ser empregado. Na Figura 4.8 são mostrados somente os diretórios contidos no diretório `Breast`, os demais foram omitidos por simplificação. For fim, opcionalmente, o usuário pode criar diretórios numerados que seguem o padrão `it0`, `it1`, ... `it(k-1)`, um para cada iteração do método de reamostragem, sendo k o número total de iterações do método, como explicado mais à frente.

Identificador	Sistema de Aprendizado
`c4.5`	C4.5 (Quinlan, 1988)
`c4.5rules`	C4.5 RULES (Quinlan, 1987a)
`id3`	ID3 (Quinlan, 1986)
`cn2`	CN2 (Clark & Boswell, 1991)
`newid`	NEWID (Boswell, 1990)

Tabela 4.2: Os identificadores especiais para diretórios utilizados atualmente pelo ambiente SNIFFER.

Ao ambiente SNIFFER deve ser fornecido um *ponto de entrada*, ou seja, um diretório que seja o raiz para todo o experimento. A partir desse diretório, o ambiente vasculha a árvore de diretório contida no ponto de entrada a procura de diretórios com identificadores especiais. Os identificadores especiais identificam um sistema de aprendizado específico. Atualmente, os identificadores especiais que o ambiente reconhece estão listados na Tabela 4.2. Entretanto, o ambiente SNIFFER está preparado para acomodar outros sistemas de aprendizado como descrito na Seção 4.4.3 na página 84.

Quando um diretório com identificador especial é encontrado, o ambiente SNIFFER

realiza as seguintes tarefas:

1. Caso for solicitado pelo usuário, o ambiente executa um método de reamostragem dividindo os dados em diversos conjuntos de treinamento e teste. Nesse caso, arquivos com extensão `.data` e `.names` precisam estar presentes no diretório com o identificador especial. Como resultado, são criados diversos diretórios com os identificadores `it0`, `it1`, ... `it(k-1)`, um para cada iteração do método de reamostragem, sendo k o número total de iterações do método. Dentro de cada um desses diretórios o ambiente encontra três arquivos com as extensões `.data`, `.test` e `.names`, os quais contém, respectivamente, o conjunto de treinamento e teste dessa iteração, e o arquivo de declaração de atributos relacionado aos conjuntos de treinamento e teste;

2. Por outro lado, o usuário pode não desejar que o ambiente aplique um método de reamostragem e, em vez disso, o usuário pode criar os diretórios com os conjuntos de treinamento e teste na forma `it0`, `it1`, ... `it(k-1)`. Isso é útil em algumas situações como, por exemplo, quando o usuário deseja avaliar dois ou mais sistemas de aprendizado nos mesmos conjuntos de treinamento e teste. Nesse caso, o usuário solicita que o ambiente não execute um método de reamostragem, e o ambiente utiliza os arquivos de dados com as extensões `.data`, `.test` e `.names`, que foram armazenados pelo usuário, dentro de cada diretório com o identificador `it`.

Uma vez que o ambiente se certificou de que os dados estão divididos em conjuntos de treinamento e teste, o ambiente passa a executar sobre esses dados o sistema de aprendizado identificado pelo nome do diretório atual — Tabela 4.2 na página anterior. Para isso, o ambiente acessa cada diretório `it` e realiza as seguintes tarefas:

1. Caso for solicitado pelo usuário, o ambiente SNIFFER converte a sintaxe dos arquivos de dados para a sintaxe do sistema de aprendizado a ser executado;

2. O ambiente executa o sistema de aprendizado selecionado sobre o conjunto de treinamento, e mede o erro de classificação sobre o conjunto de teste;

3. Por fim, o ambiente extrai dos arquivos de saída do sistema de aprendizado a matriz de confusão com as classificações incorretas no conjunto de teste. As matrizes de confusão são armazenadas pelo ambiente para a realização de cálculos de desempenho.

O ambiente SNIFFER deixa gravado em cada diretório `it` alguns arquivos com o resultado da execução de um sistema de aprendizado em uma única iteração de um método de reamostragem. Esses arquivos possuem como nome o mesmo identificador do arquivo de dados sobre o qual o sistema de aprendizado foi executado, cada arquivo é diferenciado apenas pela extensão. As extensões utilizadas são:

`.out`
: Toda saída direcionada para o dispositivo padrão de saída pelo sistema de aprendizado é redirecionada para esse arquivo. A utilidade desse arquivo pode variar de sistema de aprendizado para sistema de aprendizado. Para alguns sistemas de aprendizado, como o C4.5, essa saída é a forma mais simples de verificar as regras e a precisão do classificador induzido. Outros sistemas de aprendizado, como o CN2, gravam as regras em arquivo texto separado. De qualquer forma, esse arquivo é muito útil para verificar como foi o processo de indução como, por exemplo, a geração do classificador e a poda. Esse arquivo pode também ajudar a identificar os erros quando a execução de um sistema de aprendizado falha;

`.rules`
: Alguns sistemas de aprendizado, como o CN2 e o NEWID, permitem que suas regras sejam gravadas em arquivos texto. Nesses casos, o ambiente SNIFFER grava essas regras em um arquivo com essa extensão;

`.stdrules`
: Cada sistema de aprendizado apresenta as suas regras ou árvores de decisão com uma sintaxe própria. Diferenças nas representações utilizadas por cada sistema fazem com que seja muito complicado integrar o conhecimento induzido por cada sistema de aprendizado. Para facilitar esse processo foi desenvolvida uma sintaxe padrão para regras chamada $\mathcal{PBM}$ (Prati, Baranauskas & Monard, 2001b,a). Uma vez que o conhecimento está em uma sintaxe comum, é possível integrá-lo de diversas formas como, por exemplo, utilizando *ensembles* (Wolpert, 1992; Breiman, 1996; Quinlan, 1996), ou pela composição de classificadores simbólicos em um classificador final também simbólico, como no sistema XRULER proposto por Baranauskas (2001) e no sistema MCE proposto por Bernardini (2002).

Todos os resultados de desempenho obtidos com a aplicação de um método de reamostragem são armazenados e identificados com uma *chave*. Uma chave é o caminho composto pelos identificadores dos diretórios desde o ponto de entrada até o identificador do sistema de aprendizado a ser executado. Por exemplo, `./Experimento/Breast/c4.5`

é a chave de identificação dos resultados obtidos pela execução do sistema de aprendizado C4.5 no conjunto de dados **Breast**, o qual é o caminho mais à esquerda na Figura 4.8 na página 78.

O ambiente SNIFFER percorre toda a árvore de diretórios contida no ponto de entrada. Ao final da busca, podem ser realizadas comparações entre os resultados. Atualmente o ambiente utiliza o teste-t pareado para *k-fold cross-validation*[12] (Dietterich, 1997a; Salzberg, 1997). Para restringir as comparações, o usuário pode especificar sub-árvores de diretórios nas quais os resultados devem ser comparados. Por exemplo, como não faz sentido comparar resultados obtidos em conjuntos de dados diferentes, o usuário pode especificar `./Experimento/Breast` para que somente os resultados que estão localizados dentro desse diretório sejam comparados, ou seja, para que somente os resultados obtidos pelos sistemas de aprendizado C4.5, CN2 e C4.5 RULES sejam comparados entre si para o conjunto de dados **Breast**. Para que o ambiente compare os resultados de cada conjunto de dados em separado, o usuário deve especificar uma lista com os diretórios `[./Experimento/Breast, ./Experimento/Bupa, ./Experimento/CMC, ./Experimento/Pima]`.

Ao final do cálculo dos testes de comparação, o ambiente pode tanto gerar relatórios descrevendo os resultados obtidos, quanto ser acessado por meio de sua API para que o usuário recupere determinados dados ou estatísticas de desempenho. Em ambos os casos, as chaves descritas anteriormente são utilizadas para identificar os resultados.

O ambiente SNIFFER gera quatro relatórios de resultados, são eles:

`Summary`
: Esse relatório apresenta um resumo das estatísticas calculadas pelo ambiente. São apresentados os erros médio e os desvios padrão das k iterações do método de reamostragem. Os erros médio e os desvios padrão são calculados tanto para cada valor do atributo classe individualmente quanto para todos os valores possíveis do atributo classe. Um exemplo de relatório gerado para o experimento ilustrado na Figura 4.8 na página 78 é apresentado no Apêndice B, Seção B.1 na página 181;

`Detailed`
: Esse relatório é gerado para cada execução de um método de reamostragem. Ele apresenta, para cada iteração do método de reamostragem, a matriz de confusão e as taxas de erro para cada valor do atributo classe em separado, e para todos os valores em conjunto. Ao final, esse relatório apresenta um resumo de todas as iterações, além da taxa de erro média e do desvio padrão para todas as interações.

[12] *k-fold cross-validation paired t-test.*

No Apêndice B, Seção B.2 na página 184 é apresentado um exemplo desse relatório para o experimento descrito nesta seção;

`HypothesysTest`

Os resultados dos testes de hipótese comparando dois ou mais resultados obtidos com a aplicação de um método de reamostragem são apresentados nesse relatório. Os resultados apresentados respeitam as restrições de comparação impostas pelo usuário. Os testes de hipótese são realizados para cada valor do atributo classe individualmente e para todos os valores em conjunto. Ainda, o relatório indica quando um resultado é estatisticamente significativo (com 95% de confiança) ou altamente significativo (com 99% de confiança). No Apêndice B, Seção B.3 na página 188 é apresentado um exemplo desse relatório para o experimento utilizado nesta seção;

`GnuPlot`

O último relatório gerado apresenta os mesmos resultados do relatório `Summary`, mas em um formato mais simples. Esse relatório está no mesmo formato de dados utilizado pelo utilitário de geração de gráficos GnuPlot™ (Crawford, 1998), mas também pode ser utilizado por outros utilitários, bem como para a geração de tabelas no processador de textos LaTeX (Kopla & Daly, 1999).

4.4.2 A Arquitetura do Ambiente Computacional SNIFFER

Como descrito anteriormente, o ambiente SNIFFER foi desenvolvido utilizando a biblioteca DOL, dessa forma o ambiente SNIFFER serve também como *prova de conceito*. O ambiente provê uma API que permite estender as funcionalidades do ambiente. Essa API pode ser utilizada, por exemplo, para avaliar os sistemas de aprendizado sob novas medidas de desempenho (Provost, Fawcett & Kohavi, 1998; Provost & Fawcett, 2001) ou implementar novos testes estatísticos de hipótese (Dietterich, 1997b; Salzberg, 1997). Dessa forma, o ambiente SNIFFER complementa as funcionalidades da biblioteca DOL, fazendo com que o usuário tenha total controle, desde a implementação até a avaliação de técnicas de pré-processamento de dados.

Na Figura 4.9 na próxima página é apresentada uma representação gráfica da arquitetura do ambiente SNIFFER. Esse ambiente conta com quatro módulos principais, são eles:

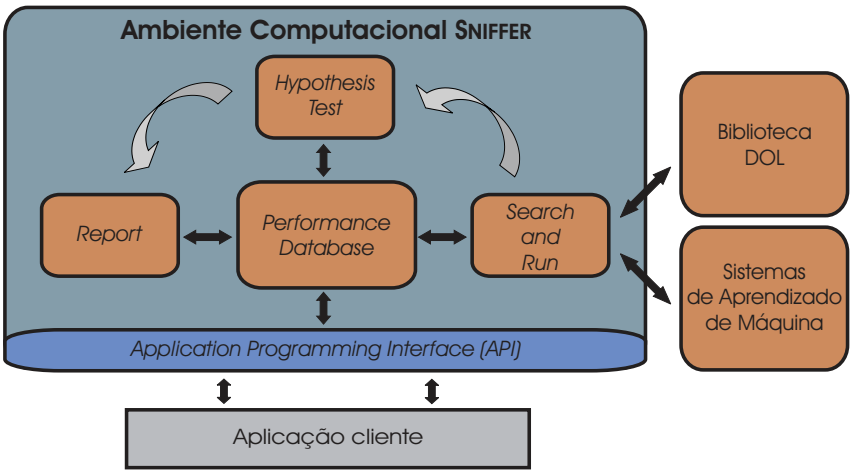

Figura 4.9: Arquitetura do ambiente computacional SNIFFER.

SearchandRun

Esse módulo vasculha a sub-árvore de diretórios contida no ponto de entrada a procura de diretórios com identificadores especiais. Quando um diretório com identificador especial é encontrado, esse módulo realiza todas as tarefas necessárias para executar o sistema de aprendizado, executa o sistema de aprendizado e extrai a matriz de confusão no conjunto de teste. As matrizes de confusão extraídas são armazenadas e gerenciadas por outro módulo chamado PerformanceDatabase. Esse módulo é responsável por armazenar e permitir consultas a todos os dados e estatísticas obtidas pelo ambiente SNIFFER. O módulo SearchandRun faz a interface com os sistemas de aprendizado. Além disso, esse módulo utiliza a biblioteca DOL para dividir o conjunto de dados em conjuntos de treinamento e teste, bem como realizar conversões entre sintaxes. Ao final da busca, o módulo SearchandRun ativa o módulo HypothesisTest, o qual realiza os testes de hipótese com o objetivo de identificar diferenças significativas entre os resultados;

HypothesisTest

Esse módulo aplica os testes de hipótese sobre os resultados obtidos com a finalidade de identificar diferenças significativas. Como descrito anteriormente, esse módulo utiliza atualmente o teste-t pareado para *k-fold cross-validation*. Outros testes podem ser implementados utilizando a API do ambiente SNIFFER, ou por meio da criação de uma classe específica para o novo teste de hipótese, como é discutido na Seção 4.4.3 na página seguinte. Os resultados dos testes de hipótese são armazenados e gerenciados pelo módulo PerformanceDatabase;

PerformanceDatabase

O módulo PerformanceDatabase implementa um banco de dados com dados e estatísticas sobre o desempenho dos sistemas de aprendizado executados. Os dados que esse módulo armazena vêm dos módulos SearchandRun e HypothesisTest. O módulo Report acessa os dados armazenados nesse banco de dados para organizar os relatórios fornecidos ao usuário. Ainda, a API do ambiente SNIFFER fornece ao usuário acesso aos dados armazenados no módulo PerformanceDatabase.

Report

O módulo Report organiza as informações presentes no módulo PerformanceDatabase em relatórios gravados em arquivos texto. Esses relatórios fornecem ao usuário visões gerais e detalhadas do desempenho dos sistemas de aprendizado, e preparam os dados para serem utilizados por outros aplicativos.

4.4.3 O Projeto do Ambiente Computacional SNIFFER

Esta seção apresenta o projeto do ambiente computacional SNIFFER. O objetivo desta seção é fornecer uma visão geral de como o ambiente SNIFFER foi projetado e implementado. De uma forma geral, o ambiente foi projetado tendo em vista possíveis extensões que podem ser adicionadas a ele. Nesse caso, alguns padrões de projeto ajudam a melhorar o projeto do ambiente, tornando o projeto flexível o bastante para que novas funcionalidades sejam adicionadas ao ambiente, sem que exista a necessidade de realizar grandes modificações no projeto original.

Algumas modificações com as quais o ambiente SNIFFER terá que lidar no futuro são:

Novos sistemas de aprendizado

Pode-se desejar adicionar um sistema de aprendizado que ainda não foi integrado ao ambiente SNIFFER. Nesse caso, o ambiente SNIFFER utiliza classes que "envolvem" os sistemas de aprendizado fazendo com que esses sistemas tenham o mesmo funcionamento do ponto de vista externo. Adicionar um novo sistema de aprendizado significa criar uma dessas classes específica para o novo sistema de aprendizado;

Novos métodos de reamostragem

Diferentes métodos de reamostragem podem ser empregados em conjuntos de dados com características distintas. Por exemplo, o método *k-fold cross-validation* pode ser utilizado em conjuntos de dados de tamanho médio com excelente precisão. Para

conjuntos de dados menores (abaixo de 200 exemplos), o método *bootstraping* (Batista, 1997; Weiss & Kulikowski, 1991) é mais recomendado. O ambiente SNIFFER utiliza a biblioteca DOL para aplicar métodos de reamostragem nos conjuntos de dados. Para adicionar novos métodos de reamostragem ao ambiente SNIFFER basta que o método de reamostragem seja adicionado a biblioteca DOL;

Novos testes de hipótese

A comunidade de AM ainda não chegou a um consenso sobre quais testes de hipóteses devem ser utilizados para comparar o desempenho de dois sistemas de aprendizado (Dietterich, 1997b; Salzberg, 1997). Novos testes de hipótese podem ser integrados ao ambiente SNIFFER de duas formas diferentes: ou por meio da API que o ambiente proporciona; ou por meio da criação de uma classe específica para esse fim.

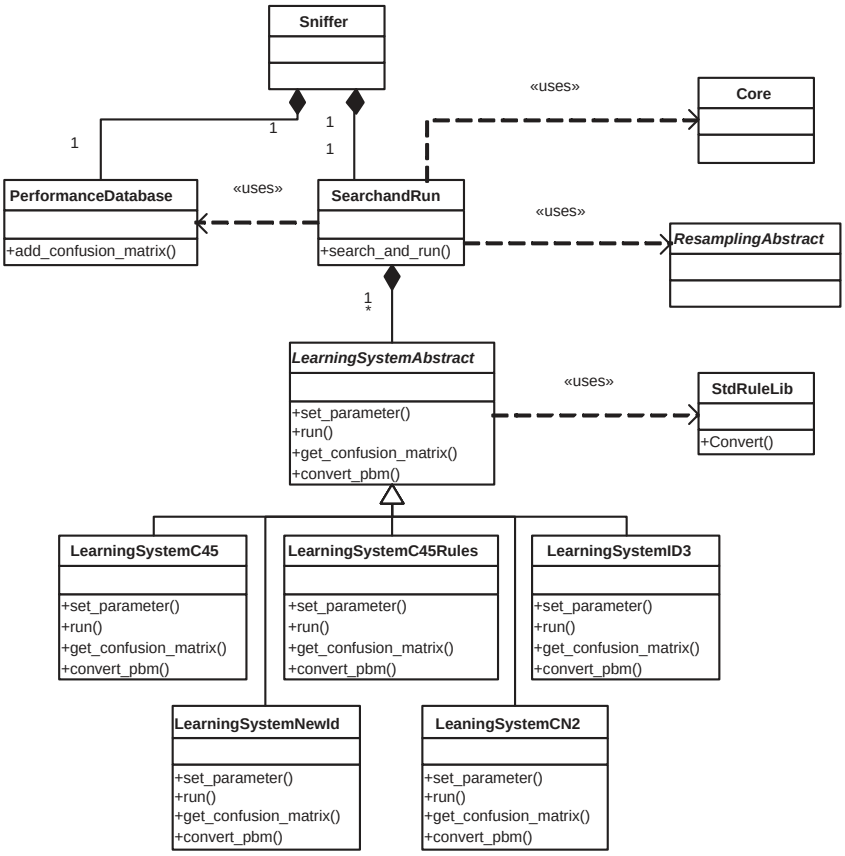

Figura 4.10: Projeto do módulo SearchandRun do ambiente computacional SNIFFER.

Na Figura 4.10 é apresentado o projeto do módulo SearchandRun. Esse módulo possui uma classe principal também chamada `SearchandRun`. Essa classe percorre a árvore de diretórios em busca de diretórios com identificadores especiais, como explicado na

Seção 4.4.1 na página 77. A classe `SearchandRun` utiliza as classes `Core` e `ResamplingAbstract` para converter entre as sintaxes dos sistemas de aprendizado e dividir os dados em conjuntos de treinamento e teste, respectivamente.

A classe `SearchandRun` utiliza a classe `LearningSystemAbstract` para realizar a interface entre o módulo e os diversos sistemas de aprendizado. A classe `LearningSystemAbstract` define métodos para ajustar os parâmetros do sistema de aprendizado, executar esse sistema, obter a matriz de confusão no conjunto de teste e converter as regras para o formato padrão de regras, isto é, o formato $\mathcal{PBM}$ (Prati, Baranauskas & Monard, 2001b). Para realizar a conversão para o formato $\mathcal{PBM}$, a classe `LearningSystemAbstract` utiliza a classe `StdRuleLib`. As sub-classes da classe `LearningSystemAbstract` implementam os métodos descritos anteriormente para cada sistema de aprendizado suportado pelo ambiente SNIFFER.

A classe `SearchandRun` utiliza a classe `PerformanceDatabase` para armazenar as matrizes de confusão obtidas nas execuções dos sistemas de aprendizado. A classe `Sniffer` faz a interface entre os módulos internos do ambiente SNIFFER e a aplicação externa. A classe `Sniffer` também provê a API para que classes externas ao ambiente possam estender as suas funcionalidades.

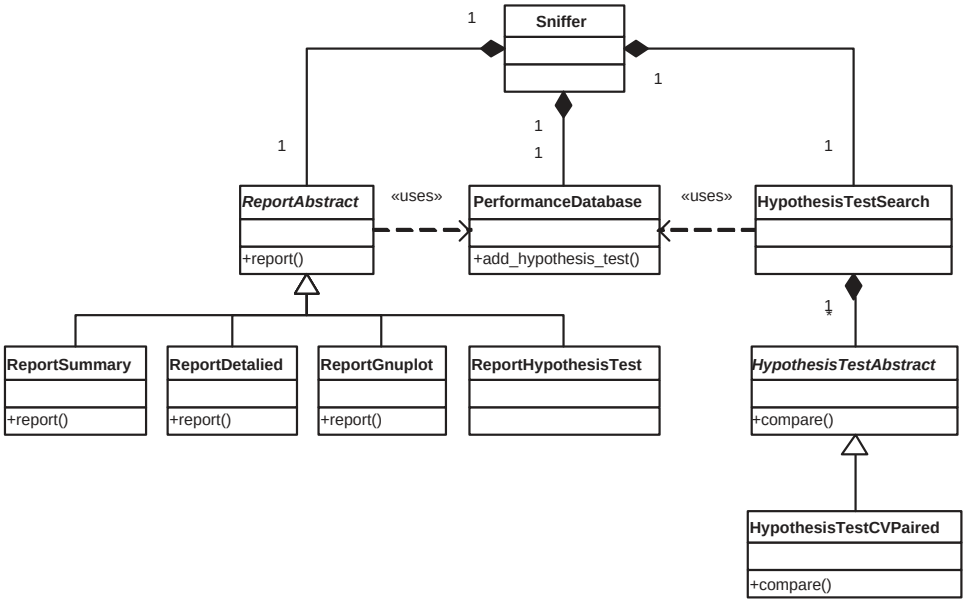

Figura 4.11: Projeto dos módulos Report e HypothesisTest do ambiente computacional SNIFFER.

Na Figura 4.11 é apresentado o projeto dos módulos Report e HypothesisTest. A classe `HypothesisTestSearch` procura por resultados a serem comparados respeitando

as limitações impostas pelo usuário. Quando dois resultados a serem comparados são encontrados, essa classe utiliza a classe `HypothesisTestAbstract` para compará-los. A classe `HypothesisTestAbstract` implementa a estrutura comum entre diversos testes de hipótese, e as sub-classes de `HypothesisTestAbstract` implementam testes de hipótese específicos. Atualmente, o teste-t pareado para *k-fold cross-validation* está implementado pela classe `HypothesisTestCVPaired`.

A classe `ReportAbstract` cria a estrutura para a implementação de relatórios. Atualmente, quatro classes derivadas da classe `ReportAbstract` implementam os relatórios disponibilizados pelo ambiente: `Summay`, `Detailed`, `GnuPlot` e `HypothesisTest`.

4.5 Considerações Finais

O objetivo deste capítulo é oferecer uma visão geral sobre o ambiente computacional de Aprendizado de Maquina DLE. Esse ambiente é constituído de dois módulos principais: a biblioteca para pré-processamento de dados DOL e o ambiente computacional para avaliação e comparação de sistemas de aprendizado SNIFFER.

A área de AM é uma área de pesquisa dinâmica, na qual novos métodos são propostos com freqüência. Dessa forma, é necessário que os projetos, tanto da biblioteca DOL, quanto do ambiente SNIFFER, sejam feitos tendo em vista futuras extensões, como por exemplo, a adição de novos sistemas de aprendizado. Nesse sentido, o uso de padrões de projeto ajudaram na criação de projetos mais flexíveis.

Como base para a criação do ambiente computacional DLE está a sintaxe DSX, descrita no Apêndice A na página 169. Essa sintaxe possui diversos recursos, como o suporte à indução construtiva apoiada pelo usuário e o gerenciamento de diversos tipos de dado.

Por fim, é importante notar que o ambiente DLE foi utilizado com sucesso, não somente neste trabalho, mas também por vários pesquisadores do nosso grupo (Lorena et al., 2002a,b; Milaré et al., 2002; Sanches, 2002; Pugliesi, 2001; Dosualdo, 2002), para realizar diversos experimentos com algoritmos de aprendizado.

Neste trabalho, o ambiente DLE foi utilizado para implementar métodos de tratamento de valores desconhecidos e conjuntos de dados com classes desbalanceadas. A pesquisa relacionada com esses dois problemas de pré-processamento de dados é apresentada nos próximos capítulos.

Capítulo 5

Tratamento de Valores Desconhecidos

5.1 Considerações Iniciais

Qualidade de dados é uma preocupação central em Aprendizado de Máquina e outras áreas de pesquisa relacionadas à Descoberta de Conhecimento de Bancos de Dados. Uma vez que a maioria dos algoritmos de aprendizado induzem conhecimento estritamente a partir de dados, a qualidade do conhecimento extraído é amplamente determinada pela qualidade dos dados de entrada.

Um problema relevante em qualidade de dados é a presença de *valores desconhecidos*, também chamados de *valores ausentes*. Valores desconhecidos ou ausentes consistem na não medição dos valores de um atributo para alguns casos. Os valores desconhecidos podem ter diversas fontes como a morte de pacientes, defeitos em equipamentos, recusa por parte de entrevistados em responder determinadas perguntas, entre outras. Apesar da freqüente ocorrência de valores desconhecidos em conjuntos de dados, muitos analistas de dados tratam os valores desconhecidos de forma bastante simplista. Entretanto, o tratamento de valores desconhecidos deve ser cuidadosamente pensado, caso contrário, distorções podem ser introduzidas no conhecimento induzido.

Na maioria dos casos, os atributos de um conjunto de dados não são independentes entre si. Dessa forma, aproximações para os valores desconhecidos podem ser determinadas por meio da identificação de relações entre os atributos. *Imputação*[1] é um termo

[1] *Imputation*.

utilizado para denotar um procedimento que substitui os valores desconhecidos de um conjunto de dados por valores estimados. Essa abordagem permite que o tratamento de valores desconhecidos seja independente do algoritmo de aprendizado utilizado, o que permite ao analista de dados selecionar o método de tratamento de valores desconhecidos mais apropriado para cada conjunto de dados.

O objetivo deste capítulo é analisar o desempenho do algoritmo K-VIZINHOS MAIS PRÓXIMOS como um método de imputação, e comparar o desempenho desse algoritmo com o desempenho obtido por outros métodos de tratamento de valores desconhecidos. Um dos métodos que será utilizado nas comparações de desempenho é a IMPUTAÇÃO PELA MÉDIA OU MODA. Esse método é bastante simples e amplamente utilizado. Ele consiste em substituir os valores desconhecidos de um atributo pela média dos valores conhecidos do atributo, se o atributo for quantitativo; ou moda dos valores conhecidos do atributo, se o atributo for qualitativo. Outros dois métodos utilizados nas comparações de desempenho são as estratégias internas utilizadas pelos algoritmos de aprendizado CN2 (Clark & Niblett, 1989) e C4.5 (Quinlan, 1988).

Este capítulo está organizado da seguinte forma: na Seção 5.2 é descrito o problema da distribuição dos valores desconhecidos e a taxonomia proposta por Little & Rubin (2002) para classificar o grau de aleatoriedade dos valores desconhecidos em um conjunto de dados; na Seção 5.3 são descritos alguns dos métodos mais utilizados para tratamento de valores desconhecidos; a Seção 5.4 é dedicada a uma classe específica de métodos de tratamento de valores desconhecidos: imputação; na Seção 5.5 é apresentada a variação do algoritmo K-VIZINHOS MAIS PRÓXIMOS proposta neste trabalho e a utilização desse algoritmo como método de imputação; na Seção 5.6 é descrito como os algoritmos de aprendizado C4.5 e CN2 tratam os valores desconhecidos internamente; na Seção 5.7 é realizado um estudo comparativo envolvendo o algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação, a IMPUTAÇÃO PELA MÉDIA OU MODA, e os métodos internos utilizados pelos algoritmos C4.5 e CN2 para tratar valores desconhecidos; por fim, na Seção 5.8 são apresentadas as considerações finais deste capítulo.

5.2 A Aleatoriedade dos Valores Desconhecidos

A aleatoriedade dos valores desconhecidos é um fator importante a ser analisado para a escolha do método de tratamento de valores desconhecidos. Na sua forma mais simples, os valores desconhecidos podem estar distribuídos aleatoriamente nos dados. Isso significa

que a probabilidade de encontrar um valor desconhecido é a mesma para qualquer valor do atributo.

Por outro lado, os valores desconhecidos podem estar não aleatoriamente distribuídos. Isso significa que a probabilidade de encontrar um valor desconhecido pode depender, por exemplo, do valor verdadeiro (não conhecido) do valor desconhecido.

Por exemplo, no gráfico da Figura 5.1-*a* é mostrada a renda média das esposas, dadas as rendas médias dos maridos[2]. A correlação entre esses dois atributos é bastante comum e pode ser encontrada com freqüência em diversos atributos de um conjunto de dados. Esse gráfico também mostra a reta de regressão linear.

Em algumas situações, os valores desconhecidos podem não estar aleatoriamente dispersos sobre os dados. Por exemplo, é comum que pessoas com renda mais alta, e provavelmente, com grau mais alto de instrução, se recusem a responder perguntas sobre a sua renda. Nesse caso, vamos supor que os pontos marcados com "□" representam maridos que se recusaram a responder a pergunta sobre a suas rendas. Sendo assim, a probabilidade de um valor ser desconhecido é proporcional à renda do marido.

Existem diversos métodos que podem ser utilizados para tratar esses valores desconhecidos. Um método amplamente utilizado, mas altamente perigoso, é a imputação dos valores desconhecidos pela média do atributo, ou seja, pela média das rendas dos maridos. Nesse caso, como os valores desconhecidos são mais prováveis em maridos com altas rendas, essa substituição distorce os dados, introduzindo falsos padrões nos dados e alterando as relações entre os atributos, como é mostrado na Figura 5.1-*b*.

Uma outra possibilidade, a qual é proposta neste trabalho, é a utilização do algoritmo K-VIZINHOS MAIS PRÓXIMOS. Nesse caso, as relações entre os atributos podem ser utilizadas para predizer os valores desconhecidos das rendas dos maridos. Como resultado, obtém-se um resultado mais próximo do conjunto de dados original. Esse resultado também preserva em grande parte as relações entre os atributos, como é mostrado na Figura 5.1-*c*.

O grau de aleatoriedade dos valores desconhecidos pode ser dividido em três classes Little & Rubin (2002):

Ausentes de forma completamente aleatória

Valores desconhecidos dispostos de forma completamente aleatória (MCAR[3]) são os

[2]Esse exemplo utiliza dados obtidos pelo *U.S. Bureau of the Census* na cidade de Nova Iorque no ano de 1993 (Freedman, Pisani & Purves, 1998).

[3]*Missing completely at random.*

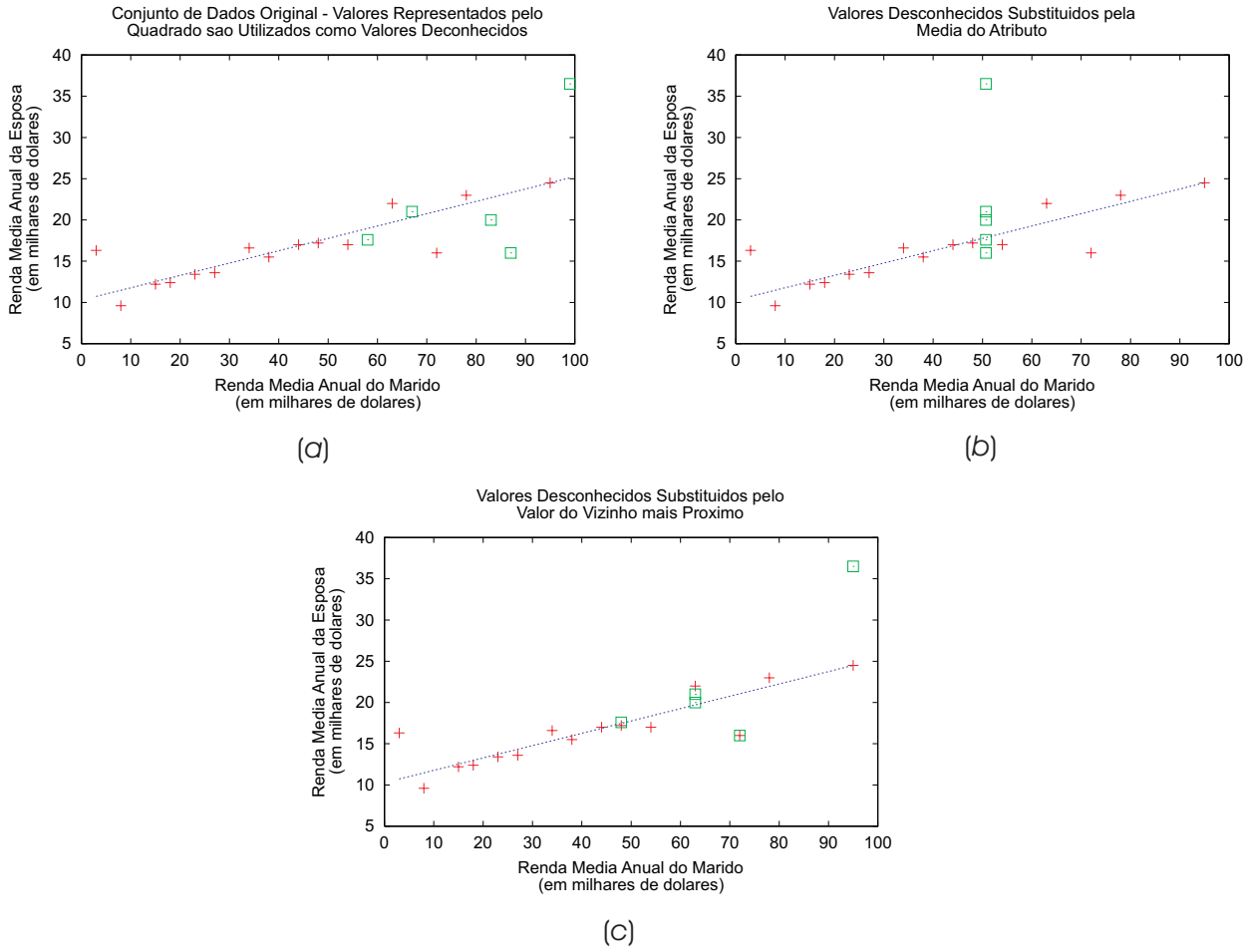

Figura 5.1: Exemplo de valores desconhecidos não aleatoriamente distribuídos.

que possuem o mais alto grau de aleatoriedade. Eles ocorrem quando os valores desconhecidos de um atributo, por exemplo, a renda do marido, não estão relacionados com os valores desse atributo nem com os valores de qualquer outro atributo do conjunto de dados. Um exemplo de processo que poderia gerar valores desconhecidos MCAR seria a perda aleatória de parte dos dados, por exemplo, por meio da perda de formulários com as respostas dos entrevistados. MCAR é uma suposição muito restrita, a qual dificilmente pode ser encontrada em problemas práticos;

Ausentes de forma aleatória

Valores desconhecidos distribuídos de forma aleatória (MAR[4]) ocorrem quando os valores desconhecidos de um atributo não estão relacionados com os valores desse atributo, mas dependem dos valores de um ou mais atributos do conjunto de dados. Por exemplo, se a probabilidade de um marido se recusar a responder sobre a sua

[4] *Missing at random.*

renda não depende da sua própria renda, mas depende da renda da esposa, então os valores desconhecidos são MAR.

Ausentes de forma não aleatória

Valores desconhecidos disposto de forma não aleatória (NMAR[5]) ocorrem quando a probabilidade de um exemplo possuir um valor desconhecido para um atributo depende dos valores do atributo com valores desconhecidos, e possivelmente dos valores dos demais atributos do conjunto de dados. Nesse caso, se a probabilidade de um marido se recusar a responder sobre a sua renda depende da sua própria renda, então esses valores desconhecidos são NMAR.

5.3 Métodos para Tratamento de Valores Desconhecidos

Existem diversos métodos para tratamento de valores desconhecidos disponíveis na literatura. Muitos desses métodos, tal como o método de *substituição de casos*, foram desenvolvidos para pesquisas de opinião, e possuem algumas limitações se analisados sob a óptica da análise de dados utilizada em KDD. Outros métodos, tal como a substituição de valores desconhecidos pela média ou moda do atributo, são muito simplistas e devem ser cuidadosamente aplicados para evitar a inserção de sérias distorções nos dados.

De uma forma geral, os métodos de tratamento de valores desconhecidos podem ser divididos em três categorias, como proposto em (Little & Rubin, 2002):

Ignorar ou descartar dados

Existem duas abordagens mais utilizadas para descartar dados com valores desconhecidos. A primeira é conhecida como *análise de casos completos*. Essa abordagem está disponível na maioria dos programas estatísticos, e é o método *default* em diversos deles. Essa abordagem consiste em descartar qualquer caso que possua um ou mais valores desconhecidos. A segunda abordagem é conhecida como *descarte de casos e/ou atributos*. Essa abordagem consiste em determinar a extensão dos valores desconhecidos em cada exemplo e em cada atributo, e remover os exemplos e/ou os atributos com grandes quantidades de valores desconhecidos. Ambas abordagens, análise de casos completos e descarte de casos e/ou atributos, devem ser aplicadas somente quando os valores desconhecidos estão aleatoriamente distribuídos, uma vez

[5] *Not missing at random.*

que valores desconhecidos não aleatoriamente distribuídos possuem elementos não aleatórios que podem introduzir distorções nos dados;

Estimativa de parâmetros

Procedimentos ML[6] são utilizados para estimar os parâmetros de um modelo definido para os valores observados dos dados. Procedimentos ML que utilizam variações do algoritmo EM[7] (Dempster, Laird & Rubin, 1977) podem estimar parâmetros de um modelo na presença de valores desconhecidos;

Imputação

Imputação é uma classe de procedimentos que visa substituir os valores desconhecidos por valores estimados. Existem diversas formas de estimar um valor desconhecido. As abordagens mais simples utilizam estatísticas obtidas dos dados como a média ou a moda dos valores conhecidos do atributo. Entretanto, métodos mais sofisticados podem fazer uso de relações entre os atributos que podem ser identificadas nas dados. Os métodos de imputação são descritos mais detalhadamente na próxima seção.

5.4 Métodos de Imputação

Como descrito anteriormente, os métodos de imputação substituem valores desconhecidos por valores estimados. Os valores são estimados por meio de alguma informação extraída do conjunto de dados. A seguir são descritos alguns métodos de imputação amplamente difundidos na literatura.

Substituição de casos

Esse método é tipicamente utilizado em pesquisas de opinião. Um caso com valores desconhecidos, por exemplo, uma pessoa que não pode ser contactada, é substituído por um outro caso, isto é, outra pessoa, não considerada amostra original da pesquisa de opinião;

Imputação pela média ou moda

Esse método é um dos métodos mais freqüentemente utilizados. Ele consiste em substituir os valores desconhecidos de um dado atributo pela média, para atributos quantitativos, ou pela moda, para atributos qualitativos, ambas calculadas por meio dos valores observados do atributo;

[6] *Maximum likelihood.*
[7] *Expectation-Maximization.*

A média é a melhor estimativa para o valor de um atributo desconhecido, na ausência de outras informações a respeito dos dados. Esse procedimento possui a vantagem de ser conservador, uma vez que essa substituição não altera a média geral do atributo. Por outro lado, a variância (dispersão) da variável é reduzida porque a média é provavelmente mais próxima dela mesma que o valor real do atributo desconhecido. Além disso, as relações entre os atributos também podem ser alteradas.

Conhecimento de domínio

Conhecimento de domínio pode utilizado pelo especialista de domínio para substituir os valores desconhecidos por valores estimados por meio da experiência do especialista. De uma forma geral, esse procedimento é seguro quando o especialista está familiarizado com a aplicação, o conjunto de dados é grande e o número de valores desconhecidos é pequeno. Alternativamente, o especialista no domínio pode discretizar um atributo quantitativo (por exemplo: o atributo `renda` pode ser discretizado em `Classe A`, `Classe B`, `Classe C` e `Classe D`) de forma que se possa predizer com mais confiança em qual categoria está o caso com valor desconhecido. Sendo assim, a variável discreta pode substituir a variável quantitativa na análise, levando-se em consideração que existe uma perda de informação nessa transformação.

Esse método possui o mérito de ser menos conservador que a simples remoção de casos ou atributos. Estando familiarizado com a aplicação, o especialista no domínio será capaz de estimar os valores com uma precisão maior que uma substituição pela média. Entretanto, essa substituição é manual e fica restrita a pequenas quantidades de valores desconhecidos. Ainda, as estimativas do especialista são limitadas ao conhecimento existente sobre os dados, o que de certa forma pode direcionar o conhecimento a ser aprendido.

Hot deck e cold deck

No método *hot deck*, um valor desconhecido é substituído por um valor obtido por meio de uma distribuição estimada a partir dos dados disponíveis. O método *hot deck* é tipicamente implementado em dois estágios. No primeiro estágio, o conjunto de dados é particionado em *clusters* utilizando um método de aprendizado não supervisionado. No segundo estágio, cada exemplo com valores desconhecidos é associado a um dos *clusters*. Os exemplos completos no *cluster* são utilizados para estimar os valores desconhecidos. Uma forma de estimar os valores desconhecidos é por meio do cálculo da média ou da moda do atributo, utilizando somente os exemplos membros do *cluster*. O método *cold deck* é similar ao *hot deck*, entretanto, a amostra de dados utilizada para gerar os *clusters* deve ser diferente da amostra com

valores desconhecidos;

Modelo de predição

Modelos de predição são procedimentos sofisticados para tratar valores desconhecidos. Esses métodos consistem na criação de um modelo preditivo para estimar valores que irão substituir os valores desconhecidos. O atributo com valores desconhecidos é utilizado como atributo classe, e os demais atributos são utilizados como entrada para o modelo de predição. Um importante argumento a favor dessa abordagem é que, freqüentemente, os atributos possuem correlações entre si. Dessa forma, essas correlações podem ser utilizadas para criar um modelo preditivo de classificação ou regressão dependendo do tipo do atributo com valores desconhecidos ser, respectivamente, qualitativo ou quantitativo. Algumas correlações entre os atributos podem ser mantidas se elas foram capturadas pelo modelo preditivo. Uma limitação importante dessa abordagem é que os valores estimados são geralmente mais bem comportados do que os valores reais (não conhecidos) seriam, ou seja, os valores preditos são mais consistentes com o conjunto de atributos do que os valores reais seriam. Uma segunda limitação é a necessidade por correlações entre os atributos. Se não existem correlações entre um ou mais atributos em um conjunto de dados, então o modelo preditivo não será preciso em estimar os valores desconhecidos.

5.5 Imputação com o Algoritmo K-Vizinhos mais Próximos

Como mencionado anteriormente, neste trabalho é proposta a utilização do algoritmo k-vizinhos mais próximos para estimar e substituir valores desconhecidos. Os principais benefícios dessa abordagem são:

- O algoritmo k-vizinhos mais próximos pode ser utilizado para predizer tanto atributos qualitativos, por meio do cálculo da moda entre os k vizinhos mais próximos, quanto atributos quantitativos, por meio da média dos k vizinhos mais próximos;
- Não é necessário criar um modelo preditivo para cada atributo com valores desconhecidos. Na realidade, o algoritmo k-vizinhos mais próximos não cria modelos explícitos como, por exemplo, uma árvore de decisão ou um conjunto de regras. O próprio conjunto de dados é utilizado como um modelo "*lazy*". Dessa forma, o

algoritmo K-VIZINHOS MAIS PRÓXIMOS pode ser facilmente adaptado para utilizar qualquer atributo como atributo classe. Essa adaptação é feita por meio da modificação de quais atributos devem ser considerados na função de distância;

- O algoritmo K-VIZINHOS MAIS PRÓXIMOS pode ser utilizado mesmo quando existem valores desconhecidos em diversos atributos. Nessa situação o algoritmo utiliza toda a informação disponível, isto é, todos os valores conhecidos, para predizer os valores desconhecidos.

A principal limitação do uso do algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação, é que a busca realizada por esse método para encontrar os exemplos mais similares necessita de uma passagem completa pelos dados. Essa limitação pode ser bastante crítica para aplicações de KDD, uma vez que essa área de pesquisa possui, como um de seus principais objetivos, a análise de grandes volumes de dados. Diversos trabalhos com o objetivo de solucionar essa limitação podem ser encontrados na literatura, tais como a criação de um conjunto reduzido de dados composto apenas por exemplos prototípicos (Wilson & Martinez, 2000). Neste trabalho é utilizado um método de acesso chamado M-tree (Ciaccia, Patella & Zezula, 1997), o qual foi implementado para a variante do algoritmo K-VIZINHOS MAIS PRÓXIMOS utilizada neste trabalho. As estruturas M-tree organizam um conjunto de dados por meio de espaços métricos genéricos. A utilização de estruturas M-tree reduz consideravelmente o número de cálculos de distância em consultas de similaridade.

Na próxima seção é apresentada uma breve descrição do algoritmo K-VIZINHOS MAIS PRÓXIMOS e da variante desse algoritmo implementada neste trabalho.

5.5.1 O Algoritmo K-VIZINHOS MAIS PRÓXIMOS

O algoritmo K-VIZINHOS MAIS PRÓXIMOS é parte de uma família de métodos de aprendizado conhecidos como *instance-based* (Aha, Kibler & Albert, 1991) (Mitchell, 1997, Capítulo 8). Os algoritmos de aprendizado *instance-based* são abordagens conceitualmente simples para aprender conceitos representados tanto por atributos quantitativos quanto qualitativos. O aprendizado nesses algoritmos consiste apenas em armazenar os dados de treinamento. Quando um novo exemplo é apresentado, um conjunto de exemplos similares é recuperado do conjunto de treinamento e utilizado para classificar o novo exemplo.

Como descrito previamente, uma limitação da abordagem *instance-based* é o alto custo de classificação de novos exemplos. Isso decorre do fato de que quase todo o proces-

samento ocorre durante a classificação dos novos exemplos, ao invés de ocorrer durante o período em que os exemplos de treinamento são vistos pela primeira vez. Dessa forma, métodos para indexar o conjunto de treinamento de forma eficiente são um importante aspecto prático para reduzir o custo computacional exigido durante a classificação de novos exemplos. Uma segunda limitação da abordagem *instance-based* é tipicamente considerar em igualdade de condições todos os atributos dos exemplos no momento de recuperar os exemplos de treinamento mais similares. Assim, se o conceito a ser aprendido depender somente de um subconjunto dos atributos disponíveis, então os exemplos que são realmente mais "similares" podem estar a uma grande distância.

Nas próximas seções é discutido o algoritmo K-VIZINHOS MAIS PRÓXIMOS em maiores detalhes. Na Seção 5.5.1.1 é descrita a implementação do algoritmo K-VIZINHOS MAIS PRÓXIMOS mais simples e mais freqüentemente utilizada. Na Seção 5.5.1.2 é descrito um melhoramento sobre a implementação básica do algoritmo. Esse melhoramento é obtido por meio da atribuição de pesos aos votos de cada um dos k exemplos mais próximos. Esses pesos são calculados de acordo com a distância entre cada um dos k exemplos e o exemplo a ser classificado. Na Seção 5.5.1.3 são descritas algumas funções de distância que podem ser utilizadas em substituição à distância euclidiana. Por fim, na Seção 5.5.1.4 é descrito brevemente o método de acesso M-tree utilizado neste trabalho para diminuir o tempo de classificação de novas instâncias.

5.5.1.1 O Algoritmo K-VIZINHOS MAIS PRÓXIMOS Básico

A versão básica do algoritmo K-VIZINHOS MAIS PRÓXIMOS assume que todos os exemplos correspondem a pontos no espaço M-dimensional $\Re^M$. Os vizinhos mais próximos de um novo exemplo são freqüentemente definidos em termos da distância euclidiana.

A distância entre dois exemplos E_i e E_j é denotada por $d(E_i, E_j)$. Para a distância euclidiana, $d(E_i, E_j)$ é definida pela Equação 5.1.

$$d(E_i, E_j) = \sqrt{\sum_{r=1}^{M}(x_{ir} - x_{jr})^2} \tag{5.1}$$

onde, segundo a notação definida na Seção 2.5 na página 18:

- M é o número de atributos presentes no conjunto de dados;
- x_{ir} é o valor assumido pelo *r-ésimo* atributo A_r do *i-ésimo* exemplo E_i.

Como mencionado previamente, para o algoritmo K-VIZINHOS MAIS PRÓXIMOS o atributo classe pode ser tanto um atributo qualitativo quanto quantitativo. Para atributos classe qualitativos deve-se considerar a tarefa de aprender uma função conceito f na forma $f : \Re^M \rightarrow C$, onde C é o conjunto finito $\{C_1, C_2, \ldots C_{Ncl}\}$ de valores que o atributo classe pode assumir. A versão básica do algoritmo K-VIZINHOS MAIS PRÓXIMOS é descrito no Algoritmo 5.1.

Algoritmo 5.1 Versão básica do algoritmo K-VIZINHOS MAIS PRÓXIMOS para problemas com classes qualitativas.

Require: $E = \{E_1, E_2, \ldots E_N\}$, um conjunto de treinamento com exemplos rotulados;
E_q, um exemplo de consulta a ser classificado;
k, o número de vizinhos mais próximos;
d, uma função de distância entre exemplos.

1: Seja $\hat{E} = \{\hat{E}_1, \hat{E}_2, \ldots \hat{E}_k\}$, o conjunto dos k exemplos $\hat{E}_j \in E$ mais similares a E_q, segundo a função de distância d

2:
$$\mathbf{h}(E_q) = \arg\max_{c \in C} \sum_{i=1}^{k} \delta(c, f(\hat{E}_i))$$
onde $\delta(a, b) = 1$, se $a = b$; e $\delta(a, b) = 0$, caso contrário

3: **return** $\mathbf{h}(E_q)$

O valor $\mathbf{h}(E_q)$ retornado pelo Algoritmo 5.1 como estimativa para $f(E_q)$ é somente a moda, isto é, o valor mais freqüente da função conceito f entre os k exemplos de treinamento mais próximos a E_q. Se $k = 1$ for escolhido, então o algoritmo 1-VIZINHO MAIS PRÓXIMO atribui a $\mathbf{h}(E_q)$ o valor de $f(E_i)$, no qual E_i é o exemplo de treinamento mais próximo a E_q. Para valores maiores de k, o algoritmo atribui a moda entre os k exemplos de treinamento mais próximos.

O algoritmo K-VIZINHOS MAIS PRÓXIMOS pode ser facilmente adaptado para problemas com classes quantitativas. Para isso é necessário fazer com que o algoritmo calcule a média entre os k exemplos de treinamento mais próximos, ao invés de calcular a moda. Mais precisamente, para aproximar um função conceito $f : \Re^M \rightarrow \Re$ é necessário substituir a equação na linha 2 do Algoritmo 5.1 pela equação

$$\mathbf{h}(E_q) = \frac{\sum_{i=1}^{k} f(\hat{E}_i)}{k} \tag{5.2}$$

5.5.1.2 O Algoritmo K-Vizinhos mais Próximos com Pesos

Um refinamento ao algoritmo K-VIZINHOS MAIS PRÓXIMOS básico é dar pesos à contribuição de cada um dos k vizinhos de acordo com a distância entre cada vizinho e o exemplo de consulta E_q. Nesse refinamento é dado maior peso aos vizinhos que estão mais próximos do exemplo de consulta, e menor peso para os mais distantes. Por exemplo, no Algoritmo 5.1 na página precedente, o qual prediz valores para classes qualitativas, pode-se pesar os votos de cada vizinho de acordo com o inverso do quadrado da distância do vizinho para E_q. Um pouco mais formalmente, isso pode ser feito substituindo a linha 2 do algoritmo pela Equação 5.3.

$$\mathbf{h}(E_q) = \arg\max_{c \in C} \sum_{i=1}^{k} \omega_i \delta(c, f(\hat{E}_i)) \tag{5.3}$$

onde

$$\omega_i = \frac{1}{d(E_q, \hat{E}_i)^2} \tag{5.4}$$

Para acomodar o caso no qual o exemplo de consulta E_q é igual a um dos exemplos de treinamento $\hat{E}_i$ e, dessa forma, o denominador $d(E_q, \hat{E}_i)^2 = 0$, deve-se fazer $\mathbf{h}(E_q) = f(\hat{E}_i)$. Caso existam diversos exemplos de treinamento iguais a E_q, deve-se atribuir a $\mathbf{h}(E_q)$ a classificação majoritária entre esses exemplos.

De forma similar, é possível adicionar pesos com base nas distâncias de cada um dos k vizinhos para o algoritmo que trabalha com atributo classe quantitativo. Para isso, deve-se substituir a linha 2 do Algoritmo 5.1 pela Equação 5.5.

$$\mathbf{h}(E_q) = \frac{\sum_{i=1}^{k} \omega_i f(\hat{E}_i)}{\sum_{i=1}^{k} \omega_i} \tag{5.5}$$

onde ω_i é definido pela Equação 5.4. Note que o denominador na Equação 5.5 é uma constante que normaliza as contribuições dos vários pesos.

Todas as variações do algoritmo K-VIZINHOS MAIS PRÓXIMOS discutidas até o momento consideram somente os k vizinhos mais próximos para classificar o novo exemplo. Uma vez que os pesos com base na distância foram introduzidos, não existe realmente nenhum problema em permitir que todos os exemplos de treinamento tenham influência na classificação de E_q, uma vez que exemplos muito distantes terão muita pouca influência em

$\mathbf{h}(E_q)$. A única desvantagem em considerar todos os exemplos é que esse novo algoritmo irá demandar um pouco mais de processamento. Se todos os exemplos de treinamento são considerados na classificação de um novo exemplo, então esse algoritmo é chamado de método *global*. Se somente os exemplos mais próximos são considerados, então o algoritmo é chamado de método *local*. Quando a Equação 5.5 na página oposta é aplicada para determinar $\mathbf{h}(E_q)$ como um método global, isto é, utilizando todos os exemplos de treinamento, o algoritmo é chamado de método de Shepard (Shepard, 1968).

A variante do algoritmo K-VIZINHOS MAIS PRÓXIMOS utilizada neste trabalho é um método local que utiliza pesos calculados por meio das distâncias. Nas próximas seções são descritos dois melhoramentos, implementados neste trabalho, sobre esse algoritmo. Primeiro, a substituição da distância euclidiana pela distância HVDM. Segundo, a implementação de uma estrutura de índice M-tree para acelerar o tempo de classificação de novos exemplos.

5.5.1.3 As Funções de Distância VDM, HEOM e HVDM

A distância euclidiana é amplamente utilizada e bastante apropriada para atributos quantitativos mas, freqüentemente, essa função de distância não manipula atributos qualitativos, também chamados de atributos simbólicos, apropriadamente.

Uma forma de lidar com conjuntos de dados com atributos qualitativos e quantitativos é usar uma função de distância heterogênea que utiliza funções de distância diferentes para diferentes tipos de atributos. Uma abordagem bastante difundida é usar a métrica *overlap*[8] para atributos qualitativos e a distância euclidiana para atributos quantitativos. Essa abordagem é conhecida como *Heterogeneous Euclidean-Overlap Metric*, ou simplesmente *HEOM*, e define a distância entre dois exemplos E_i e E_j conforme a Equação 5.6.

$$HEOM(E_i, E_j) = \sqrt{\sum_{r=1}^{M} d_a(x_{ir}, x_{jr})^2} \tag{5.6}$$

onde $d_a(x_{ir}, x_{jr})^2$ é a distância entre dois valores x_{ir} e x_{jr} de um mesmo atributo A_r. A função de distância d_a é definida pela Equação 5.7.

[8] O termo *overlap* é utilizado em inglês por ser amplamente difundido na comunidade.

$$d_a(x_{ir}, x_{jr}) = \begin{cases} 1, \text{ se } x_{ir} \text{ e/ou } x_{jr} \text{ forem desconhecidos; caso contrário} \\ overlap(x_{ir}, x_{jr}), \text{ se o atributo } A_r \text{ for qualitativo} \\ range\_normalized\_diff(x_{ir}, x_{jr}), \text{ se o atributo } A_r \text{ for quantitativo} \end{cases} \quad (5.7)$$

Para valores desconhecidos a distância máxima 1 é sempre considerada. A função *overlap* e a função $range\_normalized\_diff$ são definidas pela Equação 5.8 e pela Equação 5.9, respectivamente.

$$overlap(x_{ir}, x_{jr}) = \begin{cases} 0, \text{ se } x_{ir} = x_{jr} \\ 1, \text{ caso contrário} \end{cases} \quad (5.8)$$

$$range\_normalized\_diff(x_{ir}, x_{jr}) = \frac{|x_{ir} - x_{jr}|}{max_{A_r} - min_{A_r}} \quad (5.9)$$

onde max_{A_r} e min_{A_r} são os valores máximo e mínimo, respectivamente, observados no conjunto de treinamento para o atributo A_r. Isso significa que é possível que um novo exemplo ainda não visto possa ter um valor fora desse intervalo, produzindo uma diferença de valor maior que 1. Entretanto, esses valores costumam ser raros, e quando eles ocorrem, uma diferença maior pode ser aceitável.

A função de distância HEOM propõe uma abordagem excessivamente simplista para manipular atributos qualitativos. Essa abordagem falha por não utilizar informações adicionais providas pelos valores de um atributo qualitativo, informações essas que poderiam ajudar no processo de aprendizado.

A métrica *Value Difference Metric* — VDM foi proposta por Stanfill & Waltz (1986) e provê uma função de distância apropriada para atributos qualitativos. A métrica VDM considera a similaridade de classificação entre cada possível valor de um atributo para calcular a distância entre esses valores. Como resultado, uma matriz de distância é criada para cada atributo a partir do conjunto de treinamento. A distância $VDM(x_{ir}, x_{jr})$ entre dois valores x_{ir} e x_{jr} de um dado atributo qualitativo A_r é definida pela Equação 5.10.

$$VDM(x_{ir}, x_{jr}) = \sum_{l=1}^{Ncl} \left| \frac{N_{x_{ir}, C_l}}{N_{x_{ir}}} - \frac{N_{x_{jr}, C_l}}{N_{x_{jr}}} \right|^c \quad (5.10)$$

onde

- $N_{x_{ir}}$ é o número de exemplos no conjunto de treinamento que possuem valor x_{ir} para o atributo A_r;
- N_{x_{ir},C_l} é o número de exemplos no conjunto de treinamento que possuem valor x_{ir} para o atributo A_r e pertencem à classe C_l;
- Ncl é o número de valores que o atributo classe pode assumir no conjunto de dados;
- c é uma constante, geralmente 1 ou 2.

A métrica VDM considera dois valores similares se eles possuem classificações similares, isto é, se eles possuem correlações similares com a classe. Por exemplo, se um atributo `cor` pode assumir três valores, `vermelho`, `verde` e `azul`, e a aplicação é identificar se um objeto é uma maça ou não. O valor `vermelho` pode ser considerado mais próximo do valor `verde`, se comparado com as distâncias que separam os valores `vermelho` e `azul`, e `verde` e `azul`, pois o primeiro par de valores deve possuir correlações similares com a classe `maça`.

O algoritmo VDM original (Stanfill & Waltz, 1986) utiliza pesos para os atributos, os quais não estão incluídos na Equação 5.10 na página anterior. Algumas variantes da métrica VDM, como por exemplo, a proposta por Cost & Salzberg (1993), utilizam formas diferentes de atribuir pesos aos atributos.

Pode ser observado que a função VDM é uma função de distância, uma vez que ela possui as propriedades requeridas, ou seja:

1. $VDM(x_{ir}, x_{ir}) = 0$ e $VDM(x_{ir}, x_{jr}) > 0, x_{ir} \neq x_{jr}$ (não negatividade);
2. $VDM(x_{ir}, x_{jr}) = VDM(x_{jr}, x_{ir})$ (simetria);
3. $VDM(x_{ir}, x_{jr}) + VDM(x_{ir}, x_{lr}) \geq VDM(x_{ir}, x_{lr})$ (desigualdade triangular).

Se a métrica VDM for utilizada diretamente em atributos quantitativos, os quais podem assumir diversos valores, então $Nx_{ir} = 1$, e $N_{x_{ir},C_l} = 1$ para uma determinada classe C_l e 0 para todas as demais classes. Em adição, o novo exemplo provavelmente terá valores diferentes dos que ocorreram no conjunto de treinamento. Nesse caso, $N_{x_{ir},C_l} = 0 \; \forall \; C_l \in C$ e $N_{x_{ir}} = 0$, uma vez que $N_{x_{ir}}$ é a soma dos termos N_{x_{ir},C_l} para todas as classes, resultando em uma divisão por zero.

Mesmo se todos os valores presentes em um atributo quantitativo não forem todos diferentes, freqüentemente existem poucos valores iguais. Dessa forma, a amostra para cada valor é muito pequena para ser considerada confiável, tornando a medida de distância também pouco confiável. Como conseqüência desses problemas, é inapropriado utilizar a métrica VDM diretamente em atributos quantitativos.

Uma abordagem para solucionar o problema do uso da métrica VDM em atributos quantitativos é a *discretização* (Lebowitz, 1985; Kohavi & Sahami, 1996) desses atributos. Dessa forma, um atributo quantitativo pode ser discretizado e tratado como um atributo qualitativo pela métrica VDM. Entretanto, a discretização de atributos pode levar a perda de uma grande quantidade de informação disponível nos atributos quantitativos.

Se a distância euclidiana é inapropriada para atributos qualitativos, e a distância VDM é inapropriada para atributos quantitativos, então nenhuma delas é suficiente por si só para ser utilizada em um conjunto de dados que possui atributos qualitativos e quantitativos.

A função de distância *Heterogeneous Distance Function* — HVDM (Wilson & Martinez, 2000) é uma função de distância similar à função HEOM, exceto que ela utiliza a métrica VDM ao invés da métrica *overlap* para atributos qualitativos, e também são utilizadas normalizações diferentes se comparada com a distância HEOM. A função de distância HVDM é definida pela Equação 5.11.

$$HVDM(x_{ir}, x_{jr}) = \sqrt{\sum_{r=1}^{M} d_a(x_{ir}, x_{jr})^2} \tag{5.11}$$

no qual $d_a(x_{ir}, x_{jr})$ é a distância entre dois valores x_{ir} e x_{jr} de um dado atributo A_r, e é definida pela Equação 5.12.

$$d_a(x_{ir}, x_{jr}) = \begin{cases} 1, \text{ se } x_{ir} \text{ ou } x_{jr} \text{ forem valores desconhecidos; caso contrário} \\ normalized\_vdm(x_{ir}, x_{jr}), \text{ se } A_r \text{ for qualitativo} \\ normalized\_diff(x_{ir}, x_{jr}), \text{ se } A_r \text{ for quantitativo} \end{cases} \tag{5.12}$$

Uma vez que, em uma distribuição normal, aproximadamente 95% dos valores estão dispostos a uma distância de até dois desvios padrão da média, a diferença entre os valores numéricos pode ser dividida por 4 desvios padrão. Com isso, pode-se normalizar cada valor em um intervalo de tamanho aproximado 1. Dessa forma, a função $normalized\_diff$ é

definida como

$$normalized\_diff(x_{ir}, x_{jr}) = \frac{(|x_{ir} - x_{jr}|)}{4\sigma_{A_r}} \tag{5.13}$$

onde σ_{A_r} é o desvio padrão calculado a partir dos valores do atributo quantitativo A_r.

A função *normalized_vdm* é definida como

$$normalized\_vdm(x_{ir}, x_{jr}) = \sqrt{\sum_{l=1}^{Ncl} \left| \frac{N_{x_{ir},C_l}}{N_{x_{ir}}} - \frac{N_{x_{jr},C_l}}{N_{x_{jr}}} \right|^2} \tag{5.14}$$

Na implementação do algoritmo K-VIZINHOS MAIS PRÓXIMOS utilizada neste trabalho é possível selecionar qual função de distância deve ser utilizada. Nos experimentos descritos neste capítulo, para conjuntos de dados com atributos qualitativos e quantitativos, foi utilizada a função de distância HVDM. Para conjuntos de dados que possuem atributos quantitativos exclusivamente, foi utilizada a função de distância euclidiana.

5.5.1.4 Acelerando as Consultas com M-trees

M-trees (Ciaccia, Patella & Zezula, 1997) são estruturas de indexação capazes de suportar a execução de *consultas de similaridades*. Existem dois tipos básicos de consultas de similaridade:

Consulta por intervalo
: Dada uma *distância máxima de busca* e um exemplo de consulta, esta consulta de similaridade retorna todos os exemplos do conjunto de treinamento que estão a uma distância menor ou igual à distância máxima de busca do exemplo de consulta;

Consulta k-vizinhos mais próximos
: Dado um exemplo de consulta e um número k, essa consulta de similaridade retorna os k exemplos do conjunto de treinamento mais próximos do exemplo de consulta.

Uma estrutura M-tree é um *árvore métrica* (Uhlmann, 1991). Árvores métricas somente consideram as distâncias relativas entre os exemplos (ao invés das posições absolutas desses exemplos em um espaço multi-dimensional) para organizar e particionar o espaço métrico. Em um espaço métrico, a proximidade entre exemplos é definida por uma função de distância que satisfaz os postulados de não negatividade, simetria e desigualdade triangular.

Uma M-tree é uma árvore de crescimento "*bottom-up*" com nós de tamanho fixo, os quais correspondem a regiões restritas do espaço métrico. Os nós folhas armazenam todos os objetos indexados, isto é, todos os exemplos, e os nós internos armazenam os chamados *objetos de roteamento*. Um objeto de roteamento é uma cópia de um exemplo do conjunto de dados para o qual é atribuído um papel de roteamento.

Para cada objeto de roteamento O_r existe um ponteiro associado, denotado por $ptr(T(O_r))$, o qual referencia a raiz de uma sub-árvore, $T(O_r)$, chamada de *árvore de cobertura* de O_r. Todos os objetos na árvore de cobertura de O_r estão a uma distância máxima $r(O_r)$ do objeto de roteamento O_r, sendo $r(O_r) > 0$. $r(O_r)$ é chamado de *raio de cobertura* de O_r. Cada entrada de um nó interno de umz M-tree é composto por uma tupla

$$(O_r, ptr(T(O_r)), r(O_r)).$$

Uma entrada em um nó folha de uma M-tree é mais simples do que uma entrada de um objeto de roteamento. Isso ocorre pois um objeto folha O_l não possui sub-árvore associada e, conseqüentemente, não possui raio de cobertura. Ciaccia, Patella & Zezula (1997) provê informações adicionais sobre a estrutura de uma M-tree e algoritmos detalhados para realizar consultas de similaridade e construir uma M-tree.

Na Figura 5.2 na página oposta é apresentado um exemplo de uma M-tree com dois nós de roteamento e três nós folhas, e na Figura 5.3 na página 108 é mostrado uma representação gráfica para a estrutura dessa M-tree.

Como mencionado previamente, uma M-tree cresce de forma "*bottom-up*". Quando um novo objeto é inserido, a M-tree tenta localizar o nó folha mais adequado para acomodá-lo. Se o nó folha está cheio, então um novo nó folha é alocado. Os objetos do nó folha cheio são particionados entre esse nó e o novo nó folha, e dois objetos são promovidos para o nó pai. Se o nó pai está cheio, então um algoritmo similar é aplicado, isto é, um novo nó é alocado no mesmo nível que o nó cheio, os objetos são particionados, e dois objetos são promovidos para o nó pai. Quando o nó raiz é dividido, um novo nó raiz é criado e a M-tree cresce em um nível.

As implementações dos métodos responsáveis por promover dois objetos para o nó pai, e por particionar os objetos restantes entre os dois nós filhos definem o que é chamado de *política de divisão*. Existem diversas alternativas para implementar esses dois métodos. Na implementação utilizada neste trabalho, foi escolhido utilizar um algoritmo baseado

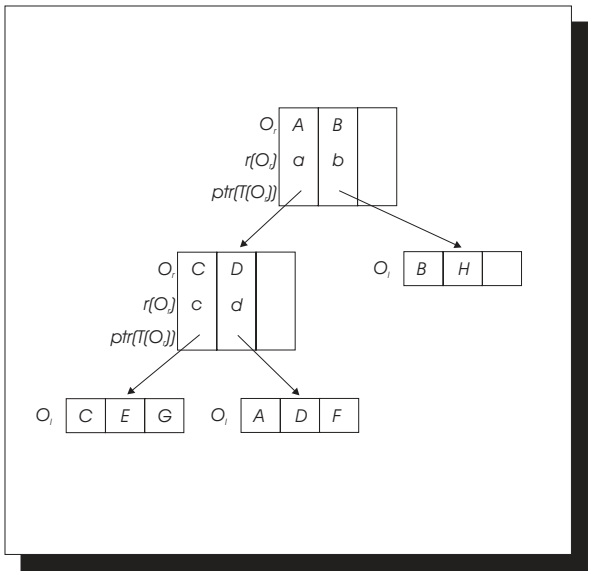

Figura 5.2: Exemplo de uma estrutura M-tree.

em *Minimal Spanning Trees* — MST, como proposto por Jr., Traina, Seeger & Faloutsos (2000) para ser utilizado em uma variante das M-trees conhecida como *Slim-trees*.

Uma política de divisão ideal deve promover e particionar os objetos de forma que duas regiões obtidas tenham mínimo *volume* e mínima *intersecção*. Esses critérios têm como objetivo melhorar a efetividade dos algoritmos de busca, uma vez que regiões com pouco volume levam a árvores bem agrupadas e reduzem a quantidade de espaço sem nenhum objeto presente, e pouca (possivelmente nenhuma) intersecção entre regiões reduz o número de caminhos que precisam ser percorridos para responder a uma consulta.

5.6 Como os Sistemas de Aprendizado C4.5 e CN2 Tratam Valores Desconhecidos

Os sistemas de aprendizado C4.5 (Quinlan, 1988) e CN2 (Clark & Boswell, 1991) são dois algoritmos de AM simbólicos, amplamente conhecidos pela comunidade, que induzem conceitos proposicionais: árvores de decisão e conjuntos de regras, respectivamente. Esses algoritmos foram selecionados neste trabalho por serem considerados dois dos melhores algoritmos de aprendizado com essas características.

O sistema C4.5 possui um efetivo algoritmo interno para tratar valores desconhecidos, uma vez que um recente estudo comparativo com outros métodos simples para

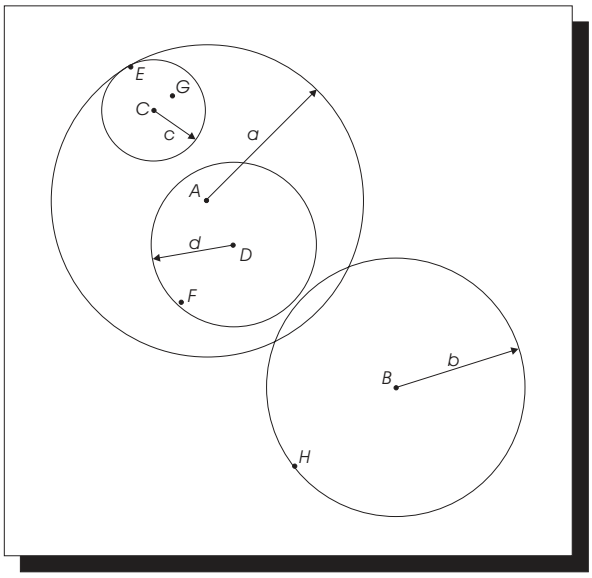

Figura 5.3: Representação gráfica da M-tree apresentada na Figura 5.2.

tratamento de valores desconhecidos, concluiu que o C4.5 era um dos melhores métodos (Grzymala-Busse & Hu, 2000).

O sistema C4.5 utiliza uma abordagem probabilística para tratar valores desconhecidos. Os valores desconhecidos podem estar presentes em qualquer atributo, exceto no atributo classe, tanto no conjunto de treinamento quanto no conjunto de teste.

Dado um conjunto de exemplos de treinamento, T, o sistema C4.5 encontra um teste apropriado, com base em um único atributo, que possua um ou mais resultados mutuamente exclusivos $R_1, R_2, \ldots R_s$. T é particionado em sub-conjuntos $T_1, T_2, \ldots T_s$, tal que T_i contém todos os exemplos em T que satisfazem o teste com resultado R_i. O mesmo algoritmo é aplicado para cada sub-conjunto T_i, $i = 1 \ldots s$, até que um critério de parada seja satisfeito.

O C4.5 utiliza a medida *information gain ratio* para escolher um bom teste para particionar os exemplos. Se existem valores desconhecidos em um atributo A, então o sistema C4.5 utiliza o sub-conjunto com todos os valores conhecidos de A para calcular o ganho de informação.

Uma vez que um teste com base em um atributo A é escolhido, o sistema C4.5 utiliza uma abordagem probabilística para particionar os exemplos com valores desconhecidos no atributo A. Quando um exemplo em T com valor conhecido é associado a um sub-conjunto T_i, isso pode ser entendido como a probabilidade desse exemplo pertencer a T_i ser 1, e para todos as demais partições a probabilidade é 0. Quando um valor não é conhecido, então

somente uma declaração probabilística mais fraca pode ser feita. O C4.5 associa a cada exemplo em T_i um *peso* representando a probabilidade desse exemplo pertencer a T_i. O peso para a partição T_i é a probabilidade desse exemplo pertencer a T_i. Essa probabilidade é estimada por meio da soma dos pesos dos exemplos em T que satisfazem o teste com resultado R_i, dividido pela soma de pesos dos casos em T com valores conhecidos para o atributo A.

O sistema CN2 utiliza uma abordagem similar à abordagem adotada pelo sistema C4.5 para tratar valores desconhecidos. O sistema CN2 induz regras na forma

$$\textbf{if } \text{<condições>} \textbf{ then } \text{<classe} = C_i\text{>} \; [\#C_1, \#C_2, \ldots, \#C_{Ncl}]$$

na qual <condições> é uma conjunção de condições que envolvem os atributos do conjunto de dados, e <classe = C_i> é a predição realizada pela regra. O número entre colchetes, $\#C_j$, indica o número de exemplos do conjunto de treinamento cobertos pela regra para cada classe C_j.

Esses valores podem ser utilizados pelo CN2 para realizar uma classificação probabilística, caso o usuário deseje. Nesse caso, o CN2 responde com as probabilidades de um novo exemplo pertencer a cada uma das classes. Também, esses valores são utilizados pelo algoritmo que avalia regras não ordenadas para verificar qual é a classe mais provável com base em todas as regras que foram disparadas para um novo exemplo. Se mais de uma classe é predita pelas regras disparadas, então os valores $\#C_j$ são somados separadamente para cada classe utilizando somente as regras disparadas. O CN2 responde com a classe mais provável, isto é, aquela que possui o maior valor total de exemplos cobertos pelo conjunto de todas as regras disparadas.

No caso da indução de regras não ordenadas pelo CN2, se o conjunto de treinamento possui exemplos com valores desconhecidos, então o indutor, após a indução do conjunto de regras, realiza uma segunda passagem sobre o conjunto de regras atualizando os valores $\#C_j$. Para cada exemplo com valores desconhecidos, o CN2 verifica quais regras cobrem esse exemplo. Para que uma regra cubra um exemplo com valores desconhecidos basta que o exemplo com valores desconhecidos satisfaça as condições da regra e, caso a regra possua condições envolvendo atributos cujos valores não sejam conhecidos para esse exemplo, então essas condições são automaticamente satisfeitas. O exemplo com valores desconhecidos é, então dividido em frações iguais, as quais são contabilizadas nos valores $\#C_j$. Por exemplo, se três regras são disparadas para um determinado exemplo com valores desconhecidos, então $\frac{1}{3}$ do exemplo é contabilizado para cada regra no respectivo

valor $\#C_j$, o qual depende da classe a qual o exemplo pertence.

5.7 Análise Experimental

Como mencionado anteriormente, o principal objetivo dos experimentos conduzidos neste trabalho é avaliar a eficiência do algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação para tratamento de valores desconhecidos, e comparar o desempenho desse método com o desempenho obtido pelos algoritmos internos utilizados pelos indutores C4.5 e CN2 para aprender na presença de valores desconhecidos, e com o desempenho do método IMPUTAÇÃO PELA MÉDIA OU MODA.

Nos experimentos realizados, os valores desconhecidos foram artificialmente implantados nos conjunto de dados, em diferentes taxas e atributos. Os desempenhos de todos os quatro métodos de tratamento de valores desconhecidos foram comparados utilizando taxas de erro obtidas por meio do método de reamostragem *k-fold cross-validation*. Em particular, este trabalho visa analisar o comportamento desses métodos quando a quantidade de valores desconhecidos é alta, uma vez que alguns pesquisadores têm reportado encontrar bases de dados nas quais mais de 50% dos valores eram desconhecidos (Lakshminarayan, Harp & Samad, 1999).

Os experimentos foram realizados com os seguintes conjuntos de dados do repositório UCI (Blake & Merz, 1998):

Bupa

Este conjunto de exemplos foi uma contribuição de R. S. Forsyth ao repositório UCI. O problema consiste em predizer se um paciente do sexo masculino tem desordens hepáticas com base em vários testes de sangue e na quantidade de álcool consumida.

Pima

Este conjunto de exemplos foi doado por V. Sigillito do Laboratório de Física Aplicada, Univesidade Johns Hopkins University. É um subconjunto de uma base de dados maior que é mantida pelo Instituto Nacional de Diabetes e Doenças Digestivas e Renais nos Estados Unidos.

Todas as pacientes são mulheres com pelo menos 21 anos de idade de descendência indígena pima vivendo próximas a Phoenix, Arizona, EUA. O problema consiste em predizer se uma paciente apresentará um resultado positivo para diabetes de acordo com os critérios da Organização Mundial da Saúde a partir de medidas fisiológicas

e resultados de testes médicos.

CMC

Este conjunto de exemplos consiste em um subconjunto do estudo de eficácia contraceptiva da Indonésia, realizado em 1987. As amostras são de mulheres casadas que não estavam grávidas ou não sabiam se estavam grávidas na ocasião da entrevista. O problema consiste em predizer a escolha do método contraceptivo de uma mulher (nenhum, método de curta duração, método de longa duração) com base nas características demográficas e sócio-econômicas.

CRX

Este conjunto de exemplos está relacionado com aplicações de cartões de crédito. Todos os nomes de atributos e valores foram alterados para símbolos sem significado para proteger a confidencialidade dos dados.

Breast

Este conjunto de exemplos foi obtido dos hospitais da Universidade de Wisconsin, Madison, pelo Dr. William H. Wolberg. O problema consiste em predizer se uma amostra de tecido retirado da mama de um paciente é um câncer benigno ou maligno. A cada amostra foi atribuído um vetor 9-dimensional. Cada componente encontra-se no intervalo de 1 a 10, com 1 significando estado normal e 10 o estado mais anormal. O grau de quão maligno é o tecido foi determinado por uma biópsia de uma amostra de tecido retirada da mama do paciente. Um diagnóstico benigno é confirmado por biópsia ou por exames periódicos, dependendo da escolha do paciente.

Sonar

Este conjunto de exemplos foi usado por Gorman & Sejnowski (1988) no estudo de classificação de sinais de sonar utilizando uma rede neural. O problema consiste em discriminar entre sinais de sonar que representam um cilindro de metal, daqueles que representam uma rocha ligeiramente cilíndrica.

O conjunto de exemplos contém 111 exemplos obtidos por varredura de sonar de um cilindro de metal em vários ângulos e sob várias condições; contém também 97 exemplos obtidos por varredura de rochas sob as mesmas condições. Cada exemplo é um conjunto de 60 números reais entre 0 e 1. Cada número representa a energia em uma banda de freqüência particular integrada sobre um certo período de tempo. A classe associada a cada exemplo contém a letra `R` se o objeto é uma rocha e `M` se o objeto é uma mina (cilindro de metal).

Os conjuntos de dados **Bupa**, **Pima**, **CMC** e **CRX** não possuem valores desconhecidos. Os conjuntos de dados **Breast** e **Sonar** possuem poucos valores desconhecidos (no

total 16 casos ou 2,28%, e 37 casos ou 5,36%, respectivamente) os quais foram removidos antes do início dos experimentos. A principal razão para não utilizar dados com valores desconhecidos é a preocupação em ter todo o controle sobre os valores desconhecidos nos conjuntos de dados. Por exemplo, é desejável que os conjuntos de teste não possuam valores desconhecidos. Caso algum conjunto de teste possua valores desconhecidos, então a habilidade do indutor em classificar exemplos com valores desconhecidos corretamente pode influenciar nos resultados. Essa influência não é desejável uma vez que o objetivo é analisar a viabilidade dos métodos de tratamento de valores desconhecidos.

Um outro problema relevante, embora não tratado neste trabalho, é como tornar um sistema de aprendizado eficiente em classificar novos exemplos com muitos valores desconhecidos. Nesse cenário, valores desconhecidos estão presentes nos casos de teste, ao invés de estarem presentes somente nos casos de treinamento. Por exemplo, em diagnóstico médico, quando um paciente inicia um tratamento, pode ser interessante realizar algum diagnóstico antes de realizar algum exame custoso ou muito demorado. Nesse caso, o exemplo, isto é, o novo paciente, deve ser classificado com muitos valores desconhecidos. Zheng & Low (1999) analisam o uso de *ensembles* para aumentar a robustez dos algoritmos de aprendizado em classificar exemplos com valores desconhecidos.

Na Tabela 5.1 na página oposta são apresentadas algumas das principais características dos conjuntos de dados utilizados neste estudo. Nela são apresentados, para cada conjunto de dados, o número de exemplos (#Exemplos), o número e percentual de exemplos duplicados (que aparecem mais de uma vez) e conflitantes (com os mesmos valores de atributos, mas com classe diferente), o número de atributos (#Atributos), o número de atributos quantitativos e qualitativos, a distribuição da classe e o erro majoritário. Essas informações foram obtidas utilizando o utilitário *info* da biblioteca $\mathcal{MLC}$++ (Kohavi, Sommerfield & Dougherty, 1997).

Na Figura 5.4 na página 114 é ilustrada a metodologia utilizada nos experimentos. Inicialmente, cada conjunto de dados foi particionado em 10 pares de conjuntos de treinamento e teste por meio da aplicação do método de reamostragem *10-fold cross-validation*. Para cada iteração do método de reamostragem, valores desconhecidos foram implantados artificialmente no conjunto de treinamento. Copias dos conjuntos de treinamento com valores desconhecidos foram fornecidas diretamente para os sistemas C4.5 e CN2. Dois classificadores foram induzidos e o erro medido no conjunto de teste. Logo após, outras cópias dos conjuntos de treinamento com valores desconhecidos foram tratadas pelo método imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS e pelo método IMPUTAÇÃO PELA MÉDIA OU MODA. Os conjuntos de treinamento tratados foram fornecidos aos

Conjunto de dados	#Exemplos	#Duplicados ou conflitantes (%)	#Atributos (quanti., quali.)	Classes	% Classes	Erro Majoritário
Bupa	345	4 (1,16%)	6 (6,0)	1 2	42,03% 57,97%	42,03% para a classe 2
CMC	1473	115 (7,81%)	9 (2,7)	1 2 3	42,70% 22,61% 34,69%	57,30% para a classe 1
Pima	769	1 (0,13%)	8 (8,0)	0 1	65,02% 34,98%	34,98% para a classe 0
CRX	690	0 (0,00%)	15 (6,9)	+ -	44,50% 55,50%	44,50% para a classe -
Breast	699	8 (1,15%)	9 (9,0)	benign malignant	65,52% 34,48%	34,48% para a classe benign
Sonar	208	0 (0,00%)	60 (60,0)	M R	53,37% 46,63%	46,63% para a classe M

Tabela 5.1: Descrição resumida dos conjuntos de dados.

indutores C4.5 e CN2 e as taxas de erro medidas nos respectivos conjuntos de teste.

Ao final das 10 iterações do método *10-fold cross-validation*, a taxa de erro verdadeira de cada método de tratamento de valores desconhecidos pode ser estimada por meio do cálculo da média das taxas de erro em cada iteração. Por fim, o desempenho dos indutores C4.5 e CN2 aliados ao método de imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS pode ser analisado e comparado com os desempenhos dos métodos utilizados internamente pelos sistemas C4.5 e CN2 para aprender na presença de valores desconhecidos, e com o desempenho dos sistemas C4.5 e CN2 aliados ao método IMPUTAÇÃO PELA MÉDIA OU MODA.

Para inserir os valores desconhecidos nos conjuntos de treinamento, alguns atributos devem ser escolhidos, e parte dos valores desses atributos devem ser escolhidos para serem modificados para desconhecido. Neste experimento foi escolhido inserir valores desconhecidos nos atributos mais representativos de cada conjunto de dados. Essa decisão foi tomada pois deseja-se medir a efetividade dos métodos de tratamento de valores desconhecidos. Tal efetividade não pode ser medida se os atributos tratados forem não representativos, os quais provavelmente não seriam incorporados ao classificador pelo sistema de aprendizado.

Uma vez que encontrar os atributos mais representativos de um conjunto de dados não é uma tarefa trivial, foram utilizados os resultados de (Lee, Monard & Baranauskas, 1999) para selecionar os três atributos mais relevantes de um conjunto de dados segundo diversos métodos de seleção de atributos tais como *wrappers* e filtros.

Com relação à quantidade de valores desconhecidos a serem inseridos nos conjuntos de treinamento, deseja-se analisar o comportamento de cada um dos métodos de tratamento com diferentes quantidades de valores desconhecidos. Dessa forma, os valores desconhecidos foram inseridos nas seguintes porcentagens: 10%, 20%, 30%, 40%, 50% e

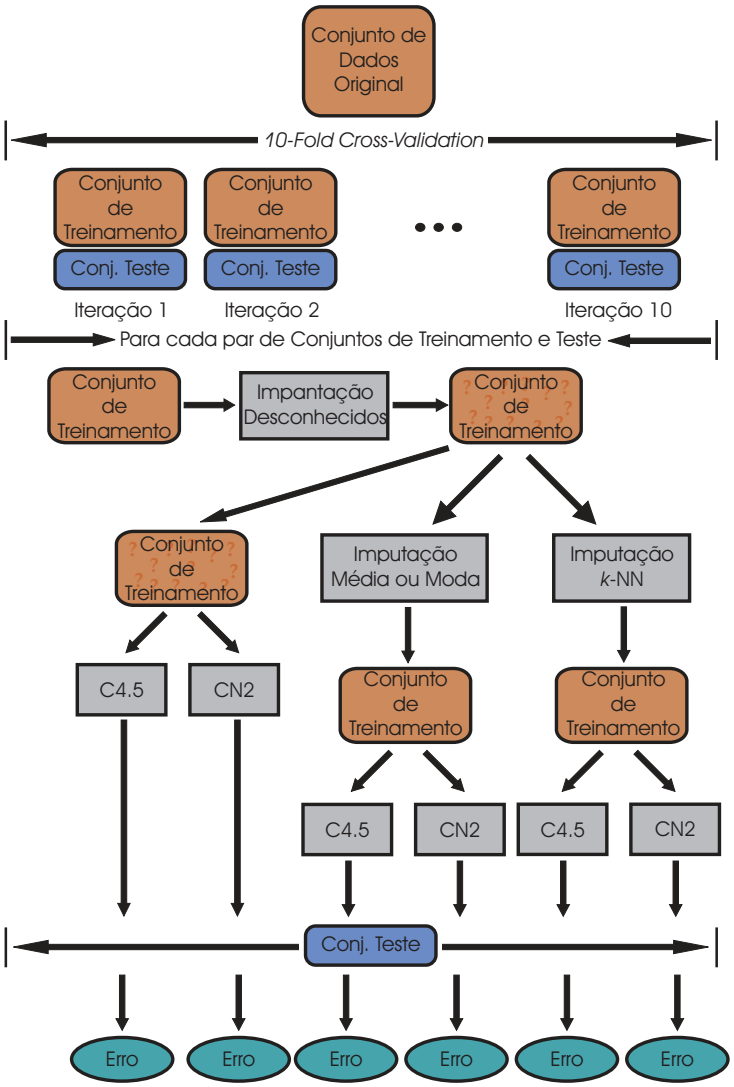

Figura 5.4: Representação gráfica da metodologia utilizada nos experimentos.

60% do total de exemplos no conjunto de treinamento. Os valores desconhecidos foram inseridos em um único atributo, em dois atributos e, por fim, nos três atributos selecionados como mais representativos.

Embora os valores desconhecidos possam ser inseridos em diferentes distribuições, decidiu-se inserir os valores de forma completamente aleatória (MCAR). Dessa forma, a distribuição dos valores desconhecidos não está sob o controle do experimento, impedindo assim que os valores desconhecidos sejam inseridos de forma que beneficiem um ou outro método.

Os valores desconhecidos foram substituídos por valores estimados utilizado 1, 3, 5, 10, 20, 30, 50 e 100 vizinhos mais próximos, além da substituição pela média ou moda

do atributo. Na tentativa de não tornar este capítulo demasiadamente longo, somente os resultados com 10 vizinhos mais próximos, identificados como 10-NNI[9], são apresentados nesta seção. Em (Batista & Monard, 2003c) é feita uma extensa apresentação de todos experimentos e dos resultados obtidos em cada um dos experimentos.

Nas próximas seções são elucidadas algumas decisões tomadas a respeito da metodologia utilizada neste trabalho. Inicialmente, na Seção 5.7.1 é explicado como foram selecionados três atributos como os três mais relevantes de cada conjunto de dados. Logo após, na Seção 5.7.2 são apresentados alguns resultados experimentais conduzidos para identificar um valor para o parâmetro k (número de vizinhos mais próximos) que fosse apropriado para a maioria dos conjuntos de dados utilizados. Por fim, na Seção 5.7.3 os resultados dos experimentos são apresentados e discutidos, para cada conjunto de dados.

5.7.1 Identificação de Atributos Relevantes

Como explicado anteriormente, deseja-se selecionar os atributos mais representativos de cada conjunto de dados para posteriormente inserir valores desconhecidos nesses atributos. A princípio, a seleção dos atributos mais representativos pode aumentar a probabilidade desses atributos serem incluídos no classificador induzido. Caso contrário, a análise pode ser comprometida tratando atributos não representativos que não serão incorporados no classificador pelo sistema de aprendizado.

Entretanto, não há garantias de que esses atributos serão incorporados pelos indutores nos classificadores induzidos nos experimentos. A existência de um atributo com informação similar (alta correlação) com um dos atributos selecionados pode fazer com que o indutor decida não utilizar o atributo selecionado no classificador induzido.

Uma vez que encontrar os atributos mais representativos em um conjunto de dados não é uma tarefa trivial, foram utilizados os resultados de (Lee, Monard & Baranauskas, 1999) na seleção de três atributos para cada conjunto de dados, como os atributos mais relevantes desses conjuntos. A seleção foi feita de acordo com os resultados obtidos por diversos métodos de seleção de atributos. Foram utilizadas as seguintes abordagens:

Filtro

Um método filtro é um método de seleção de atributos que é aplicado para selecionar os atributos antes que ocorra o processo de indução. Dessa forma, o método filtro utilizado pode ser independente do algoritmo de aprendizado utilizado para aprender

[9] *10-Nearest Neighbour Imputation.*

Conjunto de Dado	Atributos Selecionados		
	Identificador	Posição	Tipo
Bupa	Gammagt	4	real
	Sgpt	2	real
	Drinks	5	real
CMC	Nchi	3	real
	Wage	0	real
	Wedu	1	nominal
Pima	Plasma	1	real
	Body	5	real
	Number	0	real
CRX	A8	8	nominal
	A9	9	real
	A14	14	inteiro
Breast Cancer	Uniformity of Cell Size	1	inteiro
	Bare Nuclei	5	inteiro
	Clump Thickness	0	inteiro
Sonar	A10	10	inteiro
	A0	0	inteiro
	A26	26	inteiro

Tabela 5.2: Atributos selecionados como os mais representativos de cada conjunto de dados.

o conceito. Os indutores C4.5 e ID3 (Quinlan, 1986) foram utilizados como filtros, além do algoritmo *column importance* provido pelo aplicativo MineSet™ (Rathjens, 1996).

Wrapper

O método *wrapper* realiza uma busca por um subconjunto de atributos representativos por meio da adição e remoção de atributos no conjunto de dados e da medição da taxa de erro do classificador resultante. A busca realizada pode ser para frente, começando com um conjunto vazio de atributos e adicionando atributos a cada iteração; ou para trás, começando com todos os atributos e removendo atributos em cada passo, até que um critério de parada seja aplicado. A abordagem *wrapper* utiliza um indutor como uma caixa preta. Foram utilizados os indutores C4.5, C4.5 Rules e CN2 como caixas pretas, e também foram utilizadas as buscas para frente e para trás.

Foram identificados como atributos mais relevantes aqueles atributos mais freqüentemente selecionados pelos métodos de seleção. Quando possível, tentou-se dar uma ordem de importância aos atributos, da seguinte forma: entre os atributos identificados como mais relevantes, aquele mais freqüentemente selecionado pelos métodos de seleção foi escolhido como o mais relevante, o segundo mais freqüentemente selecionado foi escolhido o segundo mais relevante, e assim por diante. Na Tabela 5.2 são apresentados os atributos selecionados em cada conjunto de dados, ordenados por relevância.

5.7.2 Estimando um Bom Valor para o Parâmetro k

O algoritmo K-VIZINHOS MAIS PRÓXIMOS, como os demais sistemas de aprendizado, possui parâmetros que precisam ser ajustados para otimizar o seu desempenho. O principal parâmetro do algoritmo K-VIZINHOS MAIS PRÓXIMOS é o número de vizinhos, k, utilizados para fazer uma predição.

Como conseqüência da complexidade dos dados tratados em AM e KDD, o valor ótimo do parâmetro k pode variar entre os diferentes conjuntos de dados analisados. Entretanto, ainda é possível encontrar um valor para esse parâmetro que conduz a um desempenho satisfatório do algoritmo K-VIZINHOS MAIS PRÓXIMOS para a maioria dos domínios tratados. Uma vez encontrado, esse valor pode ser utilizado como valor *default* na implementação do algoritmo para tratamento de valores desconhecidos.

Como descrito previamente, nos experimentos conduzidos neste trabalho, valores desconhecidos foram implantados artificialmente nos conjuntos de treinamento em um, dois e três dos atributos selecionados, em diferentes proporções (de 10% a 60%) e tratados com 1, 3, 5, 10, 20, 30, 50 e 100 vizinhos mais próximos, e também com a imputação pela média ou moda.

Como os valores desconhecidos foram implantados artificialmente, o valor real de cada valor desconhecido é conhecido. Dessa forma, o erro entre os valores reais e os valores preditos pode ser medido. Médias e desvios padrão podem ser utilizados como estatísticas para avaliar o desempenho de cada valor do parâmetro k. Também, o erro obtido pelo método IMPUTAÇÃO PELA MÉDIA OU MODA pode ser utilizado como base para analisar se o processamento gasto pelo algoritmo K-VIZINHOS MAIS PRÓXIMOS é válido, ou se um método mais simples poderia prover resultados similares.

Existem diversas medidas que podem ser utilizadas para medir o erro entre os valores reais e os valores preditos. As principais medidas são (Weiss & Indurkhya, 1998):

Classificação

Para problemas de classificação a medida mais utilizada é a taxa de erro, Err, a qual pode ser definida como

$$Err = \frac{\sum_{i=1}^{N} \varepsilon(\mathbf{h}(E_i), f(E_i))}{N} \tag{5.15}$$

onde $\varepsilon(a, b) = 1$, se $a \neq b$; e $\varepsilon(a, b) = 0$, caso contrário;

Regressão

Para problemas de regressão existem duas medidas mais freqüentemente utilizadas: a diferença média absoluta (mad[10]) e o erro médio quadrático (mse[11]). Essas medidas são definidas pelas Equações 5.16 e 5.17, respectivamente.

$$mad = \frac{1}{N}\sum_{i=1}^{N}|\mathbf{h}(E_i) - f(E_i)| \tag{5.16}$$

$$mse = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{h}(E_i) - f(E_i))^2 \tag{5.17}$$

Geralmente, o erro medido utilizando mad é um pouco menor do que a raiz quadrada do erro mse.

Neste trabalho, é utilizada a taxa de erro Err — Equação 5.15 — para atributos qualitativos, e o erro médio quadrático mse — Equação 5.17 — para atributos quantitativos.

Uma vez que três atributos foram selecionados como os mais representativos em cada conjunto de dados, seis gráficos com os resultados obtidos foram gerados para cada conjunto de dados. Por exemplo, para o conjunto de dados **Breast** os atributos 1, 5 e 0[12] foram selecionados. Isso significa que os valores desconhecidos foram implantados no conjunto de dados **Breast** em três configurações: valores desconhecidos inseridos somente no atributo 1; valores desconhecidos inseridos nos atributos 1 e 5; e, por fim, valores desconhecidos inseridos nos atributos 1, 5 e 0. Quando os valores desconhecidos são inseridos somente em um atributo (por exemplo, o atributo 1), então o erro médio é medido somente para esse atributo. Quando valores desconhecidos são inseridos em dois atributos (por exemplo, os atributos 1 e 5), então dois erros médios são medidos, um para cada atributo. Por fim, quando os valores desconhecidos são inseridos nos três atributos selecionados (por exemplo, 1, 5 e 0), então três erros médios são medidos.

Nesta seção são apresentados três gráficos como resultado das análises dos conjuntos de dados **Bupa**, **Pima** e **Breast**, os demais conjuntos de dados apresentam padrões simi-

[10] *Mean absolute difference.*

[11] *Mean squared error.*

[12] Por simplicidade, este trabalho faz referência às posições relativas dos atributos no conjunto de dados correspondente, ao invés de fazer referência ao identificador do atributo. Na Tabela 5.2 na página 116 são relacionados os identificadores dos atributos selecionados com as suas posições relativas nos respectivos conjuntos de dados.

lares. Os demais resultados não incluídos neste trabalho, inclusive tabelas com resultados na forma numérica, podem ser encontrados em (Batista & Monard, 2003c).

Nas Figuras 5.5, 5.6 e 5.7 são apresentados os valores dos erros *mse* para diversos valores do parâmetro k.

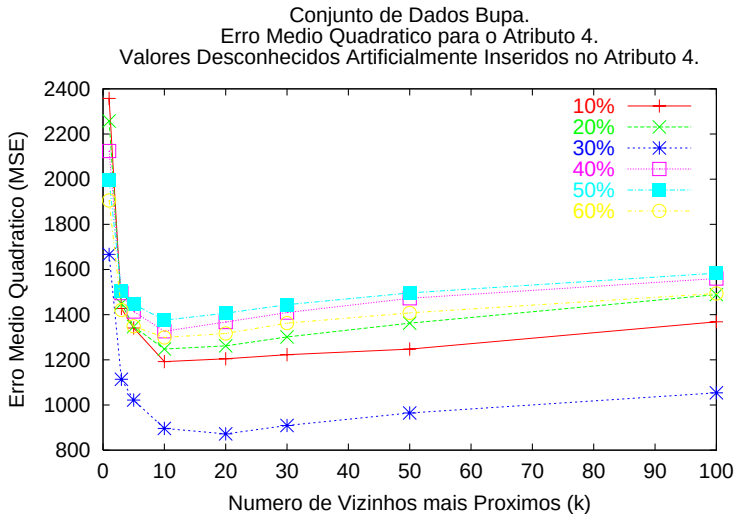

Figura 5.5: Conjunto de dados **Bupa**. Erro *mse* medido sobre o atributo 4 para diversos valores do parâmetro k do método de imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS. Valores desconhecidos inseridos no atributo 4. IMPUTAÇÃO PELA MÉDIA OU MODA obteve erros *mse* no intervalo $[1616.44 \pm 56.69, 1704.55 \pm 118.03]$.

De uma forma geral, os gráficos apresentam uma mesma tendência. Geralmente, 1 e 3 vizinhos mais próximos não fornecem os melhores resultados. Aparentemente, 50 e 100 parecem ser valores muito altos de vizinhos que não levam necessariamente a uma melhora no desempenho do algoritmo. Os melhores resultados são geralmente obtidos por 5, 10, 20 ou 30 vizinhos mais próximos. Com o objetivo de manter a busca pelos vizinhos mais próximos menos computacionalmente intensiva, foi escolhido 10 vizinhos mais próximos como o parâmetro *default* do método de imputação[13].

5.7.3 Resultados Experimentais

Nesta seção são apresentados os experimentos comparativos entre os métodos de tratamento de valores desconhecidos escolhidos. Nos resultados são mostradas as taxas de

[13]Se uma implementação tradicional do método K-VIZINHOS MAIS PRÓXIMOS for utilizada, então a busca irá requerer uma passagem completa pelo conjunto de treinamento independentemente do valor do parâmetro k. Como a implementação utilizada neste trabalho utiliza uma estrutura M-tree, um valor menor para k pode fazer com que a M-tree pode alguns ramos, tornando a busca mais rápida.

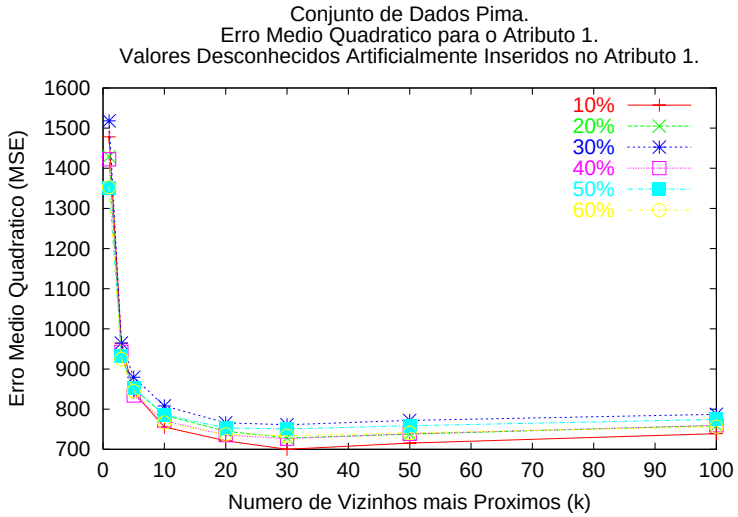

Figura 5.6: Conjunto de dados **Pima**. Erro *mse* medido sobre o atributo 1 para diversos valores do parâmetro k do método de imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS. Valores desconhecidos inseridos no atributo 1. IMPUTAÇÃO PELA MÉDIA OU MODA obteve erros *mse* no intervalo $[989.81 \pm 29.45, 1044.24 \pm 50.58]$.

erro média obtidas pelos classificadores induzidos pelos sistemas C4.5 e CN2 utilizando *10-fold cross-validation*. Para cada conjunto de dados, seis gráficos são mostrados. Cada gráfico compara o desempenho obtido pelos métodos de tratamento de valores desconhecidos com diferentes quantidades de valores desconhecidos implantados em um conjunto de atributos. Nos resultados apresentados, os seguintes rótulos são utilizados para identificar os métodos de tratamento de valores desconhecidos:

Sem Imputação
: Indica que o classificador foi gerado com dados com valores desconhecidos. Portanto, não houve um tratamento de valores desconhecidos prévio, e fica a cargo do indutor lidar com esses valores;

10-NNI
: Os valores desconhecidos são substituídos por valores estimados segundo o método de imputação 10-NNI. Após o passo de imputação dos valores desconhecidos, um classificador é induzido sobre um conjunto de treinamento sem valores desconhecidos;

Média ou Moda
: O método IMPUTAÇÃO PELA MÉDIA OU MODA é aplicado aos dados antes do processo de indução. O classificador é induzido a partir de um conjunto de treinamento completo.

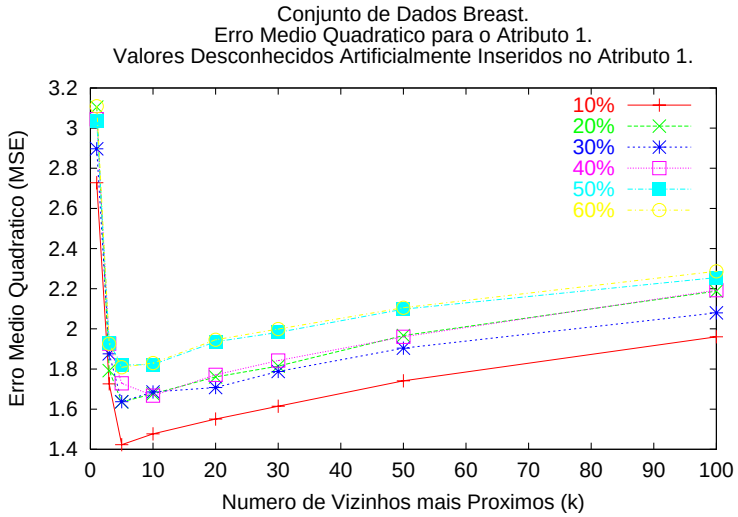

Figura 5.7: Conjunto de dados **Breast**. Erro *mse* medido sobre o atributo 1 para diversos valores do parâmetro k do método de imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS. Valores desconhecidos inseridos no atributo 1. IMPUTAÇÃO PELA MÉDIA OU MODA obteve erros *mse* no intervalo $[8.98 \pm 0.33, 9.39 \pm 0.12]$.

Para cada conjunto de dados, é apresentada uma tabela com os dados na forma numérica obtidos pelos métodos de tratamento de valores desconhecidos. Os resultados apresentam as taxas de erro médias obtidas por meio do método de reamostragem *10-fold cross-validation* e seus respectivos desvios padrão. Os resultados obtidos pelo método 10-NNI são comparados com os demais métodos por meio do teste-t pareado para *10-fold cross-validation* (Dietterich, 1997b). Os resultados comparativos são reportados nas colunas "Teste-t". Os símbolos "$\uparrow$", "$\Uparrow$", "$\downarrow$" e "$\Downarrow$" são utilizados para enfatizar os resultados nos quais o teste de significância identificou uma diferença estatisticamente significativa. As setas simples indicam diferenças com 95% de confiança e as setas duplas indicam diferenças com 99% de confiança. Por fim, as setas apontando para cima indicam que o método 10-NNI foi superior ao método concorrente, e as setas apontando para baixo indicam que o método 10-NNI foi inferior ao método concorrente.

Nas próximas seções são discutidos os resultados experimentais para os conjuntos de dados **Bupa**, **CMC**, **Pima**, **CRX**, **Breast** e **Sonar**, respectivamente.

5.7.3.1 O Conjunto de Dados Bupa

Considerando os resultados apresentados na Figura 5.8 na página 123 e na tabela correspondente na página 124 para o conjunto de dados **Bupa**, pode ser observado que o

desempenho do método 10-NNI é, na maior parte das vezes, superior ao desempenho dos algoritmos internos utilizados pelos indutores C4.5 e CN2. O método 10-NNI também é, na maioria das vezes, superior ao método IMPUTAÇÃO PELA MÉDIA OU MODA. Pode ser observado também que o algoritmo interno utilizado pelo método C4.5 obtém resultados próximos aos obtidos pelo método 10-NNI somente quando valores desconhecidos são inseridos nos três atributos selecionados. O método IMPUTAÇÃO PELA MÉDIA OU MODA obteve bons resultados somente quando valores desconhecidos foram inseridos nos três atributos selecionados, e o classificador foi induzido pelo sistema de aprendizado CN2.

5.7.3.2 Conjunto de Dados CMC

Resultados similares aos obtidos para o conjunto de dados **Bupa** podem ser encontrados na Figura 5.9 na página 125 e na tabela correspondente na página 126 para o conjunto de dados **CMC**. O desempenho do método 10-NNI é, na maioria das vezes, superior ao desempenho das estratégias utilizadas pelos sistemas C4.5 e CN2 para tratar valores desconhecidos. O desempenho do algoritmo 10-NNI também supera com freqüência o desempenho do método IMPUTAÇÃO PELA MÉDIA OU MODA. Mais precisamente, o método IMPUTAÇÃO PELA MÉDIA OU MODA somente obtém resultados próximos aos obtidos com o método 10-NNI quando os valores desconhecidos são inseridos em dois, ou três atributos selecionados, e o classificador foi induzido pelo sistema CN2.

5.7.3.3 Conjunto de Dados Pima

Na Figura 5.10 na página 127 e na tabela correspondente na página 128 são mostrados os resultados para o conjunto de dados **Pima**. O desempenho do método 10-NNI é, na maioria das vezes, superior ao desempenho obtido sem tratamento prévio de valores desconhecidos pelos indutores C4.5 e CN2. Além disso, o método 10-NNI é sempre superior ao método IMPUTAÇÃO PELA MÉDIA OU MODA e também ao não tratamento prévio de valores desconhecidos quando os valores desconhecidos foram inseridos no atributo 1 para ambos indutores, C4.5 e CN2.

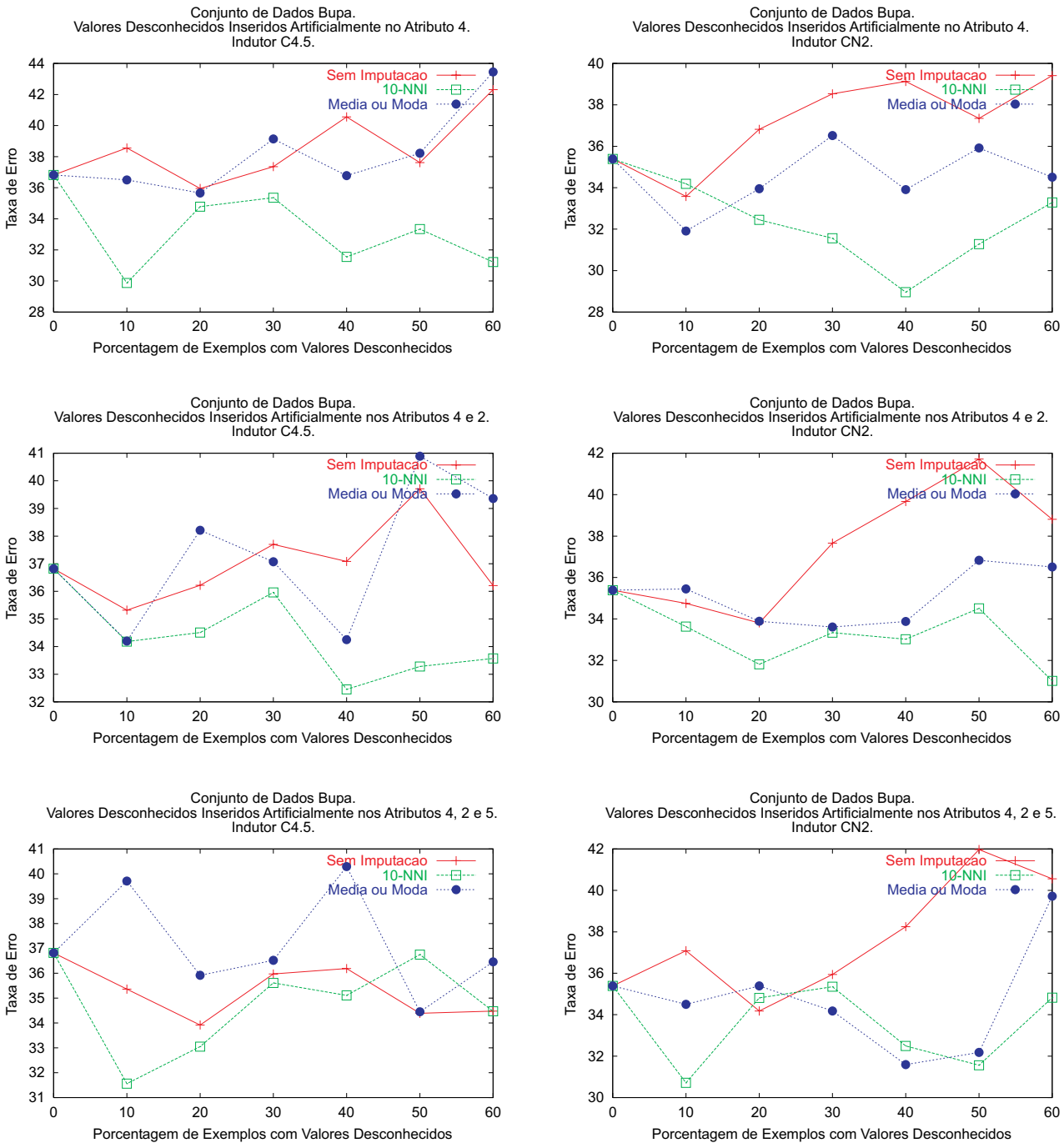

Figura 5.8: Comparação do método 10-NNI com as estratégias internas utilizada pelos indutores C4.5 e CN2 e com a IMPUTAÇÃO PELA MÉDIA OU MODA para o conjunto de dados **Bupa**. Na Tabela 5.3 são apresentados os resultados na forma numérica.

Indutor: C4.5						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	36,82 ± 2,69	-	-	-	-
	10	38,56 ± 1,74	↑ 3,01	29,87 ± 1,76	⇑ 3,45	36,50 ± 1,76
	20	35,95 ± 1,24	0,67	34,78 ± 2,43	0,32	35,66 ± 1,61
4	30	37,36 ± 1,89	0,72	35,36 ± 2,71	1,97	39,14 ± 2,41
	40	40,56 ± 2,05	↑ 3,09	31,55 ± 1,86	2,17	36,78 ± 1,72
	50	37,62 ± 2,35	1,16	33,34 ± 2,54	1,17	38,22 ± 3,03
	60	42,31 ± 2,11	↑ 2,62	31,22 ± 3,33	↑ 3,05	43,45 ± 2,08
	0	36,82 ± 2,69	-	-	-	-
	10	35,32 ± 2,36	0,52	34,18 ± 1,72	0,01	34,20 ± 2,23
	20	36,22 ± 2,18	0,57	34,51 ± 2,16	1,60	38,21 ± 2,54
4 e 2	30	37,70 ± 2,40	0,90	35,96 ± 2,05	0,53	37,07 ± 2,44
	40	37,08 ± 1,42	↑ 2,51	32,45 ± 1,09	0,84	34,25 ± 1,76
	50	39,71 ± 2,76	1,85	33,28 ± 3,07	1,84	40,89 ± 2,31
	60	36,21 ± 1,84	1,09	33,57 ± 2,38	2,13	39,36 ± 2,30
	0	36,82 ± 2,69	-	-	-	-
	10	35,36 ± 1,76	1,51	31,56 ± 2,44	↑ 3,03	39,71 ± 1,91
	20	33,92 ± 2,07	0,36	33,05 ± 2,09	1,53	35,92 ± 1,17
4, 2 e 5	30	35,97 ± 2,90	0,08	35,61 ± 3,00	0,41	36,52 ± 1,68
	40	36,19 ± 2,39	0,38	35,11 ± 2,14	1,30	40,29 ± 2,47
	50	34,39 ± 2,84	-0,97	36,75 ± 2,12	-1,16	34,45 ± 1,75
	60	34,48 ± 1,77	0,00	34,47 ± 3,02	0,60	36,46 ± 1,71
Indutor: CN2						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	35,39 ± 2,47	-	-	-	-
	10	33,58 ± 1,94	-0,22	34,19 ± 1,45	-1,05	31,91 ± 1,88
	20	36,82 ± 0,96	⇑ 3,51	32,45 ± 0,95	0,79	33,95 ± 1,70
4	30	38,53 ± 2,16	⇑ 4,02	31,56 ± 2,71	1,82	36,52 ± 1,74
	40	39,13 ± 1,09	⇑ 6,21	28,96 ± 2,24	↑ 2,56	33,91 ± 1,36
	50	37,35 ± 2,74	2,02	31,28 ± 1,91	1,69	35,92 ± 2,09
	60	39,41 ± 1,20	↑ 2,38	33,29 ± 2,64	0,48	34,51 ± 2,78
	0	35,39 ± 2,47	-	-	-	-
	10	34,75 ± 2,01	0,57	33,63 ± 1,77	0,73	35,45 ± 2,21
	20	33,81 ± 3,23	0,64	31,81 ± 2,65	1,08	33,89 ± 1,49
4 e 2	30	37,66 ± 1,48	↑ 2,65	33,34 ± 1,88	0,13	33,61 ± 1,96
	40	39,67 ± 1,98	2,25	33,02 ± 2,44	0,32	33,88 ± 1,27
	50	41,72 ± 1,38	↑ 3,04	34,51 ± 2,40	0,71	36,83 ± 1,88
	60	38,81 ± 1,58	⇑ 3,90	31,01 ± 1,48	⇑ 3,48	36,51 ± 2,33
	0	35,39 ± 2,47	-	-	-	-
	10	37,09 ± 2,55	↑ 2,45	30,71 ± 2,47	1,69	34,50 ± 1,81
	20	34,18 ± 2,03	-0,38	34,81 ± 1,49	0,30	35,39 ± 1,75
4, 2 e 5	30	35,94 ± 2,14	0,24	35,35 ± 1,39	-0,46	34,18 ± 1,92
	40	38,25 ± 1,49	↑ 3,09	32,49 ± 1,20	-0,31	31,59 ± 2,51
	50	41,97 ± 1,58	⇑ 5,34	31,56 ± 1,58	0,21	32,18 ± 2,24
	60	40,56 ± 1,88	2,05	34,82 ± 2,04	1,94	39,72 ± 1,63

Tabela 5.3: Resultados experimentais na forma numérica para o conjunto de dados **Bupa**.

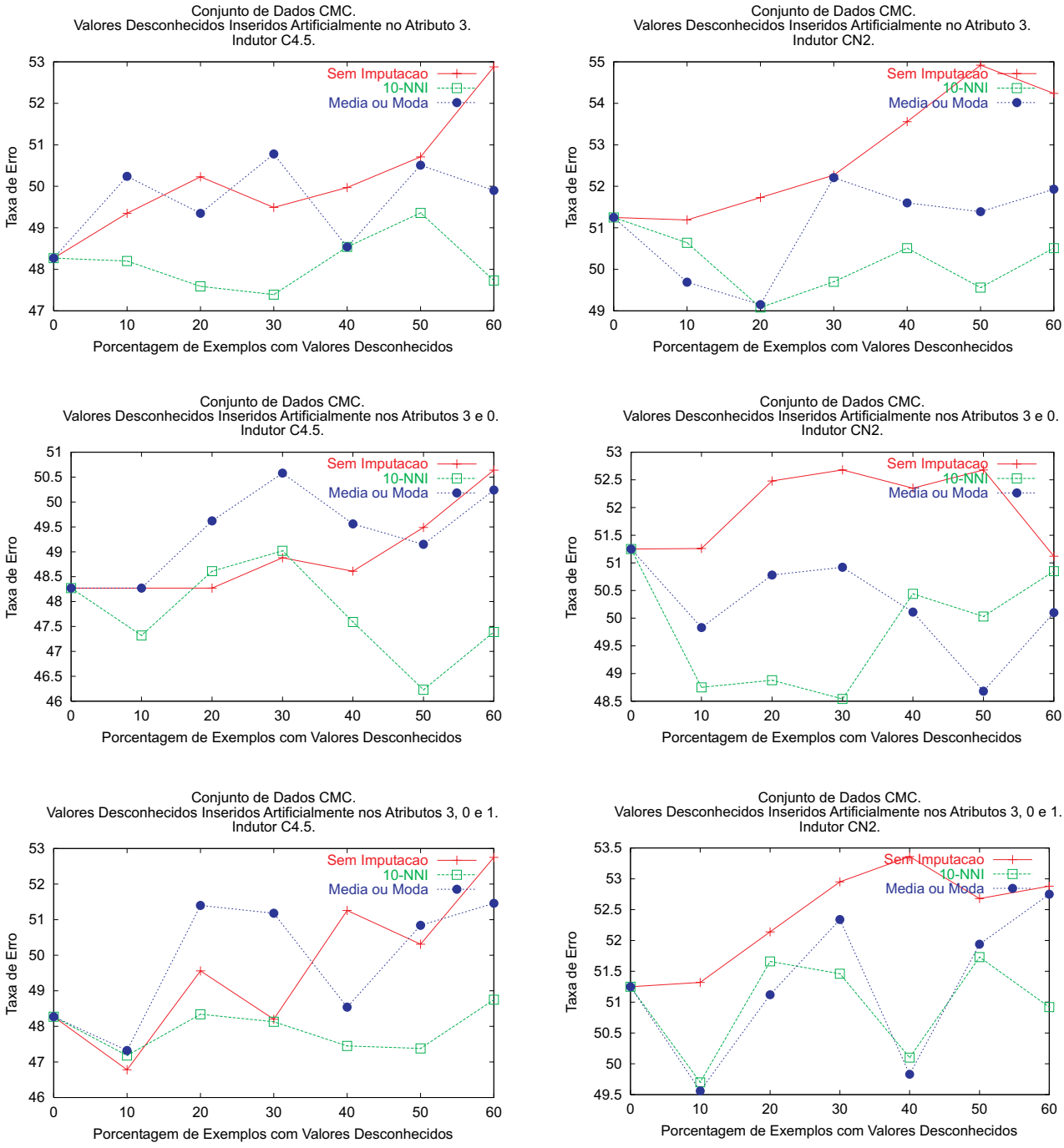

Figura 5.9: Comparação do método 10-NNI com as estratégias internas utilizada pelos indutores C4.5 e CN2 e com a IMPUTAÇÃO PELA MÉDIA OU MODA para o conjunto de dados **CMC**. Na Tabela 5.4 são apresentados os resultados na forma numérica.

Indutor: C4.5						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	48,27 ± 0,83	-	-	-	-
	10	49,35 ± 1,14	1,27	48,20 ± 1,16	1,59	50,24 ± 1,15
	20	50,23 ± 1,12	↑ 2,31	47,59 ± 0,98	1,62	49,35 ± 0,85
3	30	49,49 ± 0,95	2,16	47,39 ± 1,48	⇑ 3,66	50,78 ± 1,45
	40	49,97 ± 0,87	1,58	48,54 ± 1,12	0,00	48,54 ± 1,46
	50	50,71 ± 1,11	1,14	49,36 ± 0,91	0,87	50,51 ± 1,15
	60	52,88 ± 1,25	⇑ 7,08	47,73 ± 0,95	↑ 2,99	49,90 ± 1,07
	0	48,27 ± 0,83	-	-	-	-
	10	48,27 ± 0,67	1,10	47,32 ± 1,30	0,78	48,27 ± 1,37
	20	48,27 ± 0,99	-0,30	48,61 ± 1,30	0,56	49,62 ± 1,42
3 e 0	30	48,88 ± 1,40	-0,09	49,02 ± 1,36	1,33	50,58 ± 0,98
	40	48,61 ± 1,20	0,76	47,59 ± 1,53	0,91	49,56 ± 1,33
	50	49,49 ± 0,84	↑ 3,16	46,23 ± 1,06	↑ 2,34	49,15 ± 1,38
	60	50,64 ± 1,16	1,62	47,39 ± 1,87	1,48	50,24 ± 0,91
	0	48,27 ± 0,83	-	-	-	-
	10	46,78 ± 1,46	-0,29	47,18 ± 1,19	0,13	47,32 ± 0,78
	20	49,56 ± 1,34	1,75	48,34 ± 1,29	↑ 2,76	51,40 ± 1,49
3, 0 e 1	30	48,20 ± 1,19	0,05	48,13 ± 1,51	1,87	51,18 ± 0,89
	40	51,26 ± 1,33	↑ 3,18	47,45 ± 1,46	0,81	48,54 ± 1,12
	50	50,31 ± 1,23	↑ 2,52	47,38 ± 1,74	2,06	50,84 ± 1,61
	60	52,75 ± 1,16	⇑ 3,49	48,75 ± 1,86	↑ 2,53	51,46 ± 1,06
Indutor: CN2						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	51,25 ± 0,80	-	-	-	-
	10	51,19 ± 1,51	0,48	50,64 ± 1,22	-0,99	49,69 ± 1,34
	20	51,73 ± 1,17	↑ 2,61	49,08 ± 0,95	0,05	49,15 ± 1,42
3	30	52,27 ± 0,94	1,96	49,70 ± 1,71	1,85	52,21 ± 1,13
	40	53,56 ± 1,47	↑ 2,28	50,51 ± 1,11	1,36	51,60 ± 0,73
	50	54,92 ± 0,95	⇑ 4,18	49,56 ± 1,74	1,46	51,39 ± 1,39
	60	54,24 ± 1,31	↑ 2,63	50,51 ± 1,12	1,13	51,93 ± 1,50
	0	51,25 ± 0,80	-	-	-	-
	10	51,26 ± 0,80	2,16	48,75 ± 1,42	0,76	49,83 ± 0,77
	20	52,48 ± 1,51	↑ 2,84	48,88 ± 1,46	1,82	50,78 ± 1,20
3 e 0	30	52,68 ± 0,91	↑ 2,77	48,54 ± 1,34	2,25	50,92 ± 0,95
	40	52,35 ± 1,10	↑ 2,88	50,44 ± 1,09	-0,28	50,11 ± 1,43
	50	52,68 ± 0,81	1,60	50,03 ± 1,76	-0,99	48,68 ± 1,18
	60	51,12 ± 1,53	0,28	50,85 ± 1,49	-0,69	50,10 ± 1,38
	0	51,25 ± 0,80	-	-	-	-
	10	51,32 ± 1,19	1,13	49,70 ± 1,61	-0,10	49,56 ± 1,46
	20	52,14 ± 1,04	0,39	51,66 ± 1,06	-0,53	51,12 ± 1,07
3, 0 e 1	30	52,95 ± 1,25	1,55	51,46 ± 1,15	0,60	52,34 ± 1,45
	40	53,36 ± 1,23	↑ 2,91	50,10 ± 1,49	-0,34	49,83 ± 0,85
	50	52,68 ± 1,02	0,55	51,73 ± 1,82	0,11	51,94 ± 1,29
	60	52,88 ± 0,76	1,32	50,92 ± 1,35	1,50	52,75 ± 1,05

Tabela 5.4: Resultados experimentais na forma numérica para o conjunto de dados **CMC**.

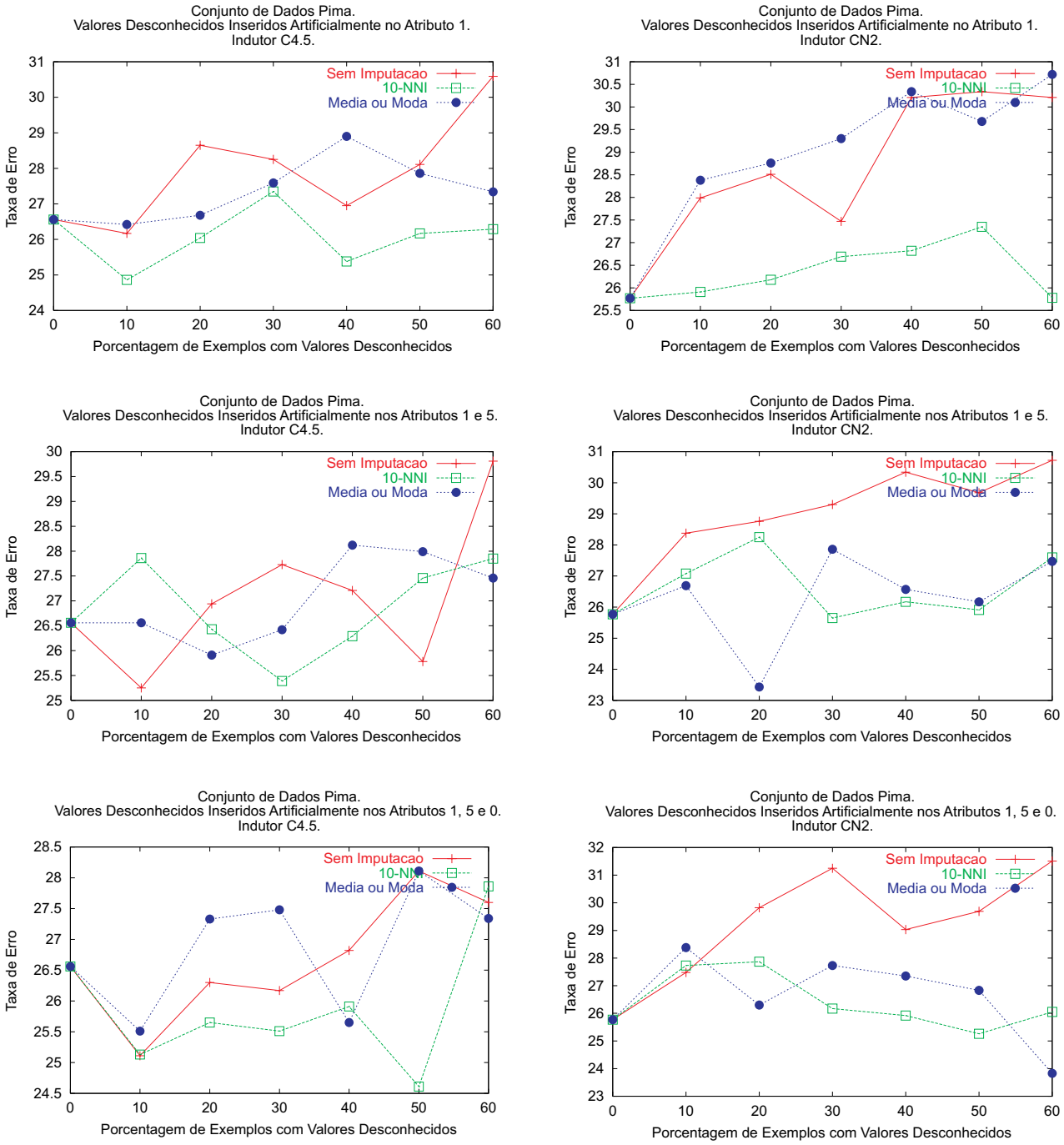

Figura 5.10: Comparação do método 10-NNI com as estratégias internas utilizada pelos indutores C4.5 e CN2 e com a IMPUTAÇÃO PELA MÉDIA OU MODA para o conjunto de dados **Pima**. Na Tabela 5.5 são apresentados os resultados na forma numérica.

Indutor: C4.5						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	26,56 ± 1,16	-	-	-	-
	10	26,17 ± 1,03	1,06	24,86 ± 0,88	1,01	26,42 ± 1,48
	20	28,65 ± 1,15	1,48	26,04 ± 1,68	0,43	26,68 ± 1,18
1	30	28,25 ± 1,85	0,46	27,35 ± 1,03	0,18	27,59 ± 1,38
	40	26,95 ± 1,67	0,93	25,38 ± 1,15	↑ 2,56	28,90 ± 1,23
	50	28,11 ± 1,14	1,10	26,17 ± 1,11	1,39	27,86 ± 0,84
	60	30,59 ± 1,13	2,21	26,29 ± 1,90	0,56	27,34 ± 1,05
	0	26,56 ± 1,16	-	-	-	-
	10	25,25 ± 1,10	-2,00	27,86 ± 1,15	-0,85	26,56 ± 1,08
	20	26,94 ± 1,22	0,39	26,43 ± 1,08	-0,49	25,91 ± 1,34
1 e 5	30	27,73 ± 1,60	1,26	25,39 ± 0,81	0,62	26,42 ± 1,27
	40	27,21 ± 1,45	0,51	26,29 ± 1,69	0,89	28,12 ± 1,11
	50	25,78 ± 1,13	-1,39	27,46 ± 1,16	0,36	27,99 ± 1,37
	60	29,81 ± 1,43	1,18	27,85 ± 1,51	-0,18	27,46 ± 1,67
	0	26,56 ± 1,16	-	-	-	-
	10	25,11 ± 1,70	-0,01	25,13 ± 0,90	0,18	25,51 ± 1,90
	20	26,30 ± 1,01	0,66	25,65 ± 1,35	0,88	27,33 ± 1,42
1, 5 e 0	30	26,17 ± 1,35	0,38	25,51 ± 1,75	1,17	27,48 ± 1,19
	40	26,82 ± 1,28	0,67	25,91 ± 1,44	-0,21	25,65 ± 0,84
	50	28,11 ± 1,32	⇑ 3,41	24,61 ± 1,16	2,15	28,11 ± 1,65
	60	27,60 ± 1,05	-0,19	27,86 ± 1,55	-0,41	27,34 ± 1,53
Indutor: CN2						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	25,77 ± 1,12	-	-	-	-
	10	27,99 ± 0,98	2,23	25,91 ± 0,86	1,82	28,38 ± 0,87
	20	28,51 ± 1,06	1,80	26,18 ± 0,78	1,66	28,76 ± 1,51
1	30	27,47 ± 1,11	0,41	26,69 ± 1,61	1,14	29,30 ± 1,23
	40	30,21 ± 1,08	2,00	26,82 ± 0,98	↑ 3,15	30,34 ± 1,59
	50	30,34 ± 1,21	1,54	27,35 ± 1,47	1,23	29,68 ± 1,58
	60	30,21 ± 1,28	2,21	25,78 ± 1,33	↑ 2,55	30,72 ± 1,47
	0	25,77 ± 1,12	-	-	-	-
	10	28,38 ± 0,87	1,32	27,08 ± 0,98	-0,30	26,69 ± 1,31
	20	28,76 ± 1,51	0,32	28,25 ± 1,09	⇓ -4,97	23,43 ± 0,68
1 e 5	30	29,30 ± 1,23	↑ 2,45	25,65 ± 1,13	1,81	27,86 ± 1,16
	40	30,34 ± 1,59	↑ 2,56	26,17 ± 1,07	0,27	26,57 ± 1,73
	50	29,68 ± 1,58	2,02	25,91 ± 1,08	0,23	26,17 ± 0,82
	60	30,72 ± 1,47	1,18	27,60 ± 1,47	-0,07	27,47 ± 0,75
	0	25,77 ± 1,12	-	-	-	-
	10	27,48 ± 1,00	-0,21	27,73 ± 0,68	0,79	28,38 ± 0,99
	20	29,82 ± 0,82	1,48	27,87 ± 1,26	-1,31	26,30 ± 1,13
1, 5 e 0	30	31,25 ± 0,89	⇑ 3,55	26,17 ± 1,32	1,97	27,73 ± 0,91
	40	29,03 ± 0,90	⇑ 3,67	25,92 ± 1,32	1,77	27,35 ± 0,92
	50	29,69 ± 0,41	⇑ 7,98	25,26 ± 0,68	1,31	26,83 ± 1,29
	60	31,51 ± 1,17	↑ 3,05	26,05 ± 0,86	-1,83	23,83 ± 0,95

Tabela 5.5: Resultados experimentais na forma numérica para o conjunto de dados **Pima**.

5.7.3.4 Conjunto de Dados CRX

Na Figura 5.11 na próxima página e na tabela correspondente na página 131 são mostrados os resultados para o conjunto de dados **CRX**. O desempenho do método 10-NNI é superior ao desempenho dos demais métodos de tratamento de valores desconhecidos. A única ocasião em que o método 10-NNI obteve uma taxa de erro superior aos demais métodos foi quando os valores desconhecidos foram inseridos no atributo 9 com uma taxa de 10% e o classificador foi induzido pelo sistema C4.5. Em todos os demais casos o método 10-NNI obteve taxas de erro inferiores aos demais métodos de tratamento de valores desconhecidos. Os métodos IMPUTAÇÃO PELA MÉDIA OU MODA e as estratégias internas utilizadas pelos indutores C4.5 e CN2 obtiveram resultados similares.

5.7.3.5 Conjunto de Dados Breast

Embora a imputação com K-VIZINHOS MAIS PRÓXIMOS pode prover bons resultados, existem ocasiões em que seu uso deve ser evitado. Uma dessas situações pode ser ilustrada pelo conjunto de dados **Breast**. Esse conjunto de dados possui fortes correlações entre seus atributos. As correlações causam uma situação interessante: por um lado, o algoritmo K-VIZINHOS MAIS PRÓXIMOS pode predizer os valores desconhecidos com uma precisão bem superior à IMPUTAÇÃO PELA MÉDIA OU MODA; por outro lado, o indutor pode decidir não utilizar o atributo tratado, substituindo esse atributo por outro com alta correlação. Os resultados obtidos com o conjunto de dados **Breast** são mostrados na Figura 5.12 na página 132 e na tabela correspondente na página 133, nas quais pode ser visto que o método 10-NNI não supera os demais métodos de tratamento de valores desconhecidos.

Como explicado anteriormente, o cenário causado pelo conjunto de dados **Breast** é interessante pois o método 10-NNI foi capaz de predizer os valores desconhecidos com uma precisão superior a IMPUTAÇÃO PELA MÉDIA OU MODA. Como os valores desconhecidos foram inseridos artificialmente nos dados, o erro médio quadrático (mse) entre os valores preditos e os valores reais pode ser medido. Esses erros são apresentados na Tabela 5.8 na página 134 para os três atributos selecionados como mais relevantes no conjunto de dados.

Se o método 10-NNI é mais preciso em predizer os valores desconhecidos, então porque essa precisão não é traduzida em um classificador mais preciso? A resposta pode estar na alta correlação ente os atributos no conjunto de dados, e porque (ou conseqüentemente) o conjunto de dados **Breast** possui diversos atributos com poder de precisão similar.

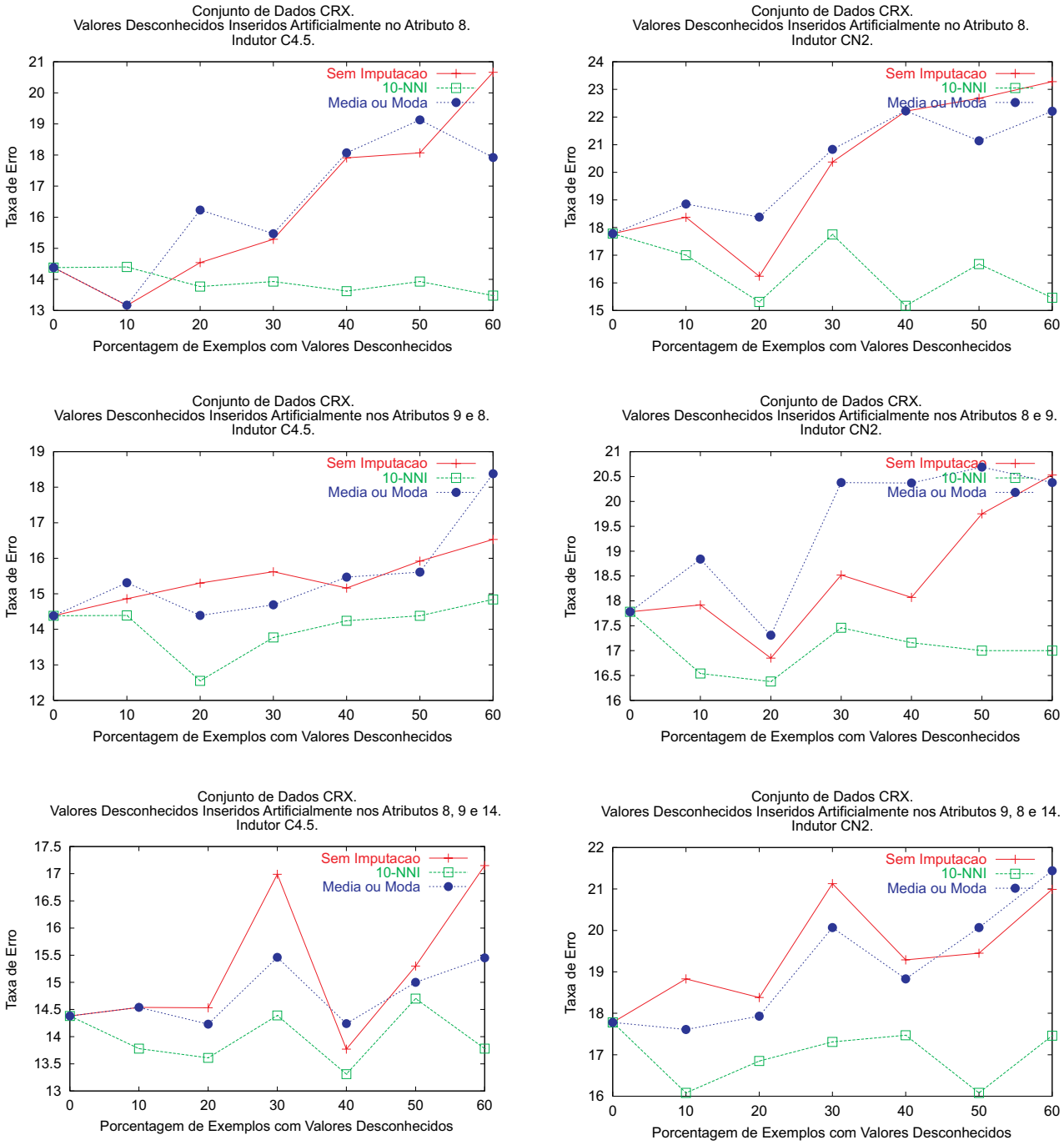

Figura 5.11: Comparação do método 10-NNI com as estratégias internas utilizada pelos indutores C4.5 e CN2 e com a IMPUTAÇÃO PELA MÉDIA OU MODA para o conjunto de dados **CRX**. Na Tabela 5.6 são apresentados os resultados na forma numérica.

Indutor: C4.5						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
8	0	14,38 ± 1,57	-	-	-	-
	10	13,16 ± 1,65	-2,24	14,40 ± 1,88	↓ -2,75	13,17 ± 1,77
	20	14,54 ± 1,61	0,41	13,77 ± 1,37	1,95	16,23 ± 1,23
	30	15,29 ± 1,68	1,17	13,93 ± 1,27	1,02	15,47 ± 1,75
	40	17,91 ± 1,71	↑ 3,09	13,62 ± 1,30	↑ 3,18	18,07 ± 1,93
	50	18,07 ± 2,13	↑ 2,88	13,93 ± 1,14	↑ 2,75	19,13 ± 2,22
	60	20,66 ± 1,64	⇑ 6,74	13,48 ± 1,04	↑ 2,79	17,92 ± 2,27
8 e 9	0	14,38 ± 1,57	-	-	-	-
	10	14,86 ± 1,70	0,35	14,39 ± 1,22	1,21	15,31 ± 0,96
	20	15,30 ± 1,53	2,18	12,55 ± 1,52	1,42	14,39 ± 1,54
	30	15,62 ± 1,08	1,92	13,77 ± 1,79	0,94	14,69 ± 1,78
	40	15,16 ± 1,84	0,70	14,24 ± 1,37	1,35	15,47 ± 1,46
	50	15,92 ± 1,26	1,58	14,38 ± 0,92	1,18	15,61 ± 1,35
	60	16,53 ± 1,91	1,30	14,84 ± 1,37	2,15	18,38 ± 1,58
8, 9 e 14	0	14,38 ± 1,57	-	-	-	-
	10	14,54 ± 1,45	1,10	13,78 ± 1,31	0,91	14,54 ± 1,19
	20	14,53 ± 1,37	0,97	13,61 ± 1,53	0,85	14,23 ± 1,25
	30	16,99 ± 1,65	↑ 2,27	14,39 ± 1,49	1,21	15,46 ± 1,46
	40	13,77 ± 1,50	0,70	13,31 ± 1,26	0,90	14,24 ± 1,41
	50	15,30 ± 1,23	0,53	14,70 ± 1,54	0,51	15,00 ± 1,94
	60	17,15 ± 1,54	⇑ 3,59	13,78 ± 1,30	1,67	15,45 ± 1,71
Indutor: CN2						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
8	0	17,78 ± 1,60	-	-	-	-
	10	18,37 ± 1,41	1,09	17,00 ± 1,94	1,14	18,85 ± 1,57
	20	16,24 ± 1,26	0,95	15,31 ± 1,24	↑ 2,37	18,38 ± 2,07
	30	20,37 ± 1,96	1,39	17,75 ± 1,90	1,78	20,83 ± 1,47
	40	22,21 ± 2,28	⇑ 3,46	15,17 ± 1,72	⇑ 3,97	22,22 ± 1,62
	50	22,68 ± 2,16	2,06	16,68 ± 1,86	2,04	21,14 ± 1,56
	60	23,28 ± 2,16	⇑ 3,34	15,46 ± 1,04	⇑ 3,39	22,21 ± 1,94
8 e 9	0	17,78 ± 1,60	-	-	-	-
	10	17,92 ± 1,83	0,92	16,54 ± 1,10	2,08	18,84 ± 1,41
	20	16,85 ± 1,69	0,30	16,38 ± 2,20	0,77	17,31 ± 2,00
	30	18,52 ± 1,92	0,99	17,46 ± 1,93	1,76	20,38 ± 2,11
	40	18,07 ± 1,97	0,79	17,16 ± 1,46	↑ 2,30	20,37 ± 1,72
	50	19,75 ± 1,71	↑ 2,70	17,00 ± 1,80	↑ 2,58	20,69 ± 1,83
	60	20,53 ± 2,52	1,92	17,00 ± 1,87	↑ 2,53	20,38 ± 1,50
8, 9 e 14	0	17,78 ± 1,60	-	-	-	-
	10	18,83 ± 1,53	↑ 2,33	16,08 ± 1,25	1,46	17,61 ± 1,70
	20	18,38 ± 1,76	0,97	16,85 ± 1,22	1,10	17,93 ± 1,69
	30	21,13 ± 1,36	⇑ 3,33	17,31 ± 1,43	1,58	20,07 ± 1,44
	40	19,29 ± 0,97	1,10	17,47 ± 1,64	0,87	18,83 ± 1,58
	50	19,45 ± 1,62	1,92	16,08 ± 2,20	1,61	20,07 ± 1,90
	60	20,99 ± 1,64	↑ 2,29	17,46 ± 1,18	2,26	21,44 ± 1,52

Tabela 5.6: Resultados experimentais na forma numérica para o conjunto de dados **CRX**.

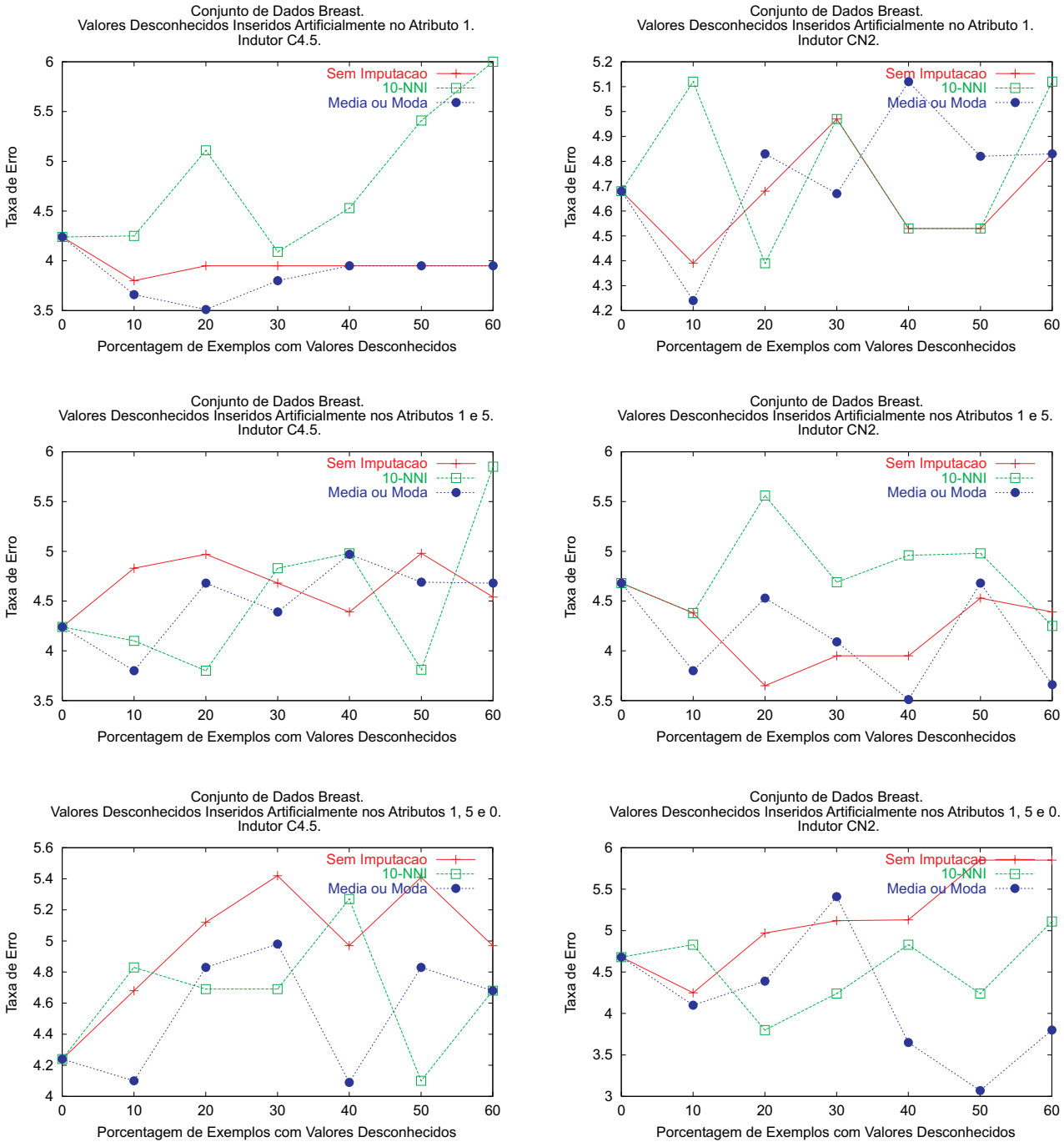

Figura 5.12: Comparação do método 10-NNI com as estratégias internas utilizada pelos indutores C4.5 e CN2 e com a IMPUTAÇÃO PELA MÉDIA OU MODA para o conjunto de dados **Breast**. Na Tabela 5.7 são apresentados os resultados na forma numérica.

Indutor: C4.5						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	4,24 ± 0,67	-	-	-	-
	10	3,80 ± 0,93	-0,61	4,25 ± 0,67	-0,94	3,66 ± 0,82
	20	3,95 ± 0,90	-1,56	5,11 ± 0,99	-2,01	3,51 ± 0,88
1	30	3,95 ± 0,90	-0,24	4,09 ± 0,91	-0,47	3,80 ± 0,93
	40	3,95 ± 0,90	-0,73	4,53 ± 0,82	-0,73	3,95 ± 0,90
	50	3,95 ± 0,90	-1,79	5,41 ± 1,00	-1,79	3,95 ± 0,90
	60	3,95 ± 0,90	↓ -2,94	6,00 ± 0,88	↓ -2,94	3,95 ± 0,90
	0	4,24 ± 0,67	-	-	-	-
	10	4,83 ± 0,61	0,99	4,10 ± 0,61	-0,44	3,80 ± 0,85
	20	4,97 ± 0,65	1,49	3,80 ± 0,88	0,97	4,68 ± 0,64
1 e 5	30	4,68 ± 0,61	-0,21	4,83 ± 0,69	-1,40	4,39 ± 0,65
	40	4,39 ± 0,65	-0,77	4,98 ± 0,54	-0,01	4,97 ± 0,44
	50	4,98 ± 0,73	1,44	3,81 ± 0,63	1,50	4,69 ± 0,37
	60	4,54 ± 0,71	-1,96	5,85 ± 0,53	-1,92	4,68 ± 0,65
	0	4,24 ± 0,67	-	-	-	-
	10	4,68 ± 0,75	-0,19	4,83 ± 0,81	-1,34	4,10 ± 0,61
	20	5,12 ± 0,73	0,40	4,69 ± 0,68	0,14	4,83 ± 0,69
1, 5 e 0	30	5,42 ± 0,69	0,63	4,69 ± 1,02	0,28	4,98 ± 0,50
	40	4,97 ± 0,62	-0,32	5,27 ± 0,85	-1,40	4,09 ± 0,68
	50	5,41 ± 0,57	1,17	4,10 ± 0,84	0,68	4,83 ± 0,61
	60	4,97 ± 0,73	0,34	4,68 ± 0,80	0,00	4,68 ± 0,78
Indutor: CN2						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
	0	4,68 ± 0,60	-	-	-	-
	10	4,39 ± 0,44	-1,10	5,12 ± 0,84	-1,03	4,24 ± 0,46
	20	4,68 ± 0,75	0,32	4,39 ± 0,57	0,67	4,83 ± 0,69
1	30	4,97 ± 0,82	0,00	4,97 ± 0,62	-0,39	4,67 ± 1,03
	40	4,53 ± 0,73	0,00	4,53 ± 0,70	1,82	5,12 ± 0,90
	50	4,53 ± 0,91	0,00	4,53 ± 0,63	0,43	4,82 ± 0,87
	60	4,83 ± 0,84	-0,33	5,12 ± 0,69	-0,36	4,83 ± 1,07
	0	4,68 ± 0,60	-	-	-	-
	10	4,38 ± 0,65	0,00	4,38 ± 0,75	-0,87	3,80 ± 0,66
	20	3,65 ± 0,84	⇓ -3,55	5,56 ± 0,77	-1,91	4,53 ± 0,67
1 e 5	30	3,95 ± 0,54	-1,87	4,69 ± 0,57	-0,70	4,09 ± 0,97
	40	3,95 ± 0,87	-1,29	4,96 ± 0,99	-2,12	3,51 ± 0,66
	50	4,53 ± 0,63	-0,56	4,98 ± 0,76	-0,50	4,68 ± 0,75
	60	4,39 ± 0,95	0,16	4,25 ± 0,64	-0,85	3,66 ± 0,66
	0	4,68 ± 0,60	-	-	-	-
	10	4,25 ± 0,71	-0,88	4,83 ± 0,76	-1,05	4,10 ± 0,52
	20	4,97 ± 0,79	2,05	3,80 ± 0,62	1,32	4,39 ± 0,66
1, 5 e 0	30	5,12 ± 0,54	1,41	4,24 ± 0,80	1,44	5,41 ± 0,78
	40	5,13 ± 0,55	0,32	4,83 ± 0,62	-1,45	3,65 ± 0,82
	50	5,85 ± 0,76	1,82	4,24 ± 0,91	-1,57	3,07 ± 0,82
	60	5,85 ± 0,61	0,87	5,11 ± 0,97	-1,78	3,80 ± 0,73

Tabela 5.7: Resultados experimentais na forma numérica para o conjunto de dados **Breast**.

Atributo	*mse* 10-NNI	*mse* Média ou Moda
0 (*Clump Thickness*)	$4,02 \pm 0,14$	$7,70 \pm 0,28$
1 (*Uniformity of Cell Size*)	$1,72 \pm 0,11$	$8,96 \pm 0,36$
5 (*Bare Nuclei*)	$4,23 \pm 0,30$	$13,29 \pm 0,46$

Tabela 5.8: Erro médio quadrático (*mse*) entre os valores preditos e os valores reais para os métodos 10-NNI e imputação pela média ou moda — conjunto de dados **Breast**.

Para realizar uma análise mais profunda, é necessário verificar como cada atributo é utilizado pelo classificador induzido. Por exemplo, é interessante entender como o sistema C4.5 foi capaz de obter uma taxa de erro constante mesmo com uma grande quantidade de valores desconhecidos inseridos no atributo 1. Analisando as árvores de decisão geradas pelo indutor C4.5, é possível verificar que o C4.5 foi capaz de substituir o atributo 1 — *Uniformity of Cell Size* — pelo atributo 2 — *Uniformity of Cell Shape*. Essa substituição foi possível pois esses dois atributos possuem uma alta correlação (coeficiente de correlação linear $r = 0.9072$). De uma forma geral, para o conjunto de dados **Breast**, o indutor C4.5 foi capaz de trocar todos os atributos com valores desconhecidos por outros atributos, e ainda assim obter resultados similares ou melhores que os obtidos pelo método 10-NNI.

Utilizando o nível mais alto da árvore de decisão em que o atributo foi incorporado como uma medida heurística da importância do atributo no classificador, na Tabela 5.9 na página oposta é mostrado que o indutor C4.5 foi capaz de descartar gradualmente os atributos com valores desconhecidos conforme a quantidade de valores desconhecidos aumentava. De forma similar, o indutor C4.5 mostra uma tendência de descartar os atributos com valores desconhecidos quando esses atributos são tratados pelo método imputação pela média ou moda. Esse resultado é esperado uma vez que no método imputação pela média ou moda todos os valores desconhecidos são substituídos por um mesmo valor, ou seja, a média ou moda do atributo. Conseqüentemente, o poder de discriminação do atributo, medido por diversos indutores por meio da *entropia* ou de outras medidas similares, tende a decrescer. O mesmo não ocorre quando os valores desconhecidos são tratados pelo método 10-NNI. Quando o método 10-NNI é utilizado, o indutor C4.5 mantém os atributos com valores desconhecidos como os atributos mais próximos da raiz da árvore de decisão. Essa situação poderia ter sido uma vantagem se o conjunto de dados **Breast** não possuísse outros atributos com poder de predição similar aos atributos selecionados.

% Desconhecidos	Sem Imputação			Média ou Moda			10-NNI		
	Atrib. 1	Atrib. 5	Atrib. 0	Atrib. 1	Atrib. 5	Atrib. 0	Atrib. 1	Atrib. 5	Atrib. 0
0%	1	2	3	1	2	3	1	2	3
10%	2	2	3	2	2	3	1	2	3
20%	-	2	3	-	3	3	1	2	3
30%	-	5	-	-	3	-	1	2	3
40%	5	4	-	3	-	-	1	2	3
50%	-	-	-	6	7	3	1	3	2
60%	-	5	-	-	3	-	1	2	3

Tabela 5.9: Nível da árvore de decisão no qual os atributos 1, 5 e 0 do conjunto de dados **Breast** foram incorporados pelo indutor C4.5. "-" significa que o atributo não foi incorporado à árvore de decisão. Nível 1 representa a raiz da árvore.

5.7.3.6 Conjunto de Dados Sonar

Para confirmar os resultados obtidos com o conjunto de dados **Breast**, foi incluído nos experimentos o conjunto de dados **Sonar**. O conjunto de dados **Sonar** possui características similares ao conjunto de dados **Breast**, uma vez que seus atributos possuem fortes relações entre si. Uma outra característica interessante do conjunto de dados **Sonar** é que esse conjunto de dados possui uma grande quantidade de atributos, 60 no total. Essa grande quantidade de atributos pode fornecer ao indutor diversas possibilidades durante a escolha dos atributos que irão compor o classificador. Na Tabela 5.10 são mostrados os índices de correlação linear entre os atributos selecionados e os atributos de maior correlação linear presentes no conjunto de dados.

Atributo selecionado	Atributo de maior correlação	Índice de correlação linear r
A10	A9	$0,8531$
A0	A1	$0,7359$
A26	A25	$0,8572$

Tabela 5.10: Índice de correlação linear r entre os atributos selecionados como mais representativos e os atributos de maior correlação — conjunto de dados **Sonar**.

Na Figura 5.13 na próxima página e na tabela correspondente na página 137 são mostrados os resultados para o conjunto de dados **Sonar**. Da mesma forma que nos resultados obtidos para o conjunto de dados **Breast**, o sistema C4.5 foi capaz de substituir os atributos com valores desconhecidos por outros atributos com informações similares. Induzindo o mesmo classificador, o sistema C4.5 foi capaz de apresentar a mesma taxa de erro, mesmo para grandes quantidades de valores desconhecidos. Diferentemente do conjunto de dados **Breast**, o método 10-NNI foi capaz de superar o sistema C4.5 em duas situações: quando os valores desconhecidos foram inseridos no atributo 10 e nos atributos 10, 0 e 26.

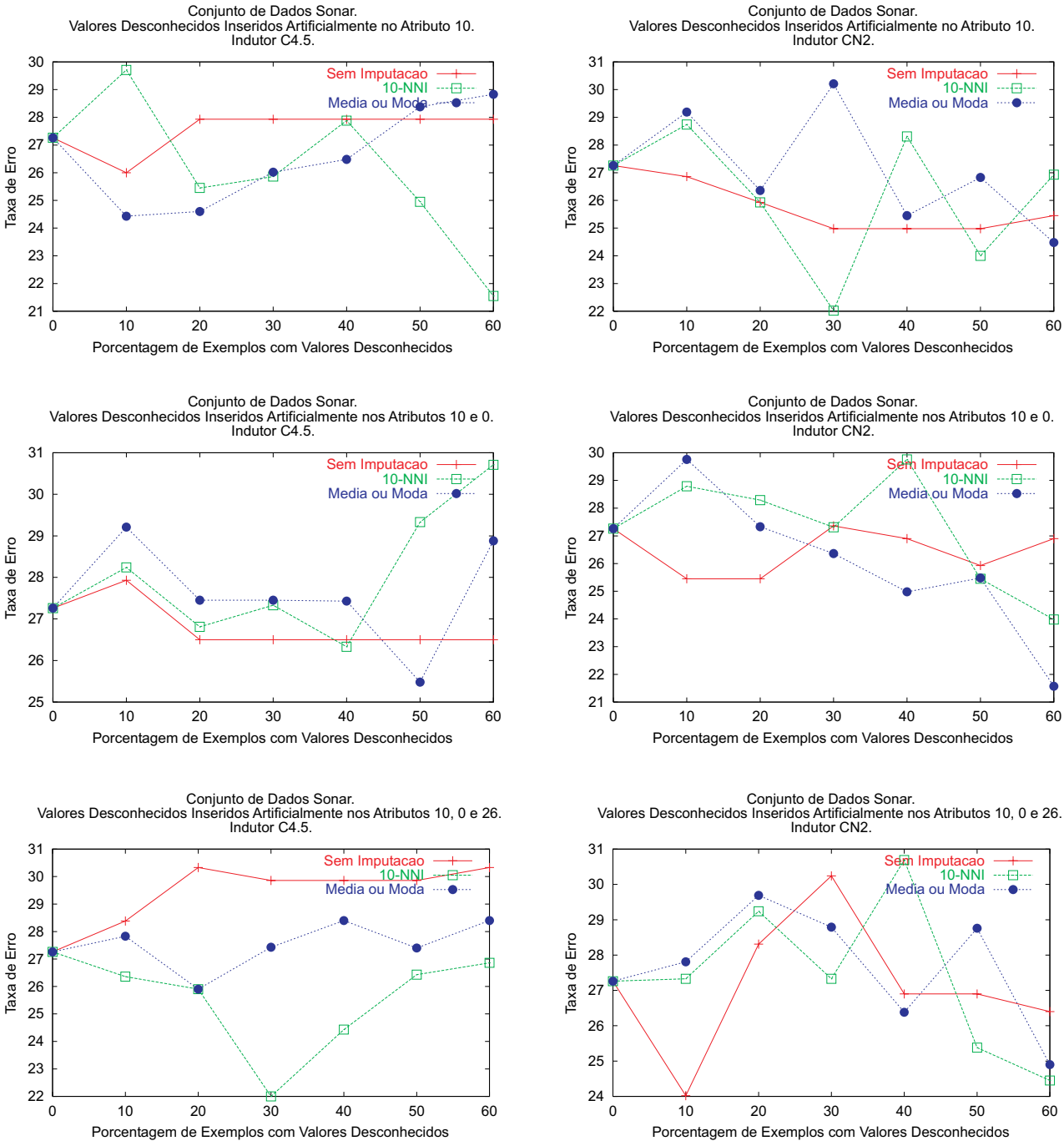

Figura 5.13: Comparação do método 10-NNI com as estratégias internas utilizada pelos indutores C4.5 e CN2 e com a IMPUTAÇÃO PELA MÉDIA OU MODA para o conjunto de dados **Sonar**. Na Tabela 5.11 são apresentados os resultados na forma numérica.

Indutor: C4.5						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
10	0	27,26 ± 3,64	-	-	-	-
	10	26,00 ± 3,23	-0,77	29,71 ± 2,75	-1,48	24,43 ± 3,17
	20	27,93 ± 2,96	0,57	25,45 ± 3,04	-0,20	24,60 ± 2,88
	30	27,93 ± 2,96	0,47	25,86 ± 3,30	0,04	26,02 ± 1,88
	40	27,93 ± 2,96	0,01	27,88 ± 3,09	-0,37	26,48 ± 3,07
	50	27,93 ± 2,96	0,73	24,95 ± 3,39	0,89	28,38 ± 2,89
	60	27,93 ± 2,96	1,61	21,55 ± 2,93	1,87	28,83 ± 3,16
10 e 0	0	27,26 ± 3,64	-	-	-	-
	10	27,93 ± 2,96	-0,06	28,24 ± 3,66	0,27	29,21 ± 4,50
	20	26,50 ± 3,25	-0,07	26,81 ± 2,89	0,16	27,45 ± 2,43
	30	26,50 ± 3,25	-0,16	27,33 ± 3,78	0,03	27,45 ± 2,43
	40	26,50 ± 3,25	0,03	26,33 ± 3,99	0,21	27,43 ± 3,12
	50	26,50 ± 3,25	-0,69	29,33 ± 4,68	-0,81	25,48 ± 3,55
	60	26,50 ± 3,25	-0,90	30,71 ± 2,75	-0,34	28,88 ± 3,58
10, 0 e 26	0	27,26 ± 3,64	-	-	-	-
	10	28,38 ± 3,37	0,56	26,36 ± 3,54	0,59	27,83 ± 3,95
	20	30,33 ± 3,46	1,61	25,90 ± 2,91	0,00	25,90 ± 4,08
	30	29,86 ± 3,59	1,79	22,00 ± 3,94	1,63	27,43 ± 2,96
	40	29,86 ± 3,52	1,48	24,43 ± 3,50	1,01	28,40 ± 3,96
	50	29,86 ± 3,52	0,91	26,43 ± 2,94	0,32	27,40 ± 3,69
	60	30,33 ± 3,53	0,88	26,86 ± 4,08	0,41	28,40 ± 3,09
Indutor: CN2						
Attributos	% ?	Sem Imputação	Teste-t	10-NNI	Teste-t	Média ou Moda
10	0	27,26 ± 3,64	-	-	-	-
	10	26,86 ± 3,23	-0,79	28,74 ± 2,87	0,30	29,19 ± 3,75
	20	25,93 ± 2,34	0,00	25,93 ± 3,54	0,24	26,36 ± 4,47
	30	24,98 ± 1,95	0,86	22,02 ± 4,31	↑ 2,45	30,21 ± 3,28
	40	24,98 ± 1,95	-1,55	28,31 ± 3,83	-1,20	25,45 ± 3,71
	50	24,98 ± 1,95	0,35	24,00 ± 3,83	0,99	26,83 ± 2,81
	60	25,45 ± 2,10	-0,51	26,93 ± 3,47	-1,06	24,48 ± 2,94
10 e 0	0	27,26 ± 3,64	-	-	-	-
	10	25,45 ± 2,33	-1,47	28,79 ± 2,98	1,50	29,76 ± 2,98
	20	25,45 ± 2,43	-1,39	28,29 ± 3,94	-1,00	27,33 ± 4,11
	30	27,36 ± 2,20	0,02	27,31 ± 3,72	-0,51	26,36 ± 3,68
	40	26,90 ± 2,24	-1,03	29,76 ± 3,85	-1,95	24,98 ± 3,73
	50	25,93 ± 1,72	0,21	25,45 ± 3,55	0,01	25,48 ± 3,01
	60	26,90 ± 2,12	0,91	23,98 ± 3,87	-0,97	21,57 ± 2,65
10, 0 e 26	0	27,26 ± 3,64	-	-	-	-
	10	24,02 ± 2,73	-1,63	27,33 ± 3,14	0,36	27,81 ± 3,27
	20	28,31 ± 2,65	-0,32	29,24 ± 4,24	0,18	29,69 ± 3,90
	30	30,24 ± 1,80	1,12	27,33 ± 3,72	1,16	28,79 ± 3,43
	40	26,90 ± 2,24	-1,29	30,69 ± 3,73	-1,50	26,38 ± 3,48
	50	26,90 ± 0,69	0,37	25,38 ± 4,48	0,89	28,76 ± 2,86
	60	26,40 ± 0,97	0,53	24,45 ± 4,01	0,22	24,90 ± 3,27

Tabela 5.11: Resultados experimentais na forma numérica para o conjunto de dados **Sonar**.

Para o indutor CN2, não é possível dizer que um dos métodos foi superior aos demais. Os resultados apresentam alta variância, e freqüentemente um método apresenta a taxa de erro mais baixa para um determinado nível de valores desconhecidos, e logo em seguida apresenta a taxa de erro mais alta para o nível de valores desconhecidos imediatamente maior.

5.8 Considerações Finais

Neste capítulo é analisado o comportamento de quatro métodos de tratamento de valores desconhecidos: o método 10-NNI, o qual utiliza o algoritmo K-VIZINHOS MAIS PRÓXIMOS para imputação de valores desconhecidos; a IMPUTAÇÃO PELA MÉDIA OU MODA; e os algoritmos internos utilizados pelos sistemas de aprendizado C4.5 e CN2 para tratar valores desconhecidos.

Esses métodos foram analisados em diversos conjuntos de dados com diferentes porcentagens de valores desconhecidos inseridos em diferentes atributos, utilizando o ambiente DLE para implementar os métodos de imputação, gerenciar os experimentos e criar as tabelas e gráficos correspondentes. Os resultados obtidos são promissores. Para os conjuntos de dados **Bupa**, **CMC**, **Pima** e **CRX**, o método 10-NNI provê muito bons resultados, mesmo para conjuntos de treinamento com grandes quantidades de valores desconhecidos.

De fato, para os conjuntos de dados **Bupa**, **CMC**, **Pima** e **CRX**, os métodos internos utilizados pelos indutores C4.5 e CN2 para tratar valores desconhecidos apresentam taxas de erro inferior comparadas com as obtidas pelo método 10-NNI em apenas 13 de 144 medidas (10 para o C4.5 e 3 para o CN2), como pode ser visto nas Tabelas 5.3 na página 124, 5.4 na página 126, 5.5 na página 128 e 5.6 na página 131. Em somente duas dessas 13 medidas os métodos internos obtiveram diferenças estatisticamente significantes. Por outro lado, o método 10-NNI obteve diferenças significantes em 31 medidas, sendo que em 10 dessas 31 medidas as diferenças são altamente significantes.

Os conjuntos de dados **Breast** e **Sonar** provêem uma análise importante sobre as limitações dos métodos de tratamento de valores desconhecidos. A existência de outros atributos com informações similares, isto é, com alta correlação, ou poder de predição similar, podem fazer a imputação de valores desconhecidos pouco efetiva, ou até mesmo danosa.

A imputação de valores desconhecidos pode ser danosa uma vez que mesmo o mais

avançado método de imputação somente é capaz de aproximar os valores reais, não conhecidos, dos valores desconhecidos. Os valores preditos são geralmente mais bem comportados, uma vez que eles são preditos em conformidade com os valores dos outros atributos. Nos experimentos conduzidos, quanto mais atributos com valores desconhecidos são introduzidos, e quanto maior é a quantidade de valores desconhecidos, mas simples são os classificadores induzidos. Dessa forma, a imputação de valores desconhecidos deve ser cuidadosamente aplicada, sob o risco de simplificar demasiadamente o problema em estudo.

Capítulo 6

Aprendizado com Classes Desbalanceadas

6.1 Considerações Iniciais

Conforme os métodos propostos pelos pesquisadores da área de Aprendizado de Máquina são aplicados em problemas "reais", novas questões, algumas delas nunca previamente consideradas pela comunidade de AM, têm sido levantadas. Uma dessas questões diz respeito ao problema de *classes desbalanceadas*. O problema de classes desbalanceadas corresponde a domínios nos quais uma classe é representada por um grande número de exemplos, enquanto que a outra é representada por poucos exemplos[1].

O problema de classes desbalanceadas é de grande importância uma vez que conjuntos de dados com essa característica podem ser encontrados em diversos domínios. Por exemplo, em detecção de fraudes em chamadas telefônicas (Fawcett & Provost, 1997), e em transações de cartões de crédito (Stolfo, Fan, Lee, Prodromidis & Chan, 1997), o número de transações legítimas é muito maior que o número de transações fraudulentas. Em análise de risco para seguradoras (Pednault, Rosen & Apte, 2000), somente uma pequena porcentagem dos clientes acionam o seguro em um dado período de tempo. Em *marketing* direto (Ling & Li, 1998), é comum obter somente um pequena taxa de resposta, em torno de 1%, para a maior parte das campanhas. Outros exemplos de domínios com um grande desbalanço intrínseco entre as classes podem ser encontrados na literatura.

[1]Neste capítulo é analisado o aprendizado com duas classes, as quais podem ser consideradas como exemplos e contra-exemplos de um conceito.

Muitos sistemas de aprendizado assumem que as classes estão balanceadas e, dessa forma, esses sistemas falham em induzir um classificador que seja capaz de predizer a classe minoritária com precisão na presença de dados com classes desbalanceadas. Freqüentemente, o classificador possui uma boa precisão para a classe majoritária, mas uma precisão inaceitável para a classe minoritária. O problema agrava-se ainda mais quando o custo de classificação incorreta da classe minoritária é muito maior do que o custo de classificação incorreta da classe majoritária. No mundo real, essa é a norma para a maior parte das aplicações com conjuntos de dados com classes desbalanceadas, uma vez que essas aplicações visam traçar o perfil de um pequeno conjunto de entidades valiosas que estão dispersas em um grande grupo de entidades "pouco interessantes".

Neste capítulo são discutidos alguns dos métodos de pré-processamento de dados mais utilizados para solucionar o problema de aprender com conjuntos de dados com classes desbalanceadas. Um desses métodos, a *seleção unilateral*, é pesquisado com maior profundidade e analisado experimentalmente.

Este capítulo está organizado da seguinte forma: na Seção 6.2 são descritos alguns dos métodos mais utilizados para solucionar o problema de classes desbalanceadas; na Seção 6.3 são discutidos os motivos pelos quais duas das medidas mais utilizadas para medir o desempenho de sistemas de aprendizado, a precisão e a taxa de erro, serem inadequadas para medir o desempenho quando os dados possuem custos assimétricos de classificação incorreta e/ou quando as classes são desbalanceadas; na Seção 6.4 é explicada a relação entre desbalanço de classes e aprendizado sensível ao custo; na Seção 6.5 é iniciada uma discussão sobre qual distribuição de classes é melhor para o aprendizado; na Seção 6.6 são resumidos alguns métodos de pré-processamento de dados propostos pela comunidade de AM para balancear a distribuição das classes, bem como é apresentado em maiores detalhes o método de *seleção unilateral*; na Seção 6.7 são apresentadas algumas evidências que mostram que em muitos sistemas de aprendizado o balanceamento da distribuição das classes tem pouco efeito sobre o classificador final; na Seção 6.8 são descritos alguns dos experimentos realizados com o objetivo de avaliar o método de seleção unilateral; por fim, na Seção 6.9 são apresentadas as considerações finais deste capítulo.

6.2 Métodos para Solucionar o Problema de Classes Desbalanceadas

Vários pesquisadores têm analisado o problema de aprender a partir de conjuntos de dados com classes desbalanceadas (Pazzani, Merz, Murphy, Ali, Hume & Brunk, 1994; Ling & Li, 1998; Kubat & Matwin, 1997; Fawcett & Provost, 1997; Kubat, Holte & Matwin, 1998; Japkowicz & Stephen, 2002). Dentre os diversos métodos propostos por esses pesquisadores, três abordagens principais têm sido utilizadas com maior freqüência, são elas:

Atribuição de custos de classificação incorreta
: Para muitos domínios de aplicação, classificar incorretamente exemplos da classe minoritária é mais custoso do que classificar incorretamente exemplos da classe majoritária. Para esses domínios é possível utilizar sistemas de aprendizado sensíveis ao custo de classificação. Esses sistemas objetivam minimizar o custo total ao invés da taxa de erro de classificação. A principal restrição ao uso desses sistemas é que o custo de classificação incorreta de cada classe deve ser precisamente conhecido e deve ser um valor constante;

Under-sampling
: Uma forma bastante direta de solucionar o problema de classes desbalanceadas é balancear artificialmente a distribuição das classes no conjunto de exemplos. Os métodos de *under-sampling* visam balancear o conjunto de dados por meio da eliminação de exemplos da classe majoritária;

Over-sampling
: Os métodos de *over-sampling* são similares aos métodos de *under-sampling*. Entretanto, esses métodos visam balancear a distribuição das classes por meio da replicação de exemplos da classe minoritária.

Das três abordagens descritas acima, os métodos de *under-* e *over-sampling* são métodos de pré-processamento de dados, ou seja, esses métodos podem ser aplicados em uma fase anterior à fase de extração de conhecimento.

6.3 Precisão, Taxa de Erro e Classes Desbalanceadas

Os diferentes tipos de erros e acertos realizados por um classificador podem ser sintetizados em uma *matriz de confusão*. Na Tabela 6.1 é mostrada uma matriz de confusão para um problema que possui duas classes rotuladas como classe *positiva* e classe *negativa*.

	Predição Positiva	*Predição Negativa*
Classe Positiva	Verdadeiro Positivo (a)	Falso Negativo (b)
Classe Negativa	Falso Positivo (c)	Verdadeiro Negativo (d)

Tabela 6.1: Diferentes tipos de erros e acertos para um problema com duas classes.

A partir dessa matriz de confusão é possível extrair diversas medidas de desempenho para sistemas de aprendizado, tal como a *taxa de erro*, Err — Equação 6.1 — e a *precisão*, Acc — Equação 6.2.

$$Err = \frac{c + b}{a + b + c + d} \tag{6.1}$$

$$Acc = \frac{a + d}{a + b + c + d} \tag{6.2}$$

A taxa de erro e a precisão são duas medidas amplamente utilizadas para medir o desempenho de sistemas de aprendizado. Entretanto, quando a probabilidade a priori de cada classe é muito diferente, isto é, quando existe um grande desbalanço entre as classes, tais medidas podem ser enganosas. Por exemplo, é bastante simples criar um classificador com 99% de precisão, ou de forma similar, com 1% de taxa de erro, se o conjunto de dados possui uma classe majoritária com 99% do número total de exemplos. Esse classificador pode ser criado simplesmente rotulando todo novo caso como pertencente a classe majoritária.

Um outro argumento contra o uso da precisão, ou taxa de erro, é que essas medidas consideram erros de classificação diferentes como igualmente importantes. Por exemplo, um paciente doente diagnosticado como sadio pode ser um erro fatal, enquanto que um paciente sadio diagnosticado como doente pode ser considerado um erro menos sério, uma vez que esse erro pode ser corrigido em exames futuros. Em domínios nos quais o custo de classificação incorreta é relevante, uma matriz de custo pode ser utilizada. Uma matriz de custo define os custos de classificação incorreta, isto é, uma penalidade para cada tipo de erro que o classificador pode cometer. Nesse caso, o objetivo do classificador deve ser minimizar o custo total de classificação incorreta, ao invés da taxa de erro. Na

Seção 6.4 na página 148 é realizada uma discussão mais ampla sobre a relação entre aprendizado sensível ao custo e conjuntos de dados com classes desbalanceadas.

Para conjuntos de dados com classes desbalanceadas, uma medida de desempenho mais apropriada deve desassociar os erros, ou acertos, ocorridos para cada classe. A partir da Tabela 6.1 na página anterior é possível derivar quatro medidas de desempenho que medem o desempenho de classificação nas classes negativa e positiva independentemente, elas são:

Taxa de falso negativo
: $FN = \frac{b}{a+b}$ é a porcentagem de casos positivos classificados incorretamente como pertencentes à classe negativa;

Taxa de falso positivo
: $FP = \frac{c}{c+d}$ é a porcentagem de casos negativos classificados incorretamente como pertencentes à classe positiva;

Taxa de verdadeiro negativo
: $VN = \frac{d}{c+d} = 1 - FP$ é a porcentagem de casos negativos classificados corretamente como pertencentes à classe negativa;

Taxa de verdadeiro positivo
: $VP = \frac{a}{a+b} = 1 - FN$ é a porcentagem de casos positivos classificados corretamente como pertencentes à classe positiva.

Essas quatro medidas de desempenho possuem a vantagem de serem independentes do custo e das probabilidades a priori das classes. O principal objetivo de qualquer classificador é minimizar as taxas de falso positivo e de falso negativo, ou, de forma similar, maximizar as taxas de verdadeiro positivo e verdadeiro negativo. Entretanto, para a maioria das aplicações do "mundo real", existe uma relação de perda e ganho entre FN e FP, ou, de forma similar, entre VN e VP.

Na Figura 6.1 na página seguinte é mostrada uma relação comum entre as taxas de falso positivo e falso negativo. Esse gráfico foi criado para uma aplicação que visa identificar transações fraudulentas em cartões de crédito. Chan & Stolfo (1998a) treinaram o sistema de aprendizado C4.5 com diferentes distribuições do atributo classe no conjunto de treinamento. Os conjuntos de teste foram mantidos intactos, isto é, com a distribuição das classes que ocorre naturalmente nos dados. O gráfico na Figura 6.1 inicia com um conjunto de treinamento consistindo de 90% de casos pertencentes à classe majoritária. A proporção de casos da classe minoritária é aumentado em 10% a cada iteração. Esse

aumento no número de casos da classe minoritária no conjunto de treinamento leva a uma melhora no desempenho de classificação dos casos dessa classe. Entretanto, a precisão na classificação da classe majoritária diminui. A taxa de erro no conjunto de teste aumenta influenciada pelo desempenho ruim obtido pela classe majoritária, uma vez que a maior parte dos casos de teste pertencem a essa classe.

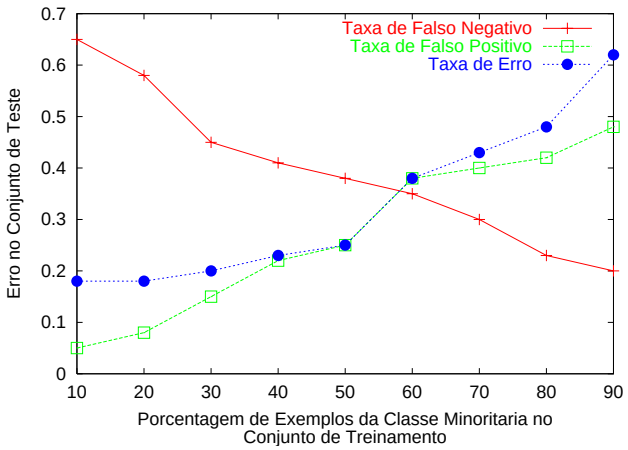

Figura 6.1: Erro no conjunto de teste para diversas distribuições de classes no conjunto de treinamento.

Um gráfico ROC[2] (Provost & Fawcett, 1997) pode ser utilizado para analisar a relação entre FN e FP, ou VN e VP, para um determinado classificador.

Considere que a classe minoritária, cujo desempenho é o principal foco da análise, é a classe positiva. Em um gráfico ROC, $VP = 1 - FN$ é associado ao eixo Y e FP é associado ao eixo X. Alguns classificadores possuem parâmetros para os quais diferentes ajustes podem produzir pontos em um gráfico ROC. Por exemplo, para um classificador que responde com a probabilidade de um exemplo pertencer a uma determinada classe, tal como o classificador Naive Bayes, pode ser utilizado um parâmetro que define um limiar. Um exemplo somente pode ser classificado como pertencente a uma determinada classe se a probabilidade fornecida pelo classificador for superior ao limiar. De forma similar, outros sistemas de aprendizado podem ser adaptados para produzir probabilidades a posteriori similares ao Naive Bayes. Em árvores de decisão, a distribuição das classes nos nós folhas pode ser utilizada como uma estimativa de probabilidade. Sistemas que induzem regras podem fazer estimativas similares. Redes neurais produzem saídas contínuas que podem também ser mapeadas para estimativas de probabilidade.

[2]ROC é uma sigla para *Receiver Operating Characteristic*, um termo utilizado em detecção de sinais para caracterizar a relação de perda e ganho entre a taxa de acerto e a taxa de falso alarme em um canal com ruído.

O desenho de todos os pontos que podem ser produzidos por meio da variação dos parâmetros do classificador produz uma *curva ROC* para o classificador. Na prática, essa curva é um conjunto discreto de pontos, incluindo os pontos (0,0) e (1,1), os quais são conectados por segmentos de reta. Na Figura 6.2 é ilustrado um gráfico ROC com três classificadores representados pelas letras A, B e C. Alguns dos pontos em um gráfico ROC devem ser notados. O ponto no canto inferior esquerdo (0,0) representa a estratégia de classificar todos os exemplos como pertencentes à classe negativa. O ponto no canto superior direito representa a estratégia de classificar todos os exemplos como pertencentes à classe positiva. O ponto (0,1) representa o classificador perfeito, e a linha $x = y$ representa a estratégia de tentar adivinhar a classe aleatoriamente.

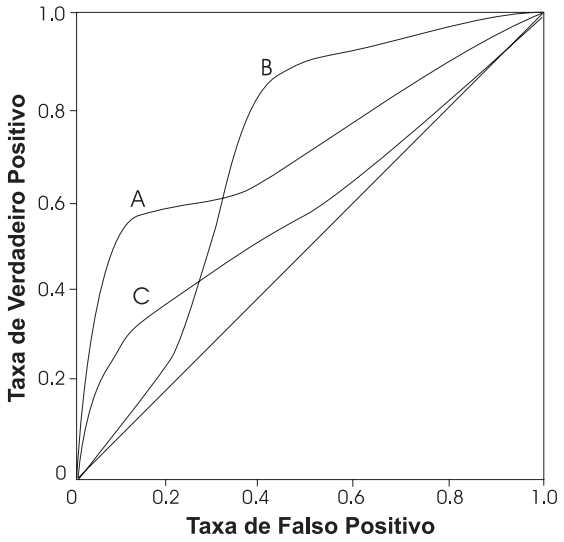

Figura 6.2: Um exemplo de gráfico ROC para três classificadores.

A partir de um gráfico ROC é possível calcular uma medida geral de qualidade, a *área sob a curva (AUC*[3]*)*. A AUC é a fração da área total que se situa sob a curva ROC. Essa medida é equivalente a diversas outras medidas estatísticas para a avaliação de modelos de classificação (Hand, 1997). A medida AUC efetivamente fatora o desempenho do classificador sobre todos os custos e distribuições.

[3] *Area under the ROC curve.*

6.4 Conjuntos Desbalanceados e Aprendizado Sensível ao Custo

Como mencionado anteriormente, um classificador induzido a partir de um conjunto de dados com classes desbalanceadas possui, tipicamente, uma taxa de erro baixa para a classe majoritária, e uma taxa de erro demasiadamente alta para a classe minoritária. O problema surge quando o custo de classificação incorreta da classe minoritária é muito superior ao custo de classificação incorreta da classe majoritária. Nessa situação, o desafio está em classificar precisamente a classe minoritária, com o objetivo de reduzir o custo total do classificador.

Um sistema de aprendizado sensível ao custo pode ser utilizado em aplicações nas quais os custos de classificação incorreta são conhecidos. Sistemas de aprendizado sensíveis ao custo visam reduzir o custo de classificação incorreta dos exemplos, ao invés do erro de classificação.

Alguns sistemas de aprendizado não são capazes de integrar informações de custo em seu processo de aprendizado. Entretanto, existe um método simples e geral para tornar qualquer sistema de aprendizado sensível ao custo para um problema de duas classes (Breiman, Friedman, Olshen & Stone, 1984). Esse método baseia-se em modificar a distribuição das classes no conjunto de treinamento de forma a aumentar o número de exemplos da classe mais custosa. Suponha que a classe positiva é cinco vezes mais custosa que a classe negativa. Se o número de exemplos positivos é artificialmente aumentado por um fator de cinco, então o sistema de aprendizado, visando reduzir o erro de classificação, irá induzir um classificador que tende evitar cometer erros na classe positiva, uma vez que qualquer erro nessa classe é penalizado cinco vezes mais.

Elkan (2001) demonstra um teorema que permite encontrar a proporção de exemplos positivos e negativos de forma a fazer classificações sensíveis ao custo ótimas para um problema de duas classes. Ainda, Domingos (1999) apresenta um método geral para fazer qualquer sistema de aprendizado sensível ao custo. Esse método possui a vantagem de ser aplicável a problemas que possuem qualquer número de classes.

Dessa forma, classes desbalanceadas e aprendizado sensível ao custo estão relacionados entre si. Uma forma de aprender com conjuntos com classes desbalanceadas é treinar um sistema sensível ao custo, com o custo da classe minoritária maior do que o da classe majoritária. Uma outra forma de fazer com que um sistema de aprendizado se torne sensível ao custo é alterar intencionalmente a distribuição das classes no conjunto

de treinamento.

Uma grande parte dos métodos que tratam conjuntos de dados com classes desbalanceadas visam melhorar o desempenho da classe minoritária por meio do balanceamento de classes do conjunto de dados. Dessa forma, a classe minoritária se torna mais custosa, e pode-se esperar que ela será melhor classificada. Desse ponto de vista, dois cenários podem ser identificados. No primeiro, existe uma grande quantidade de dados, e o problema pode ser entendido como

> *Qual proporção de exemplos positivos e negativos é a melhor para o aprendizado?*

No segundo cenário, os dados são escassos e, por isso, existe um problema adicional

> *Como descartar exemplos negativos ou duplicar exemplos positivos sem introduzir distorções no processo de aprendizado?*

Nas próximas seções, essas duas questões são discutidas mais detalhadamente.

6.5 Qual Proporção de Classes é Melhor para Aprender?

O problema de determinar qual proporção de exemplos positivos e negativos é a melhor para o aprendizado vai além do problema de aprender com conjuntos de dados com classes desbalanceadas. A comunidade de AM tem assumido implicitamente que as distribuições das classes que ocorrem naturalmente são as melhores para o aprendizado. Entretanto, uma vez que o aprendizado com as distribuições naturais das classes têm fornecido resultados ruins para conjuntos de dados com classes desbalanceadas, essa suposição começou a ser estuda mais a fundo.

A principio, a proporção de classes que fornece o melhor resultado no aprendizado varia entre os diferentes conjuntos de dados. Devido à complexidade dos dados e dos algoritmos que analisam esses dados, não é possível dizer previamente qual distribuição de classes irá fornecer o melhor resultado. Entretanto, é possível estabelecer algumas diretrizes gerais.

Weiss & Provost (2001) realizaram um estudo sobre a influência da proporção de exemplos positivos e negativos no aprendizado. Esse estudo analisa o cenário no qual existe uma grande quantidade de dados, entretanto, por motivos de restrição computacional, o conjunto de treinamento deve ser limitado a um certo número de exemplos. Nesse cenário, qual distribuição de classes deve ser a melhor para o aprendizado?

Utilizando a área sob a curva ROC como medida de desempenho, Weiss & Provost (2001) mostram que a distribuição ótima das classes geralmente contém entre 50% e 90% dos exemplos da classe minoritária. Também, alocar 50% dos exemplos de treinamento para a classe minoritária, mesmo quando não fornece resultados ótimos, apresenta resultados que não são piores, e que freqüentemente são superiores, aos resultados obtidos com a distribuição natural das classes.

6.6 Como Descartar ou Duplicar Exemplos?

Como mencionado previamente, uma das formas mais diretas de lidar com classes desbalanceadas por meio de métodos de pré-processamento de dados é alterar a distribuição dessas classes de forma a tornar o conjunto de dados mais balanceado. Dois métodos básicos de pré-processamento de dados para balancear a distribuição das classes são o método de *under-sampling* e o método de *over-sampling*.

Ambos os métodos, *under-sampling* e *over-sampling*, possuem limitações conhecidas. *Under-sampling* pode eliminar dados potencialmente úteis, e *over-sampling* pode aumentar a probabilidade de ocorrer *overfitting*, uma vez que a maioria dos métodos de *over-sampling* fazem cópias exatas dos exemplos pertencentes à classe minoritária. Dessa forma, um classificador simbólico, por exemplo, pode construir regras que são aparentemente precisas, mas que na verdade cobrem um único exemplo replicado.

Alguns trabalhos recentes têm tentado superar as limitações existentes tanto nos métodos de *under-sampling*, quanto nos métodos de *over-sampling*. Por exemplo, Chawla, Bowyer, Hall & Kegelmeyer (2002) combinam métodos de *under* e *over-sampling*. Nesse trabalho, o método de *over-sampling* não replica os exemplos da classe minoritária, mas cria novos exemplos dessa classe por meio da interpolação de diversos exemplos da classe minoritária que se encontram próximos. Dessa forma, é possível evitar o problema de *overfitting*.

Kubat & Matwin (1997) propôs um método de *under-sampling*, o qual foi posteriormente analisado em (Batista, Carvalho & Monard, 2000), chamado *seleção unilateral*, no

qual o principal objetivo é tentar minimizar a quantidade de dados potencialmente úteis descartados. Para isso, os exemplos são divididos em quatro categorias:

Ruído Ruído são casos que por algum motivo foram rotulados incorretamente, isto é, eles estão do lado errado da borda de decisão. Por exemplo, são os casos da classe majoritária, representados pelo símbolo "−", localizados na região esquerda da Figura 6.3;

Redundantes Casos redundantes são casos que podem ser representados por outros casos que estão presentes no conjunto de treinamento. Por exemplo, os casos que estão muito distantes da borda de decisão, como os casos da classe majoritária localizados na região extrema direita da Figura 6.3;

Próximos da borda Esses são ao casos que estão próximos da borda de decisão. Alguns desses casos são pouco confiáveis, uma vez que uma pequena quantidade de ruído em um dos atributos pode mover esses exemplos para o lado errado da borda de decisão;

Seguros Os casos seguros são aqueles que não são ruído, não estão excessivamente próximos à borda de decisão e, também, não estão muito distantes dela. Os casos seguros são, a princípio, os melhores casos a serem retidos para o aprendizado.

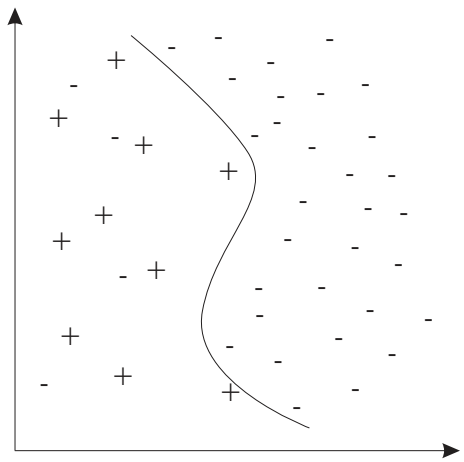

Figura 6.3: Exemplo de conjunto de dados com duas classes desbalanceadas.

Seleção Unilateral visa criar um conjunto de treinamento consistindo somente de casos seguros. Para isso, é necessário remover casos que são ruído, redundantes e próximos da borda de decisão. Entretanto, como os casos da classe minoritária são freqüentemente

muito raros para serem eliminados, a seleção unilateral somente remove casos da classe majoritária. Portanto, seleção unilateral é um método de *under-sampling*.

Casos próximos à borda e ruído podem ser identificados por meio das ligações Tomek, e removidos do conjunto de dados. Dessa forma, somente casos seguros são mantidos para o treinamento do sistema de aprendizado. Uma ligação Tomek pode ser definida da seguinte forma:

Definição 6.1 *Sejam E_i e E_j dois exemplos de classes diferentes, ou seja $f(E_i) \neq f(E_j)$. Seja d uma função de distância entre exemplos. Um par de exemplos (E_i, E_j) constitui uma ligação Tomek se não existe um exemplo E_k, tal que a distância $d(E_k, E_i) < d(E_i, E_j)$ ou $d(E_k, E_j) < d(E_i, E_j)$.*

Se dois exemplos (E_i, E_j) formam uma ligação Tomek, então ou E_i e E_j são exemplos próximos à borda de decisão, ou um desses exemplos é ruído. Na Figura 6.4 é ilustrado o processo de limpeza de um conjunto de dados por meio de ligações Tomek, no qual somente exemplos da classe majoritária são removidos.

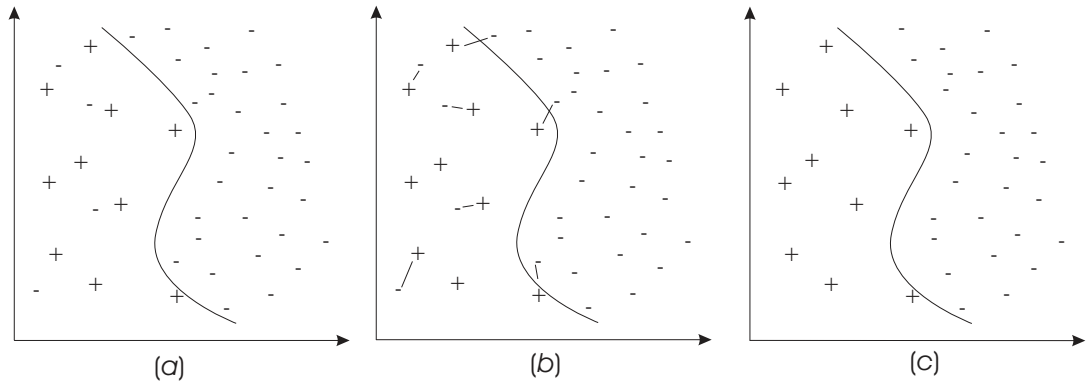

Figura 6.4: A aplicação de ligações Tomek em um conjunto de dados. O conjunto de dados original *(a)*, Ligações Tomek identificadas *(b)*, e ligações Tomek removidas *(c)*.

Parte dos casos redundantes pode ser removida por meio da identificação de um *subconjunto consistente*. Subconjuntos consistentes com um conjunto de exemplos E podem ser definidos da seguinte forma:

Definição 6.2 *Um subconjunto consistente $S \subset E$ é consistente com E se utilizando o algoritmo* 1-VIZINHO MAIS PRÓXIMO *S classifica corretamente os casos em E (Hart, 1968).*

Algoritmo 6.1 Algoritmo para encontrar um subconjunto consistente.

Require: $E = \{E_1, E_2, \ldots E_N\}$, um conjunto de exemplos e contra-exemplos do conceito a ser aprendido
1-NN(Tr,E_i), o algoritmo 1-VIZINHO MAIS PRÓXIMO que classifica o exemplo E_i utilizando o conjunto de treinamento Tr

Ensure: S é um subconjunto consiste de E

1: $S = \emptyset$
2: **for all** $E_i \in E$, tal que $f(E_i)$ é a classe minoritária **do**
3: $S = S \bigcup \{E_i\}$
4: **end for**
5: Seja $E_i \in E$ um exemplo escolhido aleatoriamente, tal que $f(E_i)$ é a classe majoritária
6: $S = S \bigcup \{E_i\}$
7: **for all** $E_i \in E$, tal que $f(E_i)$ é a classe majoritária **do**
8: **if** a classificação dada por 1-NN$(S, E_i) \neq f(E_i)$ **then**
9: $S = S \bigcup \{E_i\}$
10: **end if**
11: **end for**
12: **return** S

Um algoritmo para encontrar um subconjunto consistente é descrito no Algoritmo 6.1.

O procedimento apresentado no Algoritmo 6.1 não encontra necessariamente o menor subconjunto consistente de E. Na Figura 6.5 é mostrado o conjunto de dados apresentado na Figura 6.3 na página 151 após a criação de um subconjunto consistente.

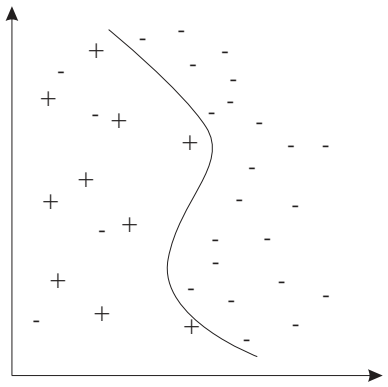

Figura 6.5: Conjunto de dados após a remoção de casos da classe majoritária por meio da criação de um subconjunto consistente.

Ligações Tomek é um método que pode ser bastante efetivo para identificar e remover ruído dos dados. Alguns exemplos da aplicação de ligações Tomek com essa finalidade podem ser encontrados em Lorena et al. (2002a,b).

Outros métodos utilizados para reduzir o número de exemplos em um conjunto de

treinamento com base no algoritmo K-VIZINHOS MAIS PRÓXIMOS são descritos em (Wilson & Martinez, 2000).

6.7 *Under-sampling*, *Over-sampling* e os Atuais Sistemas de Aprendizado

Uma grande parte da pesquisa realizada para solucionar o problema do aprendizado com conjuntos de dados com classes desbalanceadas tem base em métodos que visam balancear a distribuição das classes. Entretanto, algumas pesquisas recentes têm mostrado que diversos sistemas de aprendizado são insensíveis à distribuição das classes. Drummond & Holte (2000) mostram que existem critérios de divisão de nós em árvores de decisão que são relativamente insensíveis à distribuição das classes. Elkan (2001) faz declarações similares para o classificador *Naive Bayes* e para os algoritmos de árvore de decisão. Se um sistema de aprendizado é insensível à distribuição das classes, então alterar essa distribuição, isto é, balancear um conjunto de dados, pode ter pouco efeito nos classificadores induzidos.

Por outro lado, os métodos de *under-* e *over-sampling* têm sido empiricamente analisados em diversos domínios, com bons resultados. Japkowicz (2000) compara algumas abordagens para lidar com conjuntos com classes desbalanceadas, e conclui que *under-* e *over-sampling* são métodos efetivos para aprender na presença de conjuntos de dados com classes desbalanceadas.

Drummond & Holte (2000) declaram que *under-* e *over-sampling* devem ser analisados em termos de como eles afetam a poda e o rotulamento dos nós folhas. Entretanto, em diversos experimentos realizados por Provost & Fawcett (2001), os classificadores induzidos a partir de classes artificialmente balanceadas obtiveram resultados que foram, freqüentemente, melhores do que os obtidos com a distribuição natural das classes. Esses experimentos foram conduzidos sem realizar poda na árvore de decisão induzida, e com ajustes nos rótulos dos nós folhas para lidar com as mudanças realizadas na distribuição das classes no conjunto de treinamento.

Na próxima seção são apresentados alguns dos experimentos realizados para verificar se o método de seleção unilateral pode efetivamente melhorar o desempenho de classificação da classe minoritária. Os classificadores induzidos são analisados para verificar se existem diferenças entre o conhecimento gerado com e sem o balanceamento das classes.

6.8 Análise Experimental

Alguns experimentos foram realizados para verificar se o método de seleção unilateral é capaz de melhorar o desempenho de classificação da classe minoritária em um conjunto de dados com classes desbalanceadas. O sistema de aprendizado C4.5 (Quinlan, 1988) e o conjunto de dados **Hepatitis** do repositório UCI (Blake & Merz, 1998) foram utilizados nesse experimento. O conjunto de dados **Hepatitis** possui 155 exemplos, sendo 123 (79,3%) deles pertencentes à classe majoritária `live` e 32 exemplos (20,6%) pertencentes à classe minoritária `die`.

O conjunto de dados **Hepatitis** é conhecido na comunidade de Aprendizado de Máquina pela sua dificuldade em produzir bons resultados. De acordo com Holte (1993), muito poucos sistemas de aprendizado obtiveram uma precisão de dois pontos percentuais acima do *erro majoritário*, isto é, uma precisão dois pontos percentuais acima de 79,3%.

No experimento apresentado nesta seção, os conjuntos de treinamento são tratados por meio da seleção unilateral, e os conjuntos de teste são mantidos inalterados. O desempenho do sistema de aprendizado C4.5 foi medido no conjunto de dados original com todos os casos (a); no conjunto de dados sem os exemplos eliminados por meio de ligações Tomek (b); no conjunto de dados sem os exemplos eliminados por meio da criação de um subconjunto consistente (c); no conjunto de dados sem exemplos eliminados por ligações Tomek e pela criação de um subconjunto consistente (d); e por fim, no conjunto de dados sem alguns casos da classe majoritária removidos aleatoriamente (e).

As taxas de erro foram medidas utilizando o método de reamostragem *3-fold cross-validation*. O número de iterações $k = 3$ foi escolhido por causa do pequeno número de exemplos da classe minoritária. Um número maior para esse parâmetro poderia fazer com que o número de casos da classe minoritária ficasse muito reduzido em cada conjunto de teste, fazendo com que os resultados apresentassem uma alta variância. Para confirmar os resultados, o método *3-fold cross-validation* foi aplicado 3 vezes. Uma vez que os resultados obtidos em cada uma das três repetições foram similares, somente os resultados apresentados em um dos experimentos são mostrados na Tabela 6.2. Nessa tabela, na coluna *#Exemplos* é mostrado o número de exemplos no conjunto de treinamento para a classe majoritária e minoritária, respectivamente. Nas colunas FP e FN são apresentadas as taxas de falso positivo e falso negativo, além de seus respectivos desvios padrão. Na coluna E é apresentada a taxa de erro juntamente com o desvio padrão.

Os resultados obtidos sugerem que a seleção unilateral pode diminuir a taxa de nega-

	#Exemplos	FP	FN	E
a	80/23	$10,33 \pm 4,13$	$61,61 \pm 3,64$	$20,84 \pm 3,07$
b	70/23	$15,36 \pm 2,85$	$42,76 \pm 8,28$	$20,82 \pm 0,82$
c	62/23	$12,81 \pm 6,10$	$57,23 \pm 8,27$	$22,20 \pm 4,20$
d	55/23	$27,82 \pm 3,21$	$29,29 \pm 5,62$	$28,15 \pm 3,34$
e	50/23	$8,56 \pm 1,84$	$57,23 \pm 12,29$	$18,79 \pm 4,05$

Tabela 6.2: Resultados dos experimentos para o conjunto de dados **Hepatitis**.

tivo positivo, ou seja, o número de exemplos da classe minoritária que são incorretamente classificados. Esse fato ocorre principalmente quando ligações Tomek são utilizadas (b e d). O método de seleção unilateral obteve a menor taxa de falso negativo, entretanto, obteve a maior taxa de falso positivo. A taxa de erro obtida no conjunto de teste foi a maior se comparada com os demais métodos. Esses resultados eram esperados, como comentado na Seção 6.3 na página 144, uma vez que os exemplos da classe majoritária são maioria no conjunto de teste.

O método de seleção aleatória (e), o qual não utiliza nenhuma heurística, obteve resultados comparáveis à seleção por subconjuntos consistentes. Mesmo que o método de seleção aleatória não utilize nenhuma heurística, esse método possui o mérito de remover com a mesma probabilidade quaisquer casos. Comparando com os demais métodos utilizados neste trabalho, a seleção aleatória é o método que provavelmente causa a menor alteração na distribuição dos dados.

Possivelmente, a decisão de não remover os casos da classe minoritária, mesmo que possivelmente sejam ruído, pode ter influenciado nos resultados. Essa decisão foi tomada com base na necessidade de manter todos os poucos casos pertencentes à classe minoritária. Entretanto, o ruído presente na classe minoritária pode reduzir a precisão de classificação. Uma vez que as ligações Tomek não oferecem uma forma segura de distinguir entre exemplos que são ruído e exemplos próximos à borda de decisão, em trabalhos futuros pretende-se investigar outros métodos, ou melhoramentos às ligações Tomek, que sejam capazes de realizar essa distinção.

Deve ser observado que mesmo utilizando o método de reamostragem *3-fold cross-validation* os resultados apresentam grande variância. Essa variância faz com que seja difícil identificar diferenças significativas entre os resultados. Com a aplicação do teste-t pareado para *3-fold cross validation* foi possível identificar somente uma diferença significativa: a taxa de falso positivo para o método que remove exemplos com ligações Tomek e sub-conjuntos consistentes (d) foi inferior à taxa de falso positivo obtida com todos os dados (a), com 95% de confiança.

Uma análise das árvores de decisão induzidas mostra que as regras geradas antes da aplicação do método de seleção unilateral são sempre diferentes das regras geradas após a aplicação do método. Embora não se possa tirar conclusões mais gerais, o método de seleção unilateral tende a modificar a distribuição dos dados, e tal modificação de distribuição pode se refletir nas regras induzidas. Aparentemente, as conclusões apresentadas por Drummond & Holte (2000) não se aplicam à seleção unilateral. Entretanto, mais resultados experimentais são necessários para confirmar essa hipótese.

6.9 Considerações Finais

Aprender com conjuntos de dados com classes desbalanceadas é um tópico importante em Aprendizado de Máquina. Um método direto para lidar com conjuntos com classes desbalanceadas é balancear a distribuição das classes. Esse balanceamento pode ser obtido por meio da redução (*under-sampling*) dos casos da classe majoritária, ou do aumento (*over-sampling*) dos casos da classe minoritária, ou da aplicação de ambos os métodos em conjunto.

Existem diversos métodos na literatura que confirmam a eficiência dos métodos de *under-* e *over-sampling* em problemas práticos. Entretanto, também existem algumas evidências que mostram que balancear artificialmente a distribuição das classes não tem muito efeito no desempenho dos classificadores induzidos, uma vez que alguns sistemas de aprendizado são relativamente insensíveis às diferenças na distribuição das classes.

Nos experimentos realizados e descritos neste capítulo, o método de seleção unilateral conseguiu diminuir a taxa de falso negativo, isto é, conseguiu classificar melhor os exemplos da classe minoritária. Entretanto, o taxa de falso positivo também obteve um acréscimo considerável.

Uma análise preliminar dos classificadores induzidos mostra que as regras geradas após a aplicação do método de seleção unilateral são sempre diferentes das regras obtidas sobre todos os dados. Aparentemente, a aplicação do método de seleção unilateral pode influenciar nas regras induzidas, e não somente na poda e no rotulamento dos nós folhas, como foi notado por Drummond & Holte (2000) para outros método de *under-* e *over-sampling*.

Aparentemente é necessário haver um melhor entendimento de como a distribuição das classes afeta cada fase do processo de aprendizado. Por exemplo, em árvores de decisão, é necessário entender como a distribuição das classes afeta a indução da árvore,

sua poda e o rotulamento dos nós folhas. Um melhor entendimento desses tópicos básicos irá permitir que a comunidade projete melhores métodos de aprendizado para lidar com o problema de aprendizado com classes desbalanceadas.

Capítulo 7

Conclusão

7.1 Considerações Iniciais

Neste capítulo são apresentadas as conclusões deste trabalho. Na Seção 7.2 são apresentadas as principais contribuições desta tese; na Seção 7.3 são discutidas algumas limitações dos métodos propostos neste trabalho para tratamento de valores desconhecidos e de conjuntos com classes desbalanceadas; por fim, na Seção 7.4 são apresentadas algumas idéias para trabalhos futuros.

7.2 Principais Contribuições

A fase de pré-processamento de dados é tida como uma das fases do processo de KDD que requer mais tempo e conhecimento de domínio para a sua correta realização. Isso deve-se ao fato que, freqüentemente, os dados armazenados em sistemas de gerenciamento de dados apresentam uma grande quantidade e diversidade de problemas. Esse fato, somado ao objetivo de KDD de extrair conhecimento de grandes bases de dados, faz com que o pré-processamento de dados seja uma atividade que requer muito tempo e atenção por parte dos envolvidos no processo. Tal tempo e atenção são necessários para que os problemas nos dados sejam identificados e corrigidos cuidadosamente, sem que haja a introdução de novos problemas.

A fase de pré-processamento de dados se torna ainda mais importante pelo fato de que diversos algoritmos utilizados para extrair padrões dos dados não utilizam qualquer informação externa, com exceção dos próprios dados, para extrair os padrões. Dessa forma,

a qualidade dos resultados obtidos pelo processo de KDD é amplamente determinada pela qualidade dos dados de entrada. Isso faz com que pré-processamento de dados seja não somente uma fase trabalhosa, mas também uma fase que requer pesquisa e planejamento, uma vez que essa fase é crítica para o sucesso do processo de KDD.

Com o objetivo de tornar a fase de pré-processamento menos trabalhosa e mais dinâmica, este trabalho identifica algumas tarefas de pré-processamento de dados que podem ser realizadas por métodos com alto grau de automação. Este trabalho introduz o termo *tarefas fracamente dependentes de conhecimento de domínio* para identificar as tarefas de pré-processamento de dados que podem ser realizadas com pouco conhecimento de domínio. Nesse caso, o método que trata essa tarefa pode ser um método com alto grau de automação que utiliza as informações presentes nos próprios dados para tratar o problema de pré-processamento de dados.

Foram identificadas diversas tarefas de pré-processamento de dados fracamente dependentes de domínio, entre elas: o tratamento de valores desconhecidos, a identificação de casos extremos, a seleção de atributos relevantes, o tratamento de conjuntos de dados com classes desbalanceadas, entre outras.

Com relação à automação de tarefas fracamente dependentes de conhecimento de domínio, uma das principais contribuições deste trabalho é o projeto e desenvolvimento de um ambiente para pré-processamento de dados e gerenciamento de avaliações experimentais, ao qual foi dado o nome de Discover Learning Environmet — DLE. Esse ambiente tem como principal objetivo prover um *framework* para que métodos de pré-processamento de dados sejam implementados e avaliados. O ambiente DLE é constituído por dois módulos: a biblioteca de classes Discover Object Library — DOL e o ambiente gerenciador de avaliações experimentais Sniffer.

DOL é uma biblioteca que provê um conjunto de classes que podem ser utilizadas como base para a implementação de métodos de pré-processamento de dados. Os conceitos provenientes de *padrões de projeto* foram utilizados no projeto da biblioteca DOL. O uso de padrões de projeto tem como objetivo tornar o projeto da biblioteca DOL um projeto de alta qualidade, o qual pode ser facilmente modificado para incluir futuras extensões de funcionalidades à biblioteca.

O ambiente para gerenciamento de avaliações experimentais Sniffer complementa a biblioteca DOL, uma vez que os métodos implementados utilizando a biblioteca DOL podem ser avaliados experimentalmente pelo ambiente Sniffer. Análises experimentais possuem um papel de extrema importância em AM e outras áreas de pesquisa relacio-

nadas a KDD. Esse fato ocorre pois os métodos empregados e os dados analisados são, normalmente, muito complexos para um tratamento formal completo. Em outras palavras, para um dado problema não existem instrumentos formais para decidir qual método de extração automática de conhecimento é ótimo (Kibler & Langley, 1988; Dietterich, 1997a; Schaffer, 1994).

As avaliações experimentais são altamente repetitivas, o que as tornam cansativas de serem realizadas manualmente, e altamente sujeitas a erros. Por exemplo, nos experimentos realizados no Capítulo 5 foram realizadas 1120 execuções de indutores para cada conjunto de dados, com um total de 6720 execuções. O sistema SNIFFER automatiza o processo de avaliação experimental, realizando conversões entre sintaxes de conjuntos de dados, executando e extraindo as matrizes de confusão de diversos sistemas de aprendizado, convertendo os classificadores gerados em um formato padrão de regras, e realizando testes de significância entre os resultados. Os resultados obtidos são publicados em relatórios e tabulados em um formato que pode ser utilizado para gerar gráficos e tabelas, como aqueles apresentados no Capítulo 5.

Diversos relatórios técnicos foram publicados com o objetivo de documentar o projeto, a arquitetura e a implementação do ambiente DLE (Batista & Monard, 2003b,d; Kemp, Batista & Monard, 2002, 2001)

Vale notar que o ambiente DLE foi utilizado com sucesso, não somente neste trabalho, mas também em trabalhos desenvolvidos por vários pesquisadores do nosso grupo (Lorena, Batista, de Carvalho & Monard, 2002a,b; Milaré, Carvalho & Monard, 2002; Sanches, 2002; Pugliesi, 2001; Dosualdo, 2002), para realizar diversos experimentos com algoritmos de aprendizado.

Com relação às tarefas de pré-processamento de dados fracamente dependentes de conhecimento de domínio, foram eleitas para serem pesquisadas as seguintes tarefas de pré-processamento de dados: o tratamento de valores desconhecidos e o tratamento de conjuntos de dados com classes desbalanceadas. Como resultado da pesquisa foram propostos, implementados e avaliados experimentalmente métodos com alto grau de automação para realizar essas duas tarefas de pré-processamento de dados fracamente dependentes de conhecimento do domínio.

O tratamento de valores desconhecidos é um problema conhecido da comunidade de AM e outras áreas de pesquisa que compõem a área de KDD. Entretanto, diversos pesquisadores têm declarado encontrar bases de dados com mais de 50% dos valores ausentes em diversos atributos. Além disso, existe uma preocupação sobre a distribuição

dos valores desconhecidos. O tratamento de valores desconhecidos não aleatoriamente distribuídos pode introduzir distorções nos dados, e essas distorções podem ser refletidas no conhecimento extraído.

O tratamento de conjuntos de dados com classes desbalanceadas é um problema recente. Diversos algoritmos utilizados com freqüência em KDD, como os algoritmos que induzem árvores de decisão e regras de decisão, e outros algoritmos, como os algoritmos utilizados no treinamento de redes neurais, possuem dificuldades em aprender na presença de classes desbalanceadas. Em diversos domínios de aplicação existe uma diferença intrínseca na freqüência que ocorrem os eventos relacionados a cada classe, e esses domínios acabam por gerar conjuntos de dados desbalanceados. Alguns exemplos são o diagnóstico de doenças raras, a identificação de transações fraudulentas, a identificação de intrusões em sistemas de segurança, entre outros.

Portanto, tratamento de valores desconhecidos e de conjuntos com classes desbalanceadas são problemas atuais e importantes em AM e KDD. O tratamento de valores desconhecidos, pela necessidade de avaliar os métodos de tratamento de valores desconhecidos propostos em AM no contexto de KDD. O tratamento de conjuntos com classes desbalanceadas, por ser um problema recente e freqüentemente encontrado em dados do mundo real, o qual precisa ser superado para que KDD possa ser aplicado nos mais diversos domínios de aplicação. Nas próximas seções é comentado o que foi realizado em ambos os temas.

7.2.1 Tratamento de Valores Desconhecidos

Foi realizada uma pesquisa abrangente sobre os métodos de tratamento de valores desconhecidos. Alguns dos principais métodos de tratamento de valores desconhecidos foram identificados e estudados. Foi proposto o uso do algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação.

O método de imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS foi implementado no ambiente DLE. Diversos melhoramentos foram implementados sobre a versão básica do algoritmo K-VIZINHOS MAIS PRÓXIMOS como, por exemplo, a utilização de diversas funções de distância, incluindo a distância HVDM (Wilson & Martinez, 2000), o uso do método de acesso M-Tree (Ciaccia, Patella & Zezula, 1997), e o uso de pesos para determinar a classificação de um novo exemplo.

Foram realizados diversos experimentos com o objetivo de comparar o desempenho

do algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação com o desempenho obtido pelas estratégias internas utilizadas pelos sistemas C4.5 e CN2 para tratar valores desconhecidos, e com o desempenho obtido pela IMPUTAÇÃO PELA MÉDIA OU MODA.

Os resultados das análises experimentais mostram que o método de imputação proposto que utiliza o algoritmo K-VIZINHOS MAIS PRÓXIMOS obteve resultados superiores aos demais métodos de imputação para a maioria dos resultados. Esse método de imputação obteve bons resultados mesmo para grandes quantidades de valores desconhecidos inseridos com uma distribuição MCAR.

Como resultados da pesquisa sobre tratamento de valores desconhecidos realizada nesta tese foram publicados diversos trabalhos, incluindo um relatório técnico (Batista & Monard, 2003c), um artigo publicado em congresso nacional (Batista & Monard, 2001b), um artigo publicado em congresso latino-americano (Batista & Monard, 2001a), um artigo publicado em congresso internacional (Batista & Monard, 2002), e um artigo aceito para ser publicado em periódico internacional (Batista & Monard, 2003a).

7.2.2 Tratamento de Conjuntos com Classes Desbalanceadas

O tratamento de conjuntos de dados com classes desbalanceadas é um problema importante para a aplicação de algoritmos de AM em problemas reais. Foram pesquisados diversos problemas relacionados com o aprendizado com classes desbalanceadas como, por exemplo, a inadequação da taxa de erro e da precisão para medir o desempenho de classificadores em conjuntos com classes desbalanceadas. Foi estudado também a relação entre aprendizado sensível ao custo e a distribuição das classes.

Nas análises experimentais foi analisado o desempenho do método de *seleção unilateral*, o qual visa remover exemplos da classe majoritária de forma criteriosa. O desempenho do método de seleção unilateral foi comparado com o desempenho do método que remove exemplos da classe majoritária de forma aleatória.

A seleção unilateral obteve a maior redução na taxa de falso negativo, isto é, a maior redução no número de exemplos da classe minoritária classificados incorretamente. Entretanto, para esse método houve o maior aumento na taxa de falso positivo, isto é, no número de exemplos da classe majoritária classificados incorretamente. É necessário, portanto, analisar o custo de classificação incorreta para verificar se a redução na taxa de falso negativo compensa o aumento na taxa de falso positivo. As limitações do método de seleção unilateral são discutidas na Seção 7.3 na página seguinte.

Como resultados da pesquisa sobre tratamento de conjuntos com classes desbalanceadas realizada nesta tese foram publicados diversos trabalhos, incluindo dois artigos publicados em congressos nacionais (Batista & Monard, 1998; Batista, Carvalho & Monard, 1999), um artigo publicado em congresso internacional (Monard & Batista, 2002), e um artigo publicado e premiado em congresso internacional (Batista, Carvalho & Monard, 2000).

7.3 Limitações

Os métodos avaliados experimentalmente para tratamento de valores desconhecidos e de conjuntos com classes desbalanceadas possuem algumas limitações, descritas nas próximas seções.

7.3.1 Tratamento de Valores Desconhecidos

De uma forma geral, o método de imputação de valores desconhecidos obteve bons resultados nas análises experimentais realizadas neste trabalho. Entretanto, o uso das formas mais simples de imputação, como a IMPUTAÇÃO PELA MÉDIA OU MODA, deve ser evitado. A IMPUTAÇÃO PELA MÉDIA OU MODA pode distorcer os dados, introduzindo falsos padrões e alterando as relações entre os atributos, sobretudo quando os valores desconhecidos são introduzidos de forma não aleatória.

Mesmo quando formas sofisticadas de imputação são utilizadas, os valores imputados são freqüentemente mais bem comportados do que os valores reais, não conhecidos, seriam. Esse fato pode fazer com que os modelos induzidos com dados tratados sejam mais simples do que seriam os modelos construídos sobre todos os dados. Dessa forma, pode-se correr o risco de simplificar excessivamente o problema em estudo, sobretudo quando existem grandes quantidades de valores desconhecidos.

O algoritmo K-VIZINHOS MAIS PRÓXIMOS como método de imputação obteve muito bons resultados em diversos conjuntos de dados, entretanto, esse método não foi capaz de superar os demais métodos no conjunto de dados **Breast**. Esse conjunto de dados possui fortes correlações entre seus atributos. Uma análise mais profunda mostrou que o indutor C4.5 foi capaz de substituir os atributos com valores desconhecidos por outros atributos com alta correlação e, assim, obter resultados superiores a imputação com K-VIZINHOS MAIS PRÓXIMOS. Os resultados obtidos com o conjunto de dados **Sonar** foram similares

aos resultados obtidos com o conjunto de dados **Breast**.

Assim, sendo os valores imputados aproximações dos valores reais, recomenda-se, antes de utilizar um método de imputação, procurar verificar se não é possível coletar os dados ausentes ou, até mesmo, verificar se não existe um outro atributo com informações similares, isto é, alta correlação, no conjunto de dados.

7.3.2 Tratamento de Conjuntos com Classes Desbalanceadas

Para muitos problemas reais, criar um classificador que classifique bem as duas classes, isto é, que apresente baixas taxas de falso positivo e de falso negativo, é uma tarefa muito difícil. Freqüentemente, existe uma relação de perda e ganho entre as taxas de falso positivo e falso negativo.

Em conjuntos de dados com classes desbalanceadas, o objetivo principal é melhorar a classificação da classe minoritária. Entretanto, atingir esse objetivo muitas vezes resulta em aumentar o erro de classificação da classe majoritária. Nos experimentos realizados com o método de seleção unilateral proposto houve a maior redução na taxa de falso negativo entre todos os métodos, mas essa redução foi acompanhada pelo maior aumento na taxa de falso positivo.

Uma das principais limitações do método de seleção unilateral é, dados os custos de classificação de cada classe, identificar quantos exemplos da classe majoritária precisam ser removidos para que o método forneça bons resultados. Essa limitação é, provavelmente, a principal limitação de outros métodos de *under-* e *over-sampling* que utilizam heurísticas para remover ou adicionar exemplos. Possivelmente, os trabalhos de Breiman, Friedman, Olshen & Stone (1984); Elkan (2001) podem ser utilizados como diretrizes para procurar por alguma solução para esse problema, entretanto, uma análise mais detalhada ainda é necessária.

7.4 Trabalhos Futuros

Algumas sugestões de possíveis refinamentos e extensões dos métodos apresentados neste trabalho, além de algumas novas idéias que surgiram durante o desenvolvimento desta tese, são apresentadas a seguir.

Tratamento de valores desconhecidos

Em trabalhos futuros, pretende-se inserir valores desconhecidos com distribuições MAR e NMAR. Sob essas distribuições é esperado que o método de imputação baseado no algoritmo K-VIZINHOS MAIS PRÓXIMOS obtenha resultados ainda melhores que a IMPUTAÇÃO PELA MÉDIA OU MODA, entretanto, ainda resta verificar se essa superioridade é confirmada experimentalmente. Alguns padrões NMAR de valores desconhecidos podem ser altamente perigosos como, por exemplo, quando todos os valores de um atributo acima ou abaixo de um limiar são desconhecidos. Esse padrão pode ocorrer, por exemplo, com um sensor que não funciona corretamente abaixo ou acima de uma determinada temperatura. É necessário pesquisar se os métodos de imputação são robustos nesse tipo de cenário.

Um Algoritmo Genético — AG —poderia ser utilizado como método de imputação. Nesse caso, o AG realizaria a busca para encontrar um conjunto de valores que substituiriam os valores desconhecidos. Uma vantagem do AG sobre os demais métodos de imputação é que a função objetivo do AG poderia levar em consideração diversas medidas estatísticas dos dados completos que deveriam ser mantidas nos dados imputados como, por exemplo, as covariâncias entre os atributos, a média e a variância de cada atributo, entre outras.

Tratamento de conjuntos desbalanceados

Vários trabalhos podem ser sugeridos para estender algumas das conclusões apresentadas nesta tese. Inicialmente, pretende-se avaliar o método de seleção unilateral em outros conjuntos de dados, com o objetivo de obter conclusões mais abrangentes.

Pretende-se investigar novos métodos de identificação de ruído nos dados, e possíveis extensões para o método de ligações Tomek. Essas extensões têm como principal objetivo discriminar os casos que são ruído dos casos próximos à borda de decisão. Uma heurística bastante simples que procura realizar essa distinção é verificar as classes dos exemplos mais próximos dos exemplos que formam uma ligação Tomek. Se uma grande quantidade dos exemplos mais próximos a uma ligação Tomek for de uma única classe, então, provavelmente, um dos exemplos que formam a ligação Tomek é ruído.

Pretende-se aliar um método de *under-sampling*, como as ligações Tomek, com um método de *over-sampling*, de forma a não reduzir excessivamente o número de exemplos do conjunto de dados. Um método de *over-sampling*, que consideramos bastante promissor, consiste em encontrar exemplos com o algoritmo 1-VIZINHOS MAIS PRÓXIMO que sejam próximos e da mesma classe. Um novo exemplo pode ser criado por

meio da interpolação dos valores dos dois exemplos. Dessa forma, não são criados exemplos repetidos, o que poderia causar *overfitting*.

Por fim, pretende-se avaliar os resultados obtidos por meio de avaliações experimentais com a ajuda de gráficos ROC, e pretende-se substituir as medidas de falso positivo e falso negativo pela área sob a curva ROC — AUC.

Discover Learning Environment — DLE

Quanto ao ambiente DLE, pretende-se adicionar ao ambiente SNIFFER uma interface que permita criar gráficos ROC para avaliar os sistemas de aprendizado. Além disso, pretende-se utilizar a medida AUC como um índice numérico que fornece uma informação mais segura sobre o desempenho do sistema de aprendizado.

Apêndice A

A Sintaxe Discover Dataset Sintax — DSX

A.1 Considerações Iniciais

Na etapa de Mineração de Dados do processo de KDD, muito freqüentemente são utilizados sistemas de aprendizado, tanto acadêmicos quanto comerciais para a extração de padrões. Infelizmente, não houve uma padronização no formato do arquivo de dados utilizado como entrada para esses sistemas. Como resultado, diferentes sistemas de aprendizado utilizam diferentes sintaxes de arquivos de dados. Os sistemas de aprendizado acadêmicos normalmente aceitam como entrada somente arquivos texto em um formato proprietário. Os sistemas de aprendizado comerciais geralmente aceitam, além de arquivos texto, outras formas de entrada de dados, como por exemplo, conexões nativas a bancos de dados SQL e interface ODBC™.

Realizar uma investigação que envolve extrair conhecimento de vários conjuntos de dados utilizando diversos sistemas de aprendizado é normalmente muito trabalhoso pois necessita, entre outros, converter os arquivos de dados para a sintaxe utilizada por cada sistema de aprendizado. Para simplificar esse trabalho dentro do projeto Discover, foi decidido adotar uma sintaxe padrão para conjuntos de dados. A partir dessa sintaxe padrão, é possível utilizar a biblioteca de classes DOL para converter um arquivo de dados para a sintaxe utilizada em diversos sistemas de aprendizado, tais como as sintaxes dos sistemas listados na Tabela 4.1 na página 54.

Dessa forma, decidiu-se criar uma nova sintaxe para o ambiente Discover, a qual

foi dado o nome de DSX (*Discover Dataset Sintax*), ao invés de utilizar alguma sintaxe já definida por outros pesquisadores, como por exemplo as sintaxes utilizadas nos projetos $\mathcal{MLC}$++ (Kohavi, Sommerfield & Dougherty, 1997) e Weka (Witten & Frank, 2000). Essa decisão é justificada uma vez que as seguintes características são desejáveis na nova sintaxe:

Suporte a diversos tipos da dados
: Apesar de que os sistemas de aprendizado mais tradicionais são limitados a utilizar dados de tipo numérico (inteiros e reais) e nominal, os sistemas de aprendizado mais recentes, sobretudo os sistemas de aprendizado comerciais, são capazes de utilizar outros tipos de dado tais como data e hora. Dessa forma, a nova sintaxe deve dar suporte aos tipos de dado mais atuais. Para a definição desses tipos, foram levantados os tipos de dado mais utilizados em sistemas gerenciadores de banco de dados, sendo que os tipos de dado mais utilizados nesses sistemas foram incorporados à sintaxe DSX;

Suporte a indução construtiva apoiada pelo usuário
: É bastante comum que o usuário deseje construir novos atributos a partir de atributos já presentes nos dados (Lee, 2000). Isso ocorre, pois um atributo que compõe informações sobre dois ou mais atributos pode ser muito mais relevante para a resolução do problema do que os atributos separados. Por exemplo, em concessão de crédito pessoal, dois atributos altamente relevantes são a renda do cliente e o valor da prestação a ser paga. Entretanto, o percentual da renda do cliente comprometido com o pagamento da prestação pode fornecer uma medida mais direta se o cliente pode ou não ser capaz de honrar esse compromisso. A sintaxe DSX provê uma forma muito simples de realizar indução construtiva apoiada pelo usuário, por meio da definição de atributos virtuais, os quais podem ser definidos utilizando expressões aritméticas ou lógicas envolvendo um ou mais atributos existentes nos dados;

Suporte a diversas tarefas de aprendizado
: Conjuntos de dados declarados com a sintaxe DSX podem ser utilizados tanto em aprendizado supervisionado quanto não supervisionado. Ainda, em aprendizado supervisionado, os dados podem ser utilizados em problemas de classificação ou regressão. Portanto, a sintaxe padrão deve dar suporte a conjuntos de dados que tenham classe nominal ou numérica, ou ainda não possuem uma classe definida explicitamente.

O projeto $\mathcal{MLC}$++ utiliza uma sintaxe para conjuntos de dados muito similar a

sintaxe do sistema C4.5. O projeto WEKA propõe uma sintaxe chamada ARFF — *Attribute Relation Format File*. Ambas sintaxes possuem limitações quanto aos objetivos propostos anteriormente. Tanto a sintaxe ARFF quanto a sintaxe utilizada pelo sistema C4.5 dão suporte somente aos tipos de dado numérico e nominal. Também, essas sintaxes não oferecem suporte para realizar indução construtiva apoiada pelo usuário.

Nas próximas seções é feita uma apresentação detalhada da sintaxe DSX e das suas principais características.

A.2 Uma Visão Geral da Sintaxe DSX

A sintaxe padrão utiliza arquivos texto para declarar os atributos e seus respectivos tipos, e os valores que esses atributos assumem em um conjunto de dados. Os atributos são declarados em um arquivo com a extensão `.names`. Os valores que esses atributos assumem em um conjunto de dados são declarados em um outro arquivo com a extensão `.data`. Os dois arquivos devem possuir o mesmo nome, se diferenciando apenas pela extensão. Opcionalmente, pode haver também arquivos de dados com as extensões `.test` com casos rotulados de teste para medir o erro de classificação, `.validation` com casos rotulados para validação de modelos e `.cases` com casos não rotulados para serem rotulados por um classificador.

A seguir é mostrado um conjunto de exemplos na sintaxe DSX. O conjunto de exemplos utilizado é o conjunto de dados artificial `voyage` (Quinlan, 1988). O arquivo de declaração de atributos, declarado no arquivo `voyage.names`, é descrito na Tabela A.1.

```
1       class.           | Class Attribute
2
3       | Attributes
4       outlook:       nominal (sunny, overcast, rain).
5       temperature:   integer.
6       humity:        integer.
7       windy:         nominal (yes, no).
8       class:         nominal (go, dont_go).
```

Tabela A.1: Exemplo de arquivo de declaração de atributos: `voyage.names`.

A primeira declaração em um arquivo de declaração de atributos define qual deve ser o atributo classe, se houver atributo classe definido. No caso de aprendizado supervisi-

```
1       sunny, 25, 72, yes, go
2       sunny, 28, 91, yes, dont_go
3       overcast, 23, 90, yes, go
4       overcast, 29, 78, no, go
5       rain, 22, 95, no, go
6       rain, 19, 70, yes, dont_go
```

Tabela A.2: Exemplo de arquivo de declaração de dados: `voyage.data`.

onado, o atributo classe é mandatório, e esse atributo pode ser qualquer atributo presente no conjunto de dados. Nesse exemplo, o atributo classe é o atributo `class`, o qual é declarado posteriormente como sendo um atributo nominal que pode assumir os valores `go` e `dont_go`. No caso de conjuntos de dados para aprendizado não supervisionado, o nome do atributo classe deve ser substituído pela palavra `null`, a qual indica que o conjunto de dados não possui classe definida. Após a declaração do atributo classe, os demais atributos são declarados. Cada atributo possui um identificador e um tipo de dado associado. São considerados identificadores válidos aqueles que são combinações de números, letras e "_" (*underscore*), em qualquer seqüência. Para identificadores mais complexos, que envolvem outros caracteres que não sejam os especificados anteriormente (como espaços, letras acentuadas, etc), é necessário colocar o identificador entre aspas. Dessa forma, são identificadores válidos: `abc`, `1`, `1a`, `_1a`, `"_12a"` e `"válido"`.

Além do arquivo de declaração de atributos, existe ainda o arquivo de declaração de dados. Nesse arquivo são declarados os valores que os atributos presentes no arquivo de declaração de atributos assumem para um determinado conjunto de exemplos. A Tabela A.2 mostra um trecho desse arquivo na sintaxe DSX para o conjunto de dados **voyage**.

Cada linha de um arquivo de declaração de dados representa um exemplo. Sendo assim, o caracter responsável por identificar o final de um exemplo (separador de registros) é o caracter de nova linha (representado em muitas linguagem de programação por "`\n`"). Cada exemplo possui uma seqüência de valores separados por vírgula, ou seja, a vírgula é o caracter responsável por separar os valores de um exemplo (separador de campos). Os valores declarados em cada exemplo, em um arquivo de declaração de dados, devem estar na mesma ordem em que foram declarados no arquivo de declaração de atributos.

A.3 Os Tipos de Dado da Sintaxe DSX

Como mencionado anteriormente, a sintaxe DSX deve suportar, além dos tipos de dado mais comuns (como os tipos numérico e nominal), tipos de dado mais sofisticados. Com a popularização do uso de Mineração de Dados para extrair conhecimento de Bases de Dados, os sistemas de aprendizado têm passado a suportar novos tipos de dado, como por exemplo os tipos data e hora. É importante que a sintaxe DSX ofereça suporte a diversos tipos de dado para que a sintaxe não limite o uso dos sistemas de aprendizado mais recentes. Por exemplo, se a sintaxe não suporta o tipo de dados data, então dados com esse tipo não podem ser declarados nessa sintaxe. Logo, se o conjunto de dados for convertido da sintaxe padrão para a sintaxe do sistema de aprendizado, o sistema de aprendizado não poderá utilizar a sua capacidade de processar dados do tipo data. Por outro lado, se a sintaxe DSX oferecer suporte aos tipos de dado mais sofisticados, é possível converter um conjunto de dados que utiliza esses tipos de dado mesmo para os sistemas de aprendizado que não suportam esses tipos. Isso pode ser feito pela definição de conversões padrão entre os tipos de dado mais complexos e os tipos de dado mais utilizados (numérico e nominal) pelos sistemas de aprendizado mais tradicionais.

Para definir quais tipos de dado devem ser suportados pela sintaxe DSX, foram analisados os principais tipos de dado suportados pelos sistemas gerenciadores de bancos de dados. Após essa análise foi decidido dar suporte, na sintaxe DSX, aos tipos de dado listados na Tabela A.3.

`Nominal`	`Enumerated`
`Integer`	`Real`
`Date`	`Time`
`String`	

Tabela A.3: Tipos de dado suportados pela sintaxe DSX.

Nas próximas seções são descritos em mais detalhes cada um dos tipos de dado aceitos pela sintaxe DSX.

A.3.1 O Tipo de Dado `Nominal`

O tipo `nominal` é utilizado para declarar um atributo que pode assumir um grupo restrito de valores. Existem duas formas de declarar um atributo do tipo `nominal`:

1. A primeira consiste na palavra `nominal` seguida de uma lista de valores. Essa lista

de valores dita quais são os possíveis valores que o atributo pode assumir. Esta opção é amplamente recomendada, uma vez que com essa informação é possível realizar verificações de tipo de dado;

2. A segunda consiste apenas na palavra `nominal`, sem a declaração de uma lista de valores. Nesse caso, o atributo em questão pode assumir qualquer valor, e verificações de tipo de dado não são possíveis. Entretanto, essa forma de declaração é útil quando o atributo do tipo `nominal` pode assumir um grupo mais numeroso de valores, e o usuário não deseja digitar todos esses valores no arquivo de declaração de atributos.

São exemplos de declarações de atributos do tipo `nominal`:

```
AtribEx1: nominal (azul, amarelo, vermelho).
AtribEx2: nominal.
```

A.3.2 O Tipo de Dado `Enumerated`

O tipo de dado `enumerated` é semelhante ao tipo de dado `nominal`. A principal diferença é que com o tipo `enumerated` é possível definir uma ordem entre os valores que o atributo pode assumir. Entretanto, não existe uma definição explícita de distância entre esses valores. Um exemplo de tipo `enumerated` é um atributo que pode assumir os valores `pequeno`, `médio` e `grande`.

Existe somente uma forma de declarar um atributo do tipo `enumerated`. Consiste na palavra `enumerated` seguida de uma lista de valores que o atributo pode assumir. A lista de valores é obrigatória, pois sem ela não é possível identificar a ordem dos atributos, que é a principal informação desse tipo de dado. A lista de valores é ordenada de forma crescente.

Um exemplo de declaração de atributos de tipo `enumerated` é:

```
AtribEx3: enumerated(pequeno, "médio", grande).
```

A.3.3 O Tipo de Dado `Integer`

O tipo de dado `integer` é utilizado para declarar um atributo que pode assumir um valor inteiro. Alguns sistemas de aprendizado, como o C4.5, não possuem um tipo inteiro, e

os atributos numéricos (inteiros e reais) são declarados como um único tipo de dado. No caso específico do C4.5 é utilizado o tipo de dado `continuous`. Na sintaxe DSX, inteiros e reais são declarados separadamente, o tipo `real` é apresentado na próxima seção. A declaração de um atributo inteiro é feita como no exemplo a seguir:

```
AtribEx4: integer.
```

A.3.4 O Tipo de Dado `Real`

O tipo `real` é semelhante ao tipo de dado `integer`, com a diferença que um atributo real pode armazenar números com ou sem parte fracionária. Um exemplo de declaração de atributo do tipo `real` é:

```
AtribEx5: real.
```

A.3.5 O Tipo de Dado `String`

O tipo de dado `string` não está presente na maioria dos sistemas de aprendizado. Esse tipo foi incluído para dar algum suporte à *Mineração de Textos* (*Text Mining*) (Dörre, Gerstl & Seiffert, 1999). Um atributo do tipo `string` pode assumir como valor uma seqüência de caracteres de tamanho indefinido, essa seqüência pode conter quaisquer caracteres incluindo quebra de linha (`\n`). Para identificar os limites de um dado do tipo `string` é necessário inserir aspas no início e no fim da seqüência de caracteres. Um exemplo de declaração de atributo do tipo `string` é:

```
AtribEx6: string.
```

A.3.6 O Tipo de Dado `Date`

O tipo de dado `date` permite declarar atributos que podem armazenar datas (dia, mês e ano). A princípio, os valores das datas devem estar no formato `aaaa/mm/dd`, o qual é o formato adotado pela maioria dos sistemas gerenciados de bancos de dados. Entretanto, é possível utilizar declarações estendidas (Seção A.5 na página 177) para informar datas em outros formatos. Um exemplo de declaração de um atributo do tipo de dado `date` é:

```
AtribEx7: date.
```

A.3.7 O Tipo de Dado `Time`

O tipo de dado `time` permite declarar um atributo que pode conter um horário (hora, minuto e segundo). A princípio, os valores dos horários devem estar no formato "hh:mm:ss". Entretanto, é possível utilizar declarações estendidas (Seção A.5) para informar horários em outros formatos. Um exemplo de declaração de um atributo do tipo de dado `time` é:

```
AtribEx8: time.
```

A.4 Atributos Virtuais

Como já mencionado, a sintaxe DSX permite realizar indução construtiva apoiada pelo usuário. Para isso, *atributos virtuais* podem ser definidos no arquivo de definição de atributos. Atributos virtuais são atributos que são definidos por meio de expressões aritméticas ou lógicas que envolvem um ou mais atributos já definidos no arquivo de declaração de atributos. Os valores que os atributos virtuais assumem no conjunto de dados dependem somente da expressão aritmética que os definem. Esses atributos não possuem dados declarados no arquivo de declaração de dados. Assim, quaisquer modificações nos valores dos atributos envolvidos na expressão aritmética acarreta em uma modificação automática no valor do atributo virtual.

A expressão aritmética que define um atributo virtual pode ser qualquer expressão aritmética válida em Perl (Wall, Christiansen & Schwartz, 1996). Resumidamente, os identificadores dos atributos assumem o papel de variáveis. O símbolo "$" deve ser adicionado ao identificador da variável. Alguns operadores válidos são + (adição), - (subtração), / (divisão), * (multiplicação), % (resto de divisão), ** (exponenciação), além de funções de conversão de tipos e trigonométricas.

A Tabela A.4 na próxima página mostra o arquivo de declaração de atributos, `voyage.names` (Tabela A.1 na página 171) adicionado do atributo virtual `temp_humity_rate`. A declaração de um atributo virtual inicia com seu identificador e tipo e segue com o símbolo `:=`. Após esse símbolo é utilizada uma expressão aritmética ou lógica válida em Perl. Na expressão somente podem ser utilizados como variáveis os atributos declarados antes do atributo virtual. No exemplo anterior, o atributo `class` não poderia ser utilizado na expressão que define o atributo virtual `temp_humity_rate`, pois `class` está declarado após o atributo virtual.

```
1       class.          | Class Attribute
2
3       | Attributes
4       outlook:          nominal (sunny, overcast, rain).
5       temperature:      integer.
6       humity:           integer.
7       windy:            nominal (yes, no).
8       temp_humity_rate: real    := $temperature/$humity.
9       class:            nominal (go, dont_go).
```

Tabela A.4: Exemplo de arquivo de declaração de atributos, `voyage.names`, com declaração de atributo virtual.

A.5 Declarações Estendidas

Declarações estendidas permitem declarar propriedades adicionais aos atributos. Essas propriedades podem ser definidas conforme as necessidades do usuário. Uma declaração estendida deve ser especificada no arquivo de declaração de atributos. Cada declaração estendida é associada a um atributo especifico. Uma declaração estendida permite, por exemplo, declarar qual é o valor mínimo e o máximo que um atributo numérico pode assumir. Caso essa informação seja conhecida, especificar esses valores é mais confiável do que estimar os valores mínimo e o máximo com base no conjunto de exemplos. Essa informação pode ser utilizada para, por exemplo, normalizar o atributo.

Uma declaração estendida inicia com o símbolo de pois pontos (`:`) seguido por um identificador da declaração estendida. Se a declaração estendida requerer a especificação de um ou mais parâmetros, esses valores podem ser declarados entre parênteses logo após o identificador da declaração estendida. A sintaxe utilizada nas declarações estendidas é baseada na sintaxe dos *funtores* da linguagem de programação lógica Prolog (Bratko, 1990). Um exemplo de declaração estendida é:

```
AtribEx4: integer: min(0): max(100).
```

Nesse exemplo, foram utilizadas duas declarações estendidas para informar o valor máximo e mínimo para o atributo. A declaração `min(0)` informa que o valor mínimo que o atributo pode assumir é 0 e `max(100)` informa que o valor máximo é 100.

Atualmente, diversas declarações estendidas são utilizadas pela biblioteca DOL. Entre as principais estão:

`min`, `max` e `std_dev`

Utilizada para atributos dos tipos `real` e `integer`. Declara qual é o valor mínimo e máximo que o atributo pode assumir nos dados, e o desvio padrão do atributo, respectivamente. Essa informação e utilizada pelo módulo que normaliza os dados;

`date_language` e `date_order`

Utilizada para atributos do tipo `date`. Em `date_language` é especificada a língua em que os valores de um atributo do tipo `date` estão escritos. Por exemplo, o valor "Segunda, 7 de janeiro de 2003" corresponde a uma data em português. Atualmente, a biblioteca DOL aceita datas em sete línguas diferentes, listadas na Tabela A.5. Em `date_order` é especificada a ordem na qual os valores do dia, mês e ano estão especificados. Essa declaração pode assumir três valores `dmy` para dia-mês-ano, `mdy` para mês-dia-ano e `ymd` para ano-mês-dia.

Identificador da língua	Língua
English	Inglês
Français	Francês
Deutsch	Alemão
Español	Espanhol
Português	Português
Nederlands	Holandês
Italiano	Italiano
Norsk	Norueguês
Svenska	Sueco
Dansk	Dinamarquês
Suomi	Finlandês

Tabela A.5: As línguas aceitas pela definição estendida `date_language`.

A.6 Gramática da Sintaxe DSX

A sintaxe DSX para arquivos de declaração de atributos `.names` pode ser mais formalmente definida pela seguinte gramática:

```
S ::=
    < class-defs >
  | < feature-defs >

< class-defs > ::=
    < feature-name > .
  | null .

< feature-name > ::=
    < identifier >

< feature-defs > ::=
    < feature-name > : < feature-type > .
  | < feature-name > : < feature-type > : < ext-defs > .
  | < feature-name > : < feature-type > := < expr > : < ext-defs > .

< feature-type > ::=
    real
  | integer
  | boolean
  | nominal
  | nominal ( < list > )
  | enumerated ( < list > )
  | date
  | time
  | string

< ext-defs > ::=
    < ext-def >
  | < ext-def > : < ext-defs >

< ext-def > ::=
    < identifier >
  | < identifier > ( < list > )

< list > ::=
    < identifier >
  | < identifier > , < list >
```

Um identificador válido (`< identifier >`) pode ser qualquer seqüência de letras, números e *underscores* (_). Ainda, são considerados identificadores válidos quaisquer seqüências de caracteres desde que colocados entre aspas (“).

Uma expressão aritmética (`< expr >`) pode ser qualquer expressão aritmética válida em Perl.

Apêndice B

Relatórios do Ambiente Sniffer

B.1 Exemplo de Relatório Resumido

```
1  ========================================================================
2  Sniffer v1.0                                          By Gustavo Batista
3  Discover Project                              LABIC - ICMC - USP - Brazil
4  gbatista@icmc.sc.usp.br              http://www.icmc.sc.usp.br/~gbatista
5  Date: 01/05/2003                                          Time: 15:03:10
6  ========================================================================
7                             SUMMARY REPORT
8  ========================================================================
9
10 Base Directory: ./Experimento/breast/c4.5           Inducer: C4.5
11
12         benign  malignan   Overall
13
14 Mean      3.15%     6.70%     4.40%
15 SE        0.50%     1.42%     0.69%
16
17 ------------------------------------------------------------------------
18
19 Base Directory: ./Experimento/breast/c4.5rules      Inducer: C4.5Rules
20
21         benign  malignan   Overall
22
23 Mean      3.16%     4.64%     3.66%
```

```
SE        1.40%     1.83%     1.16%

--------------------------------------------------------------------------

Base Directory: ./Experimento/breast/cn2           Inducer: CN2

        benign  malignan   Overall

Mean      2.02%     7.10%     3.81%
SE        0.52%     2.15%     0.59%

--------------------------------------------------------------------------

Base Directory: ./Experimento/bupa/c4.5            Inducer: C4.5

             1         2   Overall

Mean     52.86%    20.00%    33.87%
SE        5.29%     3.16%     3.22%

--------------------------------------------------------------------------

Base Directory: ./Experimento/bupa/c4.5rules       Inducer: C4.5Rules

             1         2   Overall

Mean     51.24%    19.50%    32.82%
SE        4.53%     3.76%     2.51%

--------------------------------------------------------------------------

Base Directory: ./Experimento/bupa/cn2             Inducer: CN2

             1         2   Overall

Mean     60.81%    13.50%    33.34%
SE        2.76%     1.83%     1.19%

--------------------------------------------------------------------------

Base Directory: ./Experimento/cmc/c4.5             Inducer: C4.5

             1         2         3   Overall
```

```
67
68          Mean     38.46%    64.54%    50.45%    48.54%
69          SE        2.07%     2.98%     2.99%     1.61%
70
71          ---------------------------------------------------------------------
72
73          Base Directory: ./Experimento/cmc/c4.5rules         Inducer: C4.5Rules
74
75                       1         2         3   Overall
76
77          Mean     42.29%    60.68%    38.93%    45.28%
78          SE        2.25%     3.40%     2.87%     1.89%
79
80          ---------------------------------------------------------------------
81
82          Base Directory: ./Experimento/cmc/cn2               Inducer: CN2
83
84                       1         2         3   Overall
85
86          Mean     23.85%    60.42%    74.37%    49.63%
87          SE        1.94%     4.21%     2.44%     1.60%
88
89          ---------------------------------------------------------------------
90
91          Base Directory: ./Experimento/pima/c4.5             Inducer: C4.5
92
93                       0         1   Overall
94
95          Mean     16.20%    43.29%    25.64%
96          SE        1.53%     3.04%     1.73%
97
98          ---------------------------------------------------------------------
99
100         Base Directory: ./Experimento/pima/c4.5rules        Inducer: C4.5Rules
101
102                      0         1   Overall
103
104         Mean     17.00%    42.11%    25.78%
105         SE        1.72%     2.44%     1.10%
106
107         ---------------------------------------------------------------------
108
109         Base Directory: ./Experimento/pima/cn2              Inducer: CN2
```

```
                    0         1   Overall

Mean        7.60%    62.26%    26.68%
SE          1.51%     3.55%     1.65%
```

B.2 Exemplo de Relatório Detalhado

```
=========================================================================
Sniffer v1.0                                          By Gustavo Batista
Discover Project                                LABIC - ICMC - USP - Brazil
gbatista@icmc.sc.usp.br                http://www.icmc.sc.usp.br/~gbatista
Date: 01/05/2003                                            Time: 15:03:21
=========================================================================
                             DETAILED REPORT
=========================================================================

Base Directory: ./Experimento/breast/c4.5
Inducer:        C4.5
Status:         CLOSED (Everything looks fine)

Class Value 0: benign
Class Value 1: malignant

-------------------------------------------------------------------------

Fold: 0          File: ./Experimento/breast/c4.5/it0/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       44        1          2.22%
 malignan        2       22          8.33%

Error Rate for All Classes:           4.35%
```

```
28  ---------------------------------------------------------------------
29
30  Fold: 1          File: ./Experimento/breast/c4.5/it1/breast_C45.c45out
31
32              PREDICTED
33  ACTUAL     benign malignan    Class Error
34     benign       44        1          2.22%
35   malignan        0       24          0.00%
36
37  Error Rate for All Classes:           1.45%
38
39  ---------------------------------------------------------------------
40
41  Fold: 2          File: ./Experimento/breast/c4.5/it2/breast_C45.c45out
42
43              PREDICTED
44  ACTUAL     benign malignan    Class Error
45     benign       43        2          4.44%
46   malignan        1       23          4.17%
47
48  Error Rate for All Classes:           4.35%
49
50  ---------------------------------------------------------------------
51
52  Fold: 3          File: ./Experimento/breast/c4.5/it3/breast_C45.c45out
53
54              PREDICTED
55  ACTUAL     benign malignan    Class Error
56     benign       43        2          4.44%
57   malignan        2       21          8.70%
58
59  Error Rate for All Classes:           5.88%
60
61  ---------------------------------------------------------------------
62
63  Fold: 4          File: ./Experimento/breast/c4.5/it4/breast_C45.c45out
64
65              PREDICTED
66  ACTUAL     benign malignan    Class Error
67     benign       43        1          2.27%
68   malignan        2       22          8.33%
69
70  Error Rate for All Classes:           4.41%
```

```
71
72      ------------------------------------------------------------------------
73
74      Fold: 5          File: ./Experimento/breast/c4.5/it5/breast_C45.c45out
75
76                PREDICTED
77      ACTUAL     benign malignan    Class Error
78         benign       42        2          4.55%
79       malignan        2       22          8.33%
80
81      Error Rate for All Classes:           5.88%
82
83      ------------------------------------------------------------------------
84
85      Fold: 6          File: ./Experimento/breast/c4.5/it6/breast_C45.c45out
86
87                PREDICTED
88      ACTUAL     benign malignan    Class Error
89         benign       42        2          4.55%
90       malignan        4       20         16.67%
91
92      Error Rate for All Classes:           8.82%
93
94      ------------------------------------------------------------------------
95
96      Fold: 7          File: ./Experimento/breast/c4.5/it7/breast_C45.c45out
97
98                PREDICTED
99      ACTUAL     benign malignan    Class Error
100        benign       43        1          2.27%
101      malignan        1       23          4.17%
102
103     Error Rate for All Classes:           2.94%
104
105     ------------------------------------------------------------------------
106
107     Fold: 8          File: ./Experimento/breast/c4.5/it8/breast_C45.c45out
108
109               PREDICTED
110     ACTUAL     benign malignan    Class Error
111        benign       42        2          4.55%
112      malignan        1       23          4.17%
113
```

```
114         Error Rate for All Classes:           4.41%
115
116         ------------------------------------------------------------------------
117
118         Fold: 9          File: ./Experimento/breast/c4.5/it9/breast_C45.c45out
119
120                    PREDICTED
121         ACTUAL     benign malignan    Class Error
122            benign       44        0          0.00%
123          malignan        1       23          4.17%
124
125         Error Rate for All Classes:           1.47%
126
127         ------------------------------------------------------------------------
128
129         Fold: All (Total Confusion Matrix and Total Error Rates)
130
131                    PREDICTED
132         ACTUAL     benign malignan    Class Error
133            benign      430       14          3.15%
134          malignan       16      223          6.69%
135
136         Error Rate for All Classes:           4.39%
137
138         ------------------------------------------------------------------------
139
140         [ Sumary ]
141
142         Fold    benign  malignan   Overall
143
144            0      2.22%     8.33%     4.35%
145            1      2.22%     0.00%     1.45%
146            2      4.44%     4.17%     4.35%
147            3      4.44%     8.70%     5.88%
148            4      2.27%     8.33%     4.41%
149            5      4.55%     8.33%     5.88%
150            6      4.55%    16.67%     8.82%
151            7      2.27%     4.17%     2.94%
152            8      4.55%     4.17%     4.41%
153            9      0.00%     4.17%     1.47%
154
155         Mean      3.15%     6.70%     4.40%
```

```
156         SE        0.50%     1.42%     0.69%
```

B.3 Exemplo de Relatório com Testes Hipótese

```
1   ========================================================================
2   Sniffer v1.0                                          By Gustavo Batista
3   Discover Project                              LABIC - ICMC - USP - Brazil
4   gbatista@icmc.sc.usp.br              http://www.icmc.sc.usp.br/~gbatista
5   Date: 01/05/2003                                          Time: 15:03:22
6   ========================================================================
7                          HYPOTHESIS TEST REPORT
8   ========================================================================
9
10  *  Diference is statistically significant (95% confidence level)
11  ** Diference is highly significant (99% confidence level)
12
13  Positive results mean "top" algorithm has a lower error rate
14  Negative results mean "right side" algorithm has a lower error rate
15
16  Root: ./Experimento/breast
17  Performing 10-fold paired t-test
18
19  Comparing: /c4.5
20
21      benign  malignan   Overall
22        0.00     -0.95     -0.57       => With: /c4.5rules
23       -1.87      0.15     -0.59       => With: /cn2
24
25  Comparing: /c4.5rules
26
27      benign  malignan   Overall
28       -0.70      0.70      0.10       => With: /cn2
29
30  ------------------------------------------------------------------------
31
```

```
32        Root: ./Experimento/bupa
33        Performing 10-fold paired t-test
34
35        Comparing: /c4.5
36
37            benign  malignan   Overall
38             -0.21     -0.08     -0.22       => With: /c4.5rules
39              1.19     -1.90     -0.15       => With: /cn2
40
41        Comparing: /c4.5rules
42
43            benign  malignan   Overall
44              1.83     -1.35      0.21       => With: /cn2
45
46        ------------------------------------------------------------------------
47
48        Root: ./Experimento/cmc
49        Performing 10-fold paired t-test
50
51        Comparing: /c4.5
52
53            benign  malignan   Overall
54              1.85     -1.10     -2.64*      => With: /c4.5rules
55             -4.21**   -0.66      5.01**     => With: /cn2
56
57        Comparing: /c4.5rules
58
59            benign  malignan   Overall
60             -7.11**   -0.04     13.24**     => With: /cn2
61
62        ------------------------------------------------------------------------
63
64        Root: ./Experimento/pima
65        Performing 10-fold paired t-test
66
67        Comparing: /c4.5
68
69            benign  malignan   Overall
70              0.28     -0.38      0.06       => With: /c4.5rules
71             -6.44**    3.61**    0.45       => With: /cn2
72
73        Comparing: /c4.5rules
74
```

```
75              benign  malignan   Overall
76               -3.85**    4.85**    0.49       => With: /cn2
```

Referências Bibliográficas

Aha, D. W., Kibler, D. & Albert, M. (1991). Instance-based Learning Algorithms. *Machine Learning 6*, 37–66. 22, 97

Asker, L. & Boström, H. (1995). Building the DeNOx System: Experience from Real-World Application of Machine Learning. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*. 34

Axmark, D., Widenius, M. & DuBois, P. (2000). *MySQL 3.23 Reference Manual*. http://www.mysql.com/doc.html. 54, 73

Baranauskas, J. A. (2001). Extração Automática de Conhecimento por Múltiplos Indutores. Tese de Doutorado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08102001-112806. 24, 30, 36, 80

Baranauskas, J. A. & Batista, G. E. A. P. A. (2000). O Projeto DISCOVER: Idéias Iniciais. Comunicação pessoal. 27

Baranauskas, J. A. & Monard, M. C. (1998). Experimental Feature Selection using the Wrapper Approach. In *International Conference on Data Mining*, Rio de Janeiro, RJ, pp. 161–170. http://www.fmrp.usp.br/~augusto/. 45

Baranauskas, J. A. & Monard, M. C. (1999). The $\mathcal{MLC}$++ Wrapper for Feature Subset Selection using Decision Tree, Production Rule, Instance-based and Statistical Inducers: Some Experimental Results. Technical Report 87, ICMC-USP, São Carlos, SP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_87.ps.zip. 45

Baranauskas, J. A. & Monard, M. C. (2000a). An Unified Overview of Six Supervised Symbolic Machine Learning Inducers. Technical Report 103, ICMC-USP, São Carlos, SP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_103.ps.zip. 30

Baranauskas, J. A. & Monard, M. C. (2000b). Reviewing some Machine Learning Concepts and Methods. Technical Report 102, ICMC-USP, São Carlos, SP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_102.ps.zip. 30

Baranauskas, J. A. & Monard, M. C. (2003). Combining Symbolic Classifiers from Multiple Inducers. *Knowledge Based Systems 16*(3), 129–136. Elsevier Science. 30

Baranauskas, J. A., Monard, M. C. & Horst, P. S. (1999). Evaluation of CN2 Induced Rules Using Feature Selection. In *Argentine Symposium on Artificial Intelligence (ASAI/JAIIO/SADIO)*, Buenos Aires, Argentina, pp. 141–154. http://www.fmrp.usp.br/~augusto/. 45

Barnett, V. & Lewis, T. (1994). *Outliers in Statistical Data*. New York, NY: John Wiley & Sons. 44

Batista, G. E. A. P. A. (1997). Um Ambiente de Avaliação de Algoritmos de Aprendizado de Máquina utilizando Exemplos. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19082002-234842. 55, 85

Batista, G. E. A. P. A., Carvalho, A. & Monard, M. C. (1999). Aplicando Seleção Unilateral em Conjuntos de Exemplos Desbalanceados: Resultados Iniciais. In *II Encontro Nacional de Inteligência Artificial – ENIA 99*, pp. 327–340. http://www.icmc.usp.br/~gbatista. 44, 164

Batista, G. E. A. P. A., Carvalho, A. & Monard, M. C. (2000). Applying One-sided Selection to Unbalanced Datasets. In O. Cairo, L. E. Sucar, & F. J. Cantu (Eds.), *Mexican International Conference on Artificial Intelligence 2000*, pp. 315–325. Springer-Verlag. Lecture Notes in Artificial Intelligence, Best Paper Award Winner, http://www.icmc.usp.br/~gbatista. 58, 150, 164

Batista, G. E. A. P. A. & Monard, M. C. (1998). Seleção Unilateral para Melhorar a Classificação de Conjuntos de Exemplos Desbalanceados. In *XIII Simpósio Brasilero de Inteligência Artificial - Student Section*. http://www.icmc.usp.br/~gbatista. 44, 164

Batista, G. E. A. P. A. & Monard, M. C. (2001a). A Study of K-Nearest Neighbour as a Model-Based Method to Treat Missing Data. In *Argentine Symposium on Artificial Intelligence*, pp. 1–9. http://www.icmc.usp.br/~gbatista. 163

Batista, G. E. A. P. A. & Monard, M. C. (2001b). Uma Proposta de Tratamento de Valores Desconhecidos utilizando o Algoritmo K-Vizinhos mais Próximos. In *V Simpósio Brasileiro de Automação Inteligente*. http://www.icmc.usp.br/~gbatista. 163

Batista, G. E. A. P. A. & Monard, M. C. (2002). A Study of K-Nearest Neighbour as an Imputation Method. In A. Abraham, J. R. del Solar, & M. Köppen (Eds.), *Soft Computing Systems: Design, Management and Applications*, Santiago, Chile, pp. 251–260. IOS Press. http://www.icmc.usp.br/~gbatista. 59, 163

Batista, G. E. A. P. A. & Monard, M. C. (2003a). An Analysis of Four Missing Data Treatment Methods for Supervised Learning. *Applied Artificial Intelligence 17*(5), 519–533. http://www.icmc.usp.br/~gbatista. 163

Batista, G. E. A. P. A. & Monard, M. C. (2003b). Descrição da Arquitetura e do Projeto do Ambiente Computacional Discover Learning Environment — DLE. Technical Report 187, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.pdf. 27, 29, 161

Batista, G. E. A. P. A. & Monard, M. C. (2003c). Experimental Comparison of k-Nearest Neighbour and Mean or Mode Imputation Methods with the Internal Strategies used by C4.5 and CN2 to Treat Missing Data. Technical Report 186, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_186.pdf. 58, 59, 68, 115, 119, 163

Batista, G. E. A. P. A. & Monard, M. C. (2003d). The Discover Object Library (DOL) User's Manual. Technical report, ICMC-USP. (in press). 29, 61, 161

Bernardini, F. C. (2002). Combinação de Classificadores Simbólicos para Melhorar o Poder Preditivo e Descritivo de *Ensembles*. Dissertação de Mestrado, ICMC-USP. 36, 80

Blake, C. & Merz, C. (1998). UCI Repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html. 1, 110, 155

Bloedorn, E. & Michalski, R. S. (1998). Data-Driven Construtive Induction. *IEEE Intelligent Systems 13*(2), 30–37. 46

Blum, A. L. & Langley, P. (1997). Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence 97*, 245–271. 44

Booch, G., Jacobson, I., Rumbaugh, J. & Rumbaugh, J. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley. 61

Boswell, T. (1990). Manual for NewId version 4.1. Technical Report TI/P2154/RAB/4/2.3, The Turing Institute. 50, 75, 78

Bratko, I. (1990). *Prolog Programming for Artificial Intelligence*. Addison-Wesley. 15, 177

Breiman, L. (1996). Bagging Predictors. *Machine Learning 24*, 123–140. 36, 80

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984). *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth & Books. 21, 22, 148, 165

Brodley, C. E. & Friedl, M. A. (1999). Identifying Mislabeled Training Data. *Journal of Artificial Intelligence Research 11*, 131–167. 44

Chan, P. K. & Stolfo, S. (1998a). Learning with Non-uniform Class and Cost Distributions: Effects and a Distributed Multi-Classifier Approach. In *KDD-98 Workshop on Distributed Data Mining*, pp. 1–9. 145

Chan, P. K. & Stolfo, S. J. (1998b). The Effects of Training Class Distributions on Performance Using Cost Models. Draft. 38

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Jounal of Artificial Intelligence Research 16*, 321–357. 150

Ciaccia, P., Patella, M. & Zezula, P. (1997). M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Internation Conference on Very Large Data Bases*, pp. 426–435. 22, 58, 71, 97, 105, 106, 162

Clark, P. & Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. In Y. Kodratoff (Ed.), *Fifth European Conference (EWSL 91)*, pp. 151–163. Springer-Verlag. 7, 50, 75, 78, 107

Clark, P. & Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning 3*(4), 261–283. 90

Cost, S. & Salzberg, S. (1993). A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. *Machine Learning 10*(1), 57–78. 103

Craven, M. W. & Shavlik, J. S. (1995). Extracting Comprehensible Concept Representations from Trained Neural Networks. In *IJCAI-95 Workshop on Machine Learning and Comprehensibility*, pp. 61–75. 37

Crawford, D. (1998). GnuPlot: An Interactive Plotting Program. `http://www.ucc.ie/gnuplot/gnuplot.html`. 77, 82

Dempster, A. P., Laird, N. M. & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm (with Discussion). *Journal of Royal Statistical Society B39*, 1–38. 94

Descartes, A. & Bunce, T. (2000). *Programmming the Perl DBI*. O'Reilly & Associates. 54

Devaney, M. & Ram, A. (1997). Situation Development in a Complex Real-World Domain. In R. Engels, B. Evans, J. Herrmann, & F. Verdenius (Eds.), *ICML-97 Workshop on Machine Learning Applications in the Real World Methodological Aspects and Implications*, pp. 43–47. 2

Dietterich, T. G. (1997a). Limitations on Inductive Learning (Extended Abstract). Technical report, Oregon State University. `ftp://ftp.cs.orst.edu/pub/tgd/papers`. 36, 75, 81, 161

Dietterich, T. G. (1997b). Statistical Tests for Comparing Supervised Classification Learning Algorithms. Technical report, Oregon State University. `ftp://ftp.cs.orst.edu/pub/tgd/papers/stats.ps.gz`. 51, 82, 85, 121

Djeraba, C. (2003). *Multimedia Mining: A Highway to Intelligent Multimedia Documents*, Volume 22 of *Multimedia Systems and Applications*. Kluwer Academic Publishers. 63

Domingos, P. (1999). MetaCost: A General Method for Making Classifiers Cost-Sensitive. In *Knowledge Discovery and Data Mining*, pp. 155–164. 148

Dörre, J., Gerstl, P. & Seiffert, R. (1999). Text Mining: Finding Nuggets in Mountains of Textual Data. In *Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, pp. 398–401. 63, 175

Dosualdo, D. G. (2002). Investigação de Regressão para *Data Mining*. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 28, 87, 161

Dougherty, J., Kohavi, R. & Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features. In A. Priedits & S. Russell (Eds.), *XII International Conference in Machine Learning*, San Francisco, CA, pp. 194–202. Morgan Kaufmann. 47

Drummond, C. & Holte, R. C. (2000). Exploiting the Cost (In)sensitivity of Decision Tree Splitting Criteria. In *XVII International Conference on Machine Learning (ICML'2000)*, pp. 239–246. 154, 157

Elkan, C. (2001). The Foundations of Cost-Sensitive Learning. In *Seventeenth International Joint Conference on Artificial Intelligence*, pp. 973–978. 148, 154, 165

Evans, B. & Fisher, D. (1994). Overcoming Process Delays with Decision Tree Induction. *IEEE Expert 9*, 60–66. 37

Famili, A., Shen, W.-M., Weber, R. & Simoudis, E. (1997). Data Preprocessing and Intelligent Data Analysis. *Intelligent Data Analysis 1*(1). 51

Fawcett, T. & Provost, F. J. (1997). Adaptive Fraud Detection. *Data Mining and Knowledge Discovery 1*(3), 291–316. 141, 143

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996). Knowledge Discovery and Data Mining: Towards a Unifying Framework. In *Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pp. 82–88. 32, 33

Fischer, S., Klinkenberg, R., Mierswa, I. & Ritthoff, O. (2002). Yale: Yet Another Learning Environment — Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund. 26

Flannery, R. M. (2000). *Informix Handbook*. Prentice Hall. 54

Freedman, D., Pisani, R. & Purves, R. (1998). *Statistics* (3 ed.). W. W. Norton & Company, Inc. 91

Fürnkranz, J. (1999). Separate-and-Conquer Rule Learning. *Artificial Intelligence Review 13*(1), 3–54. 17, 22

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns: Elements of Resusable Object-Oriented Software*. Addison Wesley. 6, 50, 56, 60, 61, 62

Garner, S., Cunningham, S., Holmes, G., Nevill-Manning, C. & Witten, I. (1995). Applying a Machine Learning Workbench: Experience with Agricultural Datasets. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*, pp. 14–21. 2

Geromini, M. R. (2002). Projeto e Desenvolvimento da Interface Gráfica do Sistema DISCOVER. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 28

Gomes, A. K. (2002). Análise do Conhecimento Extraído de Classificadores Simbólicos utilizando Medidas de Avaliação e Interessabilidade. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-04072002-144610. 30

Gorman, R. P. & Sejnowski, T. J. (1988). Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. *Neural Networks 1*, 75–89. 111

Grzymala-Busse, J. W. & Hu, M. (2000). A Comparison of Several Approaches to Missing Attribute Values in Data Mining. In *RSCTC'2000*, pp. 340–347. 108

Hand, D. J. (1997). *Construction and Assessment of Classification Rules*. John Wiley and Sons. 147

Hart, P. E. (1968). The Condensed Nearest Neighbor Rule. *IEEE Transactions on Information Theory IT-14*, 515–516. 152

Holte, C. R. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning 11*, 63–91. 155

Hopfield, J. J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *National Academy of Sciences of the U.S.A. 79*, 2554–2558. 24

Horstmann, C. S. & Cornell, G. (1997). *Core Java*, Volume 1. Prentice Hall. 26, 74

Imamura, C. Y. (2001). Pré-processamento para Extração de Conhecimento de Bases Textuais. Dissertação de Mestrado, ICMC-USP. 30

Japkowicz, N. (2000). Learning from imbalanced data sets: a comparison of various strategies. In *AAAI Workshop on Learning from Imbalanced Data Sets*, Menlo Park, CA. AAAI Press. 154

Japkowicz, N. & Stephen, S. (2002). The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analysis 6*(5), 429–449. 143

John, G., Kohavi, R. & Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. In M. Kaufmann (Ed.), *Proceedings of the Eleventh International Conference on Machine Learning*, San Francisco, CA, pp. 167–173. 44

John, G. H. (1995). Robust Decision Trees: Removing Outliers from Databases. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*, pp. 174–179. 44

Jr., C. T., Traina, A., Seeger, B. & Faloutsos, C. (2000). Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In *Conference on Extending Database Technology – EDBT'2000*, pp. 51–65. 22, 71, 107

Kemp, A. H., Batista, G. E. A. P. A. & Monard, M. C. (2001). Descrição da Implementação dos Métodos Estatísticos de *Resampling* do Ambiente Discover. Technical Report 143, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_143.ps.zip. 30, 161

Kemp, A. H., Batista, G. E. A. P. A. & Monard, M. C. (2002). Descrição da Implementação dos Filtros para Recuperação da Taxa de Erro dos Algoritmos de Aprendizado de Máquina usados no Ambiente Discover. Technical Report 175, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_175.pdf. 161

Kibler, D. & Langley, P. (1988). Machine Learning as an Experimental Science. *Machine Learning 3*(1), 5–8. 36, 75, 161

Kimball, R. (1996). *The Data Warehouse Toolkit*. John Wiley & Sons. 35

Koch, G. & Loney, K. (1997). *Oracle 8: The Complete Reference*. Oracle Press. 54

Kohavi, R. (1997). Wrappers for Feature Subset Selection. *Artificial Intelligence 97*, 273–324. 44, 45

Kohavi, R. & Kunz, C. (1997). Option Decision Trees with Majority Votes. In *XIV International Conference in Machine Learning*, San Francisco, CA, pp. 161–169. Morgan Kaufmann. 2

Kohavi, R. & Sahami, M. (1996). Error-Based and Entropy-Based Discretization of Continuous Features. In *Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, Portland, OR, pp. 114–119. American Association for Artificial Intelligence. 47, 104

Kohavi, R., Sommerfield, D. & Dougherty, J. (1994). *$\mathcal{MLC}$++: A Machine Learning Library in C++*. IEEE Computer Society Press. 26

Kohavi, R., Sommerfield, D. & Dougherty, J. (1997). Data Mining Using $\mathcal{MLC}$++: A Machine Learning Library in C++. *International Journal on Artificial Intelligence Tools 6*(4), 537–566. 26, 112, 170

Kopla, H. & Daly, P. (1999). *A Guide to LaTeX: Document Preparation for Beginners and Advanced Users* (3 ed.). Addison-Wesley. 77, 82

Kubat, M., Holte, R. & Matwin, S. (1998). Machine Learning for the Detection of Oil Spills in Satellite Radar Images. *Machine Learning 30*, 195–215. 143

Kubat, M. & Matwin, S. (1997). Addressing the Course of Imbalanced Training Sets: One-Sided Selection. In *XIV International Conference in Machine Learning*, San Francisco, CA, pp. 179–186. Morgan Kaufmann. 44, 58, 143, 150

Lakshminarayan, K., Harp, S. A. & Samad, T. (1999). Imputation of Missing Data in Industrial Databases. *Applied Intelligence 11*, 259–275. 2, 110

Lavrač, N., Flach, P. A. & Zupan, B. (1999). Rule Evaluation Measures: A Unifying View. In S. Džeroski & P. A. Flach (Eds.), *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, Volume 1634 of *Lecture Notes in Artificial Intelligence*, Bled, Slovenia, pp. 174–185. Springer-Verlag. 30

Lebowitz, M. (1985). Categorizing Numeric Information for Generalization. *Cognitive Science 9*(3), 285–308. 104

Lee, H. D. (2000). Seleção e Construção de Features Relavantes para o Aprendizado de Máquina. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-15032002-113112. 45, 170

Lee, H. D. & Monard, M. C. (2000). Applying Knowledge-Driven Constructive Induction: Some Experimental Results. Technical Report 101, Department of Computer Science – University of São Paulo, São Carlos, SP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_101.ps.zip. 30

Lee, H. D., Monard, M. C. & Baranauskas, J. A. (1999). Empirical Comparison of Wrapper and Filter Approaches for Feature Subset Selection. Technical Report 94, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_94.ps.zip. 45, 113, 115

Lidie, S. O. & Walsh, N. (2002). *Mastering Perl/Tk*. O'Reilly & Associates. 59

Ling, C. X. & Li, C. (1998). Data Mining for Direct Mining: Problems and Solutions. In *Forth International Conference on Knownledge Discovery and Data Mining*, pp. 73–79. 141, 143

Little, R. J. & Rubin, D. B. (2002). *Statistical Analysis with Missing Data* (2 ed.). New York: John Wiley and Sons. 90, 91, 93

Lorena, A. C., Batista, G. E. A. P. A., de Carvalho, A. C. P. L. F. & Monard, M. C. (2002a). Splice Junction Recognition using Machine Learning Techniques. In *Primeiro Workshop Brasileiro de Bioinformatica (WOB 2002)*, pp. 32–39. 87, 153, 161

Lorena, A. C., Batista, G. E. A. P. A., de Carvalho, A. C. P. L. F. & Monard, M. C. (2002b). The Influence of Noisy Patterns in the Performance of Learning Methods in the Splice Junction Recognition Problem. In *VII Brazilian Symposium on Neural Networks*, pp. 31–36. 87, 153, 161

Martins, C. A. (2001). Interpretação de *Clusters* em Aprendizado de Máquina. Monografia para o Exame de Qualificação de Doutorado, ICMC-USP. 28, 30

Martins, C. A., Monard, M. C. & Halembeck, G. (2002). Combining Clustering and Inductive Learning to Find and Interpret Patterns from Dataset. In *Proceedings ACIS International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA-02)*, Foz do Iguaçu, Brazil, pp. 90–95. 30

Masters, T. (1993). *Practical Neural Network Recipes in C++*. Morgan Kaufmann. 58

McCulloch, W. S. & Pitts, W. (1943). A Logical Calculus of the Ideas Immanet in Nervous Activity. *Bulletin of Mathematical Biophysics 5*, 115–133. 23

Melanda, E. (2002). Pós-processamento de Conhecimento de Regras de Associação. Monografia para o Exame de Qualificação de Doutorado, ICMC-USP. 28, 30

Michalski, R. (1978). Pattern Recognition as Knowledge-Guided Computer Induction. Technical Report 927, Department of Computer Science, University of Illinois, Urbana-Champaign, Ill. 46

Michalski, R. S., Carbonell, J. G. & Mitchell, T. M. (1983). *Machine Learning: An Artificial Intelligence Approach*. Tioga Publishing Company. 15

Michalski, R. S., Mozetic, I., Hong, J. & Lavrac, N. (1986). The Multi-purpose Incremental Learning System AQ15 and Its Testing Application to Three Medical Domains. In *Fifth Annual National Conference on Artificial Intelligence*, pp. 1041–1045. 22

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood. 36

Milaré, C. R. (2000). Extração de Conhecimento de Redes Neurais. Monografia para o Exame de Qualificação de Doutorado, ICMC-USP. 30

Milaré, C. R., Carvalho, A. C. P. L. F. & Monard, M. C. (2002). An Approach to Explain Neural Networks Using Symbolic Algorithms. *International Journal of Computational Intelligence and Applications – IJCIA 2*(4), 365–376. 30, 87, 161

Mill, J. S. (1943). *A System of Logic, Ratiocinative and Inductive: Being a Connected View of the Principles of Evidence, and Methods of Scientific Investigation*. J. W. Parker. 17

Minsky, M. L. & Papert, S. A. (1969). *Perceptrons*. MIT Press. 23

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. 12, 97

Monard, M. C. & Baranauskas, J. A. (2003a). *Conceitos sobre Aprendizado de Máquina* (1 ed.), Chapter 4, pp. 89–114. Volume 1 of Rezende (2003). 18

Monard, M. C. & Baranauskas, J. A. (2003b). *Indução de Regras e Árvores de Decisão* (1 ed.), Chapter 5, pp. 115–140. Volume 1 of Rezende (2003). 17

Monard, M. C. & Batista, G. E. A. P. A. (2002). Learning with Skewed Class Distribution. In J. M. Abe & J. I. da Silva Filho (Eds.), *Advances in Logic, Artificial Intelligence and Robotics*, São Paulo, SP, pp. 173–180. IOS Press. 30, 164

Monard, M. C., Batista, G. E. A. P. A., Kawamoto, S. & Pugliesi, J. B. (1997). Uma Introdução ao Aprendizado Simbólico de Máquina. ftp://labic.icmc.usp.br/didatico/PostScript/ML.ps. 12

Morgan, J. & Messenger, R. (1973). THAID: A Sequential Search Program for the Analysis of Nominal Scale Dependent Variables. Technical report, Institute for Social Research, University of Michigan. 21

Paula, M. F. (2003). Ambiente para Disponibilização de Conhecimento. Dissertaçao de Mestrado, ICMC-USP. 30

Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T. & Brunk, C. (1994). Reducing Misclassification Costs. In *XI International Conference in Machine Learning*, pp. 217–225. 143

Pednault, E. P. D., Rosen, B. K. & Apte, C. (2000). Handling Imbalanced Data Sets in Insurance Risk Modeling. Technical Report RC-21731, IBM Research Report. 141

Piatetsky-Shapiro, G. (1989). Knowledge Discovery in Real Databases: A Report on the UJCAI-89 Workshop. *Artificial Intelligence Magazine 11*(5), 68–70. 25

Pila, A. D. (2001). Seleção de Atributos Relevantes para Aprendizado de Máquina utilizando a Abordagem de Rough Sets. Dissertação de Mestrado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-13022002-153921/publico/dissertacao_ADP.pdf. 30

Pila, A. D. & Monard, M. C. (2002). Rules Induced by Symbolic Learning Algorithms Using Rough Sets Reducts for Selecting Features: An Empirical Comparison with Other Filters. In A. Zapico & J. M. Santos (Eds.), *Proceedings Argentine Symposium on Artificial Intelligence, ASAI'2002*, Santa Fe, Argentina, pp. 206–217. 30

Prati, R. C. (2003). O *Framework* de Integração do Sistema DISCOVER. Dissertação de Mestrado, ICMC-USP. 28

Prati, R. C., Baranauskas, J. A. & Monard, M. C. (2001a). Extração de Informações Padronizadas para a Avaliação de Regras Induzidas por Algoritmos de Aprendizado de Máquina Simbólico. Technical Report 145, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip. 28, 29, 80

Prati, R. C., Baranauskas, J. A. & Monard, M. C. (2001b). Uma Proposta de Unificação da Linguagem de Representação de Conceitos de Algoritmos de Aprendizado de Máquina Simbólicos. Technical Report 137, ICMC-USP. ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. 28, 29, 80, 86

Prati, R. C., Baranauskas, J. A. & Monard, M. C. (2002). Padronização da Sintaxe e Informações sobre Regras Induzidas a Partir de Algoritmos de Aprendizado de Máquina Simbólico. *Revista Eletrônica de Iniciação Científica 2*(3). http://www.sbc.org.br/reic/edicoes/2002e3. 29

Pressman, R. S. (1992). *Software Engeneering: a Practitioner's Approach* (3 ed.). McGraw-Hill. 35

Provost, F. & Danyluk, A. (1995). Learning from Bad Data. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*, pp. 27–33. 2, 34

Provost, F., Fawcett, T. & Kohavi, R. (1998). The Case Against Accuracy Estimation for Comparing Induction Algorithms. In *15th International Conference on Machine Learning*, pp. 445–453. Morgan Kaufmann, San Francisco, CA. 51, 82

Provost, F. J. & Fawcett, T. (1997). Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *Knowledge Discovery and Data Mining*, pp. 43–48. 146

Provost, F. J. & Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine Learning 42*(3), 203–231. 51, 82, 154

Pugliesi, J. B. (2001). O Pós-Processamento em Extração de Conhecimento de Bases de Dados. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 30, 87, 161

Pyle, D. (1999). *Data Preparation for Data Mining*. San Francisco, CA: Morgan Kaufmann. 38, 42, 48

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning 1*, 81–106. Reprinted in Shavlik and Dieterich (eds.) Readings in Machine Learning. 21, 50, 75, 78, 116

Quinlan, J. R. (1987a). Generating Production Rules from Decision Trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Italy, pp. 304–307. 21, 50, 75, 78

Quinlan, J. R. (1987b). Simplifying Decision Trees. *International Journal of Man-Machine Studies 27*, 221–234. 21

Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. CA: Morgan Kaufmann. 7, 50, 75, 78, 90, 107, 155, 171

Quinlan, J. R. (1996). Bagging, Boosting and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 725–730. American Association for Artificial Intelligence. 36, 80

Rankins, R., Garbus, J. R., Solomon, D. & McEwan, B. W. (1996). *Sybase SQL Server 11 Unleashed*. Sams. 54

Rathjens, D. (1996). *MineSet™ User's Guide*. Silicon Graphics, Inc. 116

Rezende, S. O. (2003). *Sistemas Inteligentes: Fundamentos e Aplicações*. Barueri, SP, Brasil: Editora Manole. 200, 202

Rezende, S. O., Pugliesi, J. B., Melanda, E. A. & Paula, M. F. (2003). *Mineração de Dados* (1 ed.), Chapter 12, pp. 307–336. Volume 1 of Rezende (2003). ISBN 85-204-1683-7. 36

Roddick, J. F. & Hornsby, K. (2000). *Temporal, Spatial, and Spatio-Temporal Data Mining: First International Workshop, Tsdm 2000*. Lecture Notes in Artificial Intelligence. Springer Verlag. 63

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psycological Review 65*, 386–40. 23

Rozante, T. A. A. (2003). Implantação do Reuso de Componentes no Processo de Desenvolvimento de Software. Tese de Mestrado, ICMC-USP. 28

Rumelhart, D. E. & McClelland, J. L. (1998). *Parallel Distributed Processing: Explorations in the Microstruture of Cognition*, Volume 1. MIT Press. 24

Russel, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2 ed.). Prentice Hall. 17

Saitta, L., Giordana, A. & Neri, F. (1995). What is Real World. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*, pp. 43–40. 2

Saitta, L. & Neri, F. (1998). Learning in the "Real World". *Machine Learning 30*, 133–163. 33

Salzberg, S. L. (1997). On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. `http://www.cs.jhu.edu/~salzberg/critique.ps`. 51, 81, 82, 85

Sanches, M. K. (2002). Rotulação Automática de Exemplos a Partir de poucos Exemplos Rotulados utilizando Aprendizado de Máquina Supervisionado e não Supervisionado. Monografia para o Exame de Qualificação de Mestrado, ICMC-USP. 87, 161

Schaffer, C. (1994). A Conservation Law for Generalization Performance. In W. W. Cohen & H. Hirsh (Eds.), *Eleventh International Conference on Machine Learning*, New Brunswick, New Jersey, pp. 259–265. Morgan Kaufmann. 36, 75, 161

Schwabacher, M., Hirsh, H. & Ellman, T. (1995). Inductive Learning from Engineering Design Optimization. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*, pp. 49–55. 34

Shalloway, A. & Trott, J. R. (2002). *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley. 6, 50, 56, 60, 61, 62

Shaw, M. J. & Gentry, J. A. (1990). Inductive Leaning for Risk Classification. *IEEE Expert: Intelligent Systems and Their Applications 5*(1), 47–53. 16

Shepard, D. (1968). A Two-Dimensional Interpolation Function for Irregularly Spaced Data. In *Proccedings of the 23rd National Conference of the ACM*, pp. 517–523. 101

Silberschatz, A., Korth, H. F. & Sudarchan, S. (1997). *Database System Concepts*. McGraw-Hill. 73, 74

Silberschatz, A. & Tuzhilin, A. (1995). On Subjective Measures of Interestingness in Knowledge Discovery. In D. Aha & P. Riddle (Eds.), *ICML Workshop on Applying Machine Learning in Pratice*, pp. 50–56. 37

Snyder, A. (1986). Encapsulation and Inheritance in Object-Oriented Languages. In *Object-Oriented Programming Systems, Languages, and Applications Conference*, pp. 38–45. ACM Press. 55

Soares, C. (2002). Is the UCI Repository useful for Data Mining? In *First International Workshop on Data Mining Lessons Learned (DMLL-2002)*. http://www.hpl.hp.com/personal/Tom_Fawcett/DMLL-2002/Proceedings.html. 2

Stanfill, C. & Waltz, D. (1986). Instance-based Learning Algorithms. *Communications of the ACM 12*, 1213–1228. 23, 68, 102, 103

Stolfo, S. J., Fan, D. W., Lee, W., Prodromidis, A. L. & Chan, P. K. (1997). Credit Card Fraud Detection Using Meta-Learning: Issues and Initial Results. In *AAAI-97 Workshop on AI Methods in Fraud and Risk Management*. 141

Stroustrup, B. (1997). *The C++ Programming Language*. Addison Wesley. 74

Teller, A. & Veloso, M. (1995). Program Evolution for Data Mining. *International Journal of Expert Systems 8*(3), 213–236. 24

Tomek, I. (1976). Two Modifications of CNN. *IEEE Transactions on Systems Man and Communications SMC-6*, 769–772. 44, 58

Uhlmann, J. K. (1991). Satisfying General Proximity/Similarity Queries with Metric Trees. *Information Processing Letters 40*, 175–179. 105

Wall, L., Christiansen, T. & Schwartz, R. L. (1996). *Programming Perl* (2 ed.). O'Reilly & Associates. 27, 74, 176

Weiss, G. M. & Provost, F. (2001). The Effect of Class Distribution on Classifier Learning: An Empirical Study. Technical Report ML-TR-44, Rutgers University, Department of Computer Science. 149, 150

Weiss, S. M. & Indurkhya, N. (1998). *Predictive Data Mining: A Practical Guide*. San Francisco, CA: Morgan Kaufmann. 117

Weiss, S. M. & Kulikowski, C. A. (1991). *Computer Systems that Learn*. San Mateo, CA: Morgan Kaufmann. 18, 55, 85

Wilson, D. R. & Martinez, T. R. (2000). Reduction Techniques for Exemplar-Based Learning Algorithms. *Machine Learning 38*(3), 257–286. 58, 68, 97, 104, 154, 162

Witten, I. H. & Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann. 26, 170

Wolpert, D. H. (1992). Stacked Generalization. *Neural Networks 5*, 241–259. 36, 80

Zheng, Z. & Low, B. T. (1999). Classifying Unseen Cases with Many Missing Values. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 370–374. 112