Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Descrição da Arquitetura e do Projeto do Ambiente Computacional Discover Learning Environment — DLE

Gustavo E.A.P.A. Batista
Maria Carolina Monard

Nº 187

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos

Fev./2003

Descrição da Arquitetura e do Projeto do Ambiente Computacional Discover Learning Environment — DLE*

Gustavo E.A.P.A. Batista and Maria Carolina Monard
Universidade de São Paulo, Campus de São Carlos - USP
Instituto de Ciências Matemáticas e de Computação - ICMC
Departamento de Ciência da Computação e Estatística - SCE
Laboratório de Inteligência Computacional - LABIC
Caixa Postal 668, CEP 13560-970 - São Carlos, SP, Brasil
{gbatista, mcmonard}@icmc.usp.br

1 de março de 2003

Resumo

Por anos a comunidade científica que pesquisa na área de Aprendizado de Máquina tem utilizado repositórios de dados, tal como o repositório UCI, para avaliar propostas de melhorias e novos sistemas de Aprendizado de Máquina. Embora a utilidade desses repositórios seja reconhecida por pesquisadores da área, os repositórios de dados normalmente apresentam dados previamente pré-processados. Recentemente, foi fundada uma nova área de pesquisa conhecida por Descoberta de Conhecimento em Bancos de Dados, 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, inconsistências, excesso de valores desconhecidos, classes não representativas, etc. Dessa forma, a área de Descoberta de Conhecimento em Bases de Dados tem uma necessidade urgente pelo desenvolvimento e avaliação de métodos de pré-processamento de dados, os quais não foram foco de pesquisa na área de Aprendizado de Máquina.

Este trabalho apresenta uma descrição do ambiente computacional DLE (*Discover Learning Environment*). O ambiente computacional DLE tem como principal objetivo prover um *framework* para que novas técnicas de pré-processamento de dados possam ser facilmente e rapidamente implementadas e avaliadas experimentalmente. O ambiente computacional DLE é composto pela biblioteca DOL (Discover Object Library) e pelo ambiente computacional Sniffer. A biblioteca DOL é uma biblioteca orientada a objeto baseada em padrões de projeto na qual novos métodos de pré-processamento de dados podem ser implementados. O ambiente computacional Sniffer é um ambiente de gerenciamento de avaliações e comparações experimentais de algoritmos de Aprendizado de Máquina.

Palavras-chave: Pré-processamento de dados, Mineração de Dados, Aprendizado de Máquina.

*Esta pesquisa foi realizada com financiamento da CAPES e FINEP.

Sumário

1 Introdução **1**

2 A Biblioteca de Classes Discover Object Library — DOL **3**

2.1 O Desenvolvimento da Biblioteca de Classes DOL 5

2.2 A Arquitetura da Biblioteca DOL 6

2.3 O Projeto de Biblioteca DOL 10

2.3.1 O Módulo Core 11

2.3.2 Os Módulos ResaplingkFoldCV e ResamplingStratKFoldCV 14

2.3.3 Os Módulos DistanceHEOM e DistanceHVDM 16

2.3.4 Os Módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD . . 16

2.3.5 Os Módulos kNNMTree, kNNLinear, MTreeRandom e MTreeMST 17

2.4 Outras Soluções para a Implementação da Biblioteca DOL 19

3 O Ambiente Computacional Sniffer **21**

3.1 O Funcionamento do Ambiente Computacional Sniffer 23

3.2 A Arquitetura do Ambiente Computacional Sniffer 27

3.3 O Projeto do Ambiente Computacional Sniffer 28

4 Conclusão **31**

A A Sintaxe Padrão para Conjuntos de Dados **31**

A.1 Uma Visão Geral dos Arquivos da Sintaxe Padrão 32

A.2 Os Tipos de Dado da Sintaxe DSX 34

A.2.1 O Tipo de Dado `Nominal` 35

A.2.2 O Tipo de Dado `Enumerated` 35

A.2.3 O Tipo de Dado `Integer` 35

A.2.4 O Tipo de Dado `Real` 36

A.2.5 O Tipo de Dado `String` 36

A.2.6 O Tipo de Dado `Date` 36

A.2.7 O Tipo de Dado `Time` 36

A.3 Atributos Virtuais 37

A.4 Declarações Estendidas 38

A.5 Gramática da Sintaxe DSX 38

B Relatórios do Ambiente SNIFFER **40**

B.1 Exemplo de Relatório Resumido 40

B.2 Exemplo de Relatório Detalhado 43

B.3 Exemplo de Relatório com Testes Hipótese 46

Lista de Figuras

1 Exemplo de interação entre módulos da biblioteca DOL. 9

2 Arquitetura do mecanismo de envio de mensagens da biblioteca DOL. 10

3 Diagrama de classes em UML do projeto do módulo Core. 13

4 Diagrama de classes em UML do projeto dos módulos ResaplingkFoldCV e ResamplingStratKFoldCV. 15

5 Diagrama de classes em UML do projeto dos módulos DistanceHEOM e DistanceHVDM. 16

6 Diagrama de classes em UML do projeto dos módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD. 18

7 Diagrama de classes em UML do projeto dos módulos kNNMTree, kNNLinear, MTreeRandom e MTreeMST. 19

8 Exemplo de experimento organizado em diretórios para o ambiente SNIFFER. . . 23

9 Arquitetura do ambiente computacional SNIFFER. 27

10 Projeto do módulo SearchandRun do ambiente computacional SNIFFER. 29

11 Projeto dos módulos Report e HypothesisTest do ambiente computacional SNIFFER. 30

Lista de Tabelas

1 Sistemas de aprendizado cujas sintaxes são suportadas atualmente pela biblioteca DOL 4

2 Os identificadores especiais para diretórios utilizados atualmente pelo ambiente SNIFFER. 24

3 Exemplo de arquivo de declaração de atributos: `voyage.names` 33

4 Exemplo de arquivo de declaração de dados: `voyage.data` 34

5 Exemplo de arquivo de declaração de atributos, `voyage.names`, com declaração de atributo virtual. 37

6 As línguas aceitas pela definição estendida `date_language`. 39

1 Introdução

Por anos a comunidade científica que pesquisa na área de Aprendizado de Máquina 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 de Máquina. Embora a utilidade desses repositórios seja reconhecida por 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 por *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 ruído, inconsistências, excesso de valores desconhecidos, classes não representativas (isto é, grande desproporção entre as classes), etc.

Portanto, a área de Descoberta de Conhecimento em Bases de Dados tem uma necessidade urgente pelo desenvolvimento e avaliação de métodos de pré-processamento de dados, os quais não foram foco de pesquisa na área de Aprendizado de Máquina. Este trabalho tem como principal objetivo ajudar a suprir essa necessidade.

Este trabalho apresenta uma descrição do ambiente computacional DLE (*Discover Learning Environment*). O ambiente computacional DLE tem como principal objetivo prover um *framework* para que novas técnicas de pré-processamento de dados possam ser facilmente e rapidamente implementadas e avaliadas experimentalmente. O ambiente computacional DLE é composto por dois módulos distintos:

A biblioteca de classes DOL **(*Discover Object Library*)**

A biblioteca DOL é uma biblioteca orientada a objeto baseada em *padrões de projeto*[3] (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 *resampling*, 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.

[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 Discovey from Data Bases*, a sigla em inglês KDD é utilizada neste trabalho por ser largamente difundida na comunidade.

[3] *Design patterns*.

Dessa forma, será possível responder à pergunta mais freqüente nas pesquisas na área de Aprendizado de Máquina:

"*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 Aprendizado de Máquina. 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 de Máquina tais como C4.5 (Quinlan 1988), C4.5 RULES (Quinlan 1987), 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 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 Program Interface*[4]) 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.

Este trabalho está organizado da seguinte forma: na Seção 2 são explicados a arquitetura, o projeto e os principais módulos da biblioteca DOL; na Seção 3 é discutido o funcionamento do ambiente computacional SNIFFER, além do seu projeto e sua arquitetura interna; por fim, na Seção 4 são apresentadas as conclusões deste trabalho.

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

2 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, o objetivo é não somente ser uma base para a criação de novos métodos, mas também ser 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 transformação T que transforma um conjunto de dados D em um novo conjunto de dados D', ou seja

$$D' = T(D) \tag{1}$$

tal que:

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

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 D' não é necessariamente a mesma de D. 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 de "novos" exemplos derivados de exemplos presentes no conjunto de dados D.

Com base nessa definição de pré-processamento de dados podemos entender que um método de pré-processamento de dados é um procedimento que tem como entrada um conjunto de dados D representado por meio de uma tabela atributo-valor[5], e tem como saída um conjunto de dados D' na mesma representação.

Nesse sentido, a biblioteca DOL, na sua forma mais fundamental, 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 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.

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

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 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.

Integração com diversos sistemas de Aprendizado de Máquina

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 através 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 as informações referentes a dados e atributos nas sintaxes dos sistemas de aprendizado listados na Tabela 1;

C4.5 e C4.5 Rules	Ripper	Weka
CN2 e NewId	RT	Cart
C5.0 e Cubist	M5	MineSet
SNNS[6]	SVM Torch	Trepan

Tabela 1: Sistemas de aprendizado cujas sintaxes são suportadas atualmente pela biblioteca DOL

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 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 *DBI (Database Independent Interface for Perl)*[7] (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 pré-processamento de dados;

Métodos de *Resampling*
Os métodos de *resampling* 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 *resampling* 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 *resampling* 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 *resampling*: o *k-fold cross-validation* e o *k-fold cross validation estratificado* (Batista 1997; Weiss & Kulikowski 1991).

2.1 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 trabalho. De uma forma geral, o desenvolvimento do projeto de biblioteca de classes DOL pode ser dividido em duas etapas:

Arquitetura monolítica

[7] http://dbi.perl.org. DBI é um módulo de interface para 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.

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 através de herança é muitas vezes chamado de *reuso caixa branca*, uma vez que os detalhes internos da classe pai ficaram 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 conhecido como reuso caixa preta), ora por derivação, dependendo da necessidade do usuário. Entretanto, mesmo quando 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 com a arquitetura monolítica previamente implementada.

2.2 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 7 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 um 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 *resampling*;

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 *resampling 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 método 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 Me-*

tric (HEOM) e o segundo a função de distância *Heterogeneous Value Difference Metric (HVDM)* (Wilson & Martinez 2000; Batista & Monard 2003b);

MTreeRandom e MTreeMST
: M-tree é uma estrutura de índice que organiza os exemplos de um conjunto de dados em um "espaço métrico" (Ciaccia, Patella & Zezula 1997). 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;

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 *Tomek links* (Tomek 1976). Os *Tomek links* podem ser úteis na identificação de ruído e exemplos de borda nos dados (Kubat & Matwin 1997; Batista, Carvalho & Monard 2000);

Imputation
: Esse módulo disponibiliza alguns métodos de *imputation* (Batista & Monard 2002; Batista & Monard 2003b) 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;

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. Um exemplo é o módulo que implementa o método de *resampling 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 *resampling* depende do módulo Shuffle, o qual embaralha o conjunto de dados. 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 1 na próxima página.

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

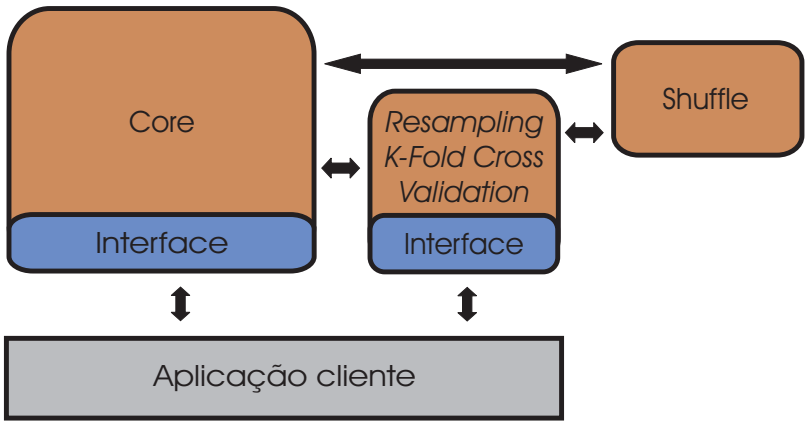

Figura 1: Exemplo de interação entre módulos da biblioteca DOL.

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.

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[8]. 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 nos valores de dados individuais, adição e remoção de atributos, adição ou remoção de exemplos, entre outras. Na Figura 2 na página seguinte é ilustrada a arquitetura do mecanismo de envio de mensagens da biblioteca DOL.

[8] 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.

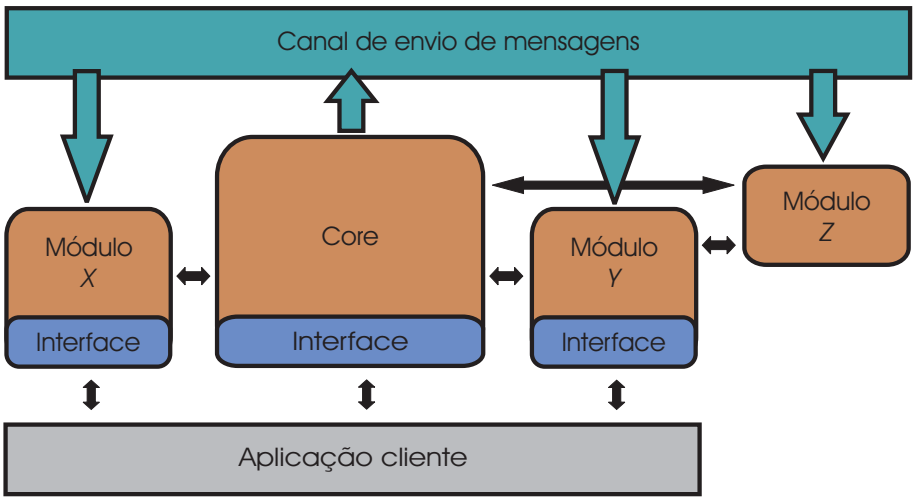

Figura 2: Arquitetura do mecanismo de envio de mensagens da biblioteca DOL.

2.3 O Projeto de 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 2003c).

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 existente 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 subclasses. 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 brevemente apresentados os projetos dos principais módulos da biblioteca DOL.

2.3.1 O Módulo Core

Como já mencionado, o módulo Core é o principal módulo da biblioteca de classes DOL. Esse módulo provê uma série de métodos capazes de fornecer ao usuário uma estrutura simples para acessar e manipular dados armazenados em uma tabela atributo-valor. Dados armazenados em um módulo Core podem ser armazenados em diversas sintaxes utilizadas pelos principais sistemas de aprendizado, ou podem ser carregados em bancos de dados relacionais. Como já foi informado anteriormente, 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 disponibilizado na interface do módulo Core;

2. O módulo deve ser capaz de suportar novos tipos de dados. Deve ser observado que os métodos de Mineração de Dados estão cada vez mais difundidos. Assim, é possível encontrar aplicações de Mineração de Dados em diversas áreas de conhecimento. Alguns pesquisadores têm trabalhado em adaptar os métodos de Mineração de Dados para diferentes domínios, dando origem a novas áreas de pesquisa como *Mineração de Dados em 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 dados, como por exemplo os tipos `blob`[9], coordenada espacial, entre outros tipos de dado;

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 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[10] (*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 de Máquina 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 3 na página oposta 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, como por exemplo, a classe `TypeReal` implementa o armazenamento e gerenciamento de atributos do tipo de dado `real` e a classe `TypeNominal` do tipo de dados `nominal`.

[9] *Binary Large Object*. São tipos de dados freqüentemente utilizados para armazenar vídeos, imagens e sons.
[10] 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 31.

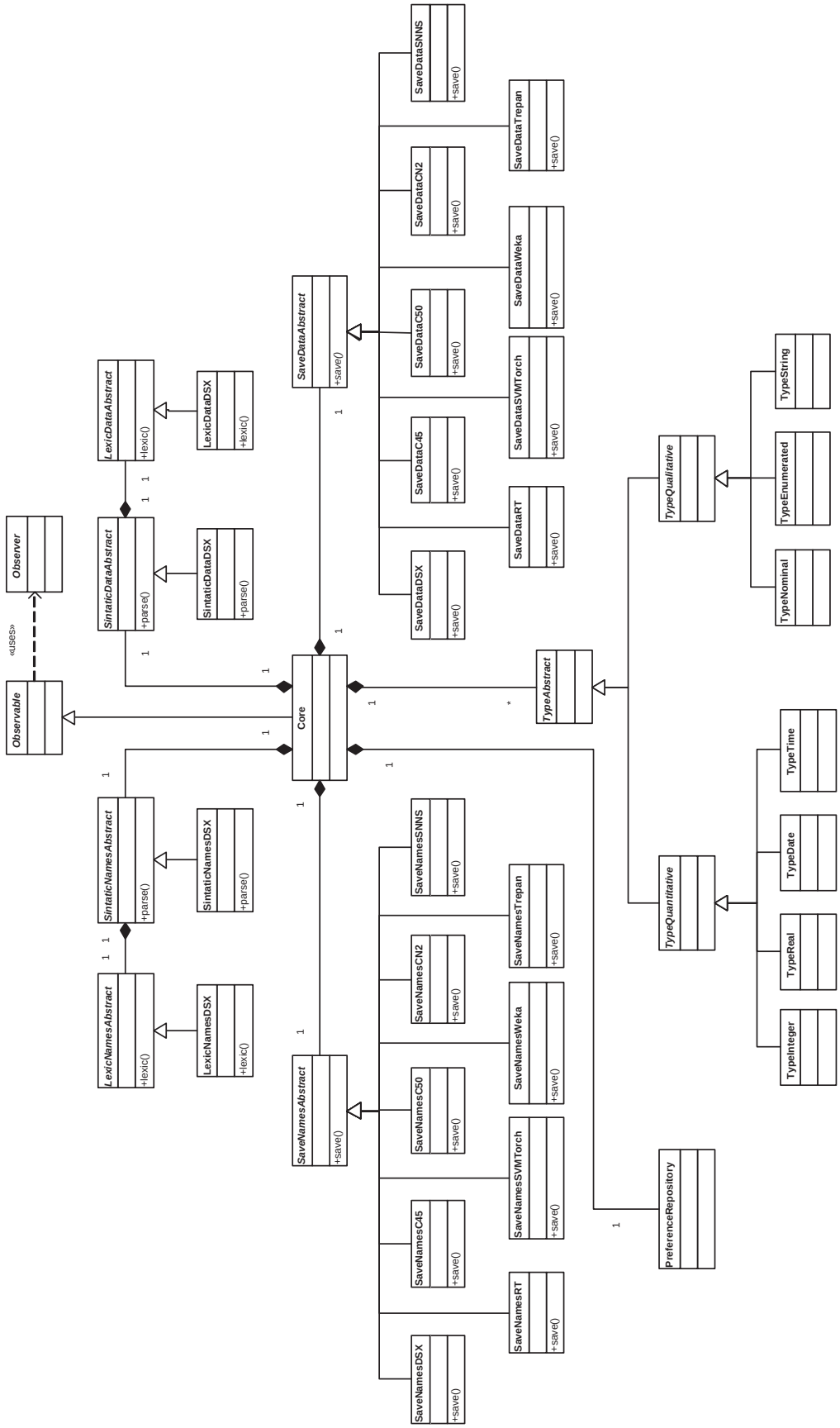

Figura 3: Diagrama de classes em UML do projeto do módulo Core.

A classe `Core` é a classe principal entre as classes apresentadas na Figura 3 na página precedente. 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**.

2.3.2 Os Módulos ResaplingkFoldCV e ResamplingStratKFoldCV

A classe abstrata `ResamplingAbstract` cria a estrutura básica para a implementação de métodos estatístico de *resampling*. Atualmente, dois métodos de *resampling* 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 na página oposta é mostrado o diagrama de classes em UML que ilustra essa interação entre as classes.

O método *k-fold cross-validation* é implementado dividindo o conjunto de dados em k partições mutuamente exclusivas $p_1, p_2, \ldots p_k$ e, logo após, são criados k pares de conjuntos de treinamento e teste da seguinte forma:

Para uma dada iteração i, $1 \leq i \leq k$

- O conjunto de teste, $c_{teste_i} = p_i$;
- O conjunto de treinamento correspondente, $c_{treinamento_i} = \bigcup_{j \neq i} p_j$.

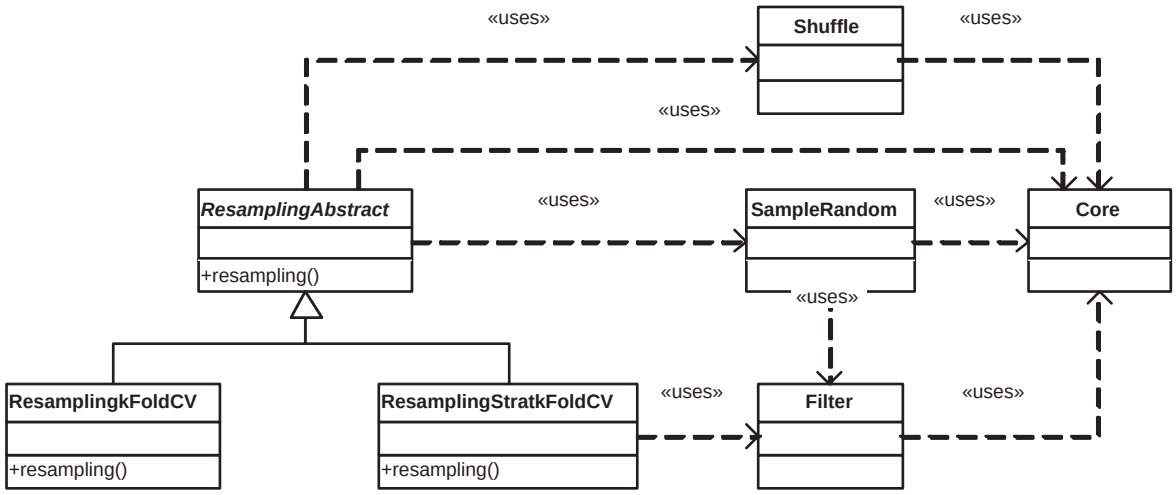

Figura 4: Diagrama de classes em UML do projeto dos módulos ResaplingkFoldCV e ResamplingStratKFoldCV.

O método *k-fold cross-validation* estratificado tem uma implementação mais complexa pois esse método 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:

Sejam $\vec{x_1}, \ldots \vec{x_n}$ os exemplos do conjunto de dados, e $f(\vec{x_i})$ o valor do atributo classe para o exemplo $\vec{x_i}$, e dado que o atributo classe pode assumir os valores do conjunto finito $\{v_1, \ldots v_s\}$

- Crie k partições vazias $p_1, \ldots p_k$;
- Para cada valor do atributo classe $v_l \in \{v_1, \ldots v_s\}$ faça
  - Filtre (remova temporariamente) todo exemplo j tal que $f(\vec{x_j}) \neq v_l$;
  - Particione os exemplos remanescentes em k partições $p'_1, \ldots p'_k$;
  - $p_i \leftarrow p_i \cup p'_i, \ \forall i \ 1 \leq i \leq k$;
  - Remova o filtro, tal que o conjunto de dados volte a possuir todos os exemplos originais.

A classe `ResamplingAbstract` e as sub-classes `ResamplingkFoldCV` e `ResamplingStratkFoldCV` seguem o padrão de projeto **Strategy** para a implementação das variações dos algoritmos de *resampling*. Ainda, a classe `ResamplingAbstract` utiliza do padrão de projeto **Template** para definir a estrutura básica do algoritmo de *resampling*, 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.

2.3.3 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)* (Wilson & Martinez 2000; Batista & Monard 2003b), repectivamente. 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 Figura 5 é mostrado um diagrama de classes UML que ilustra o projeto dos módulos DistanceHEOM e DistanceHVDM.

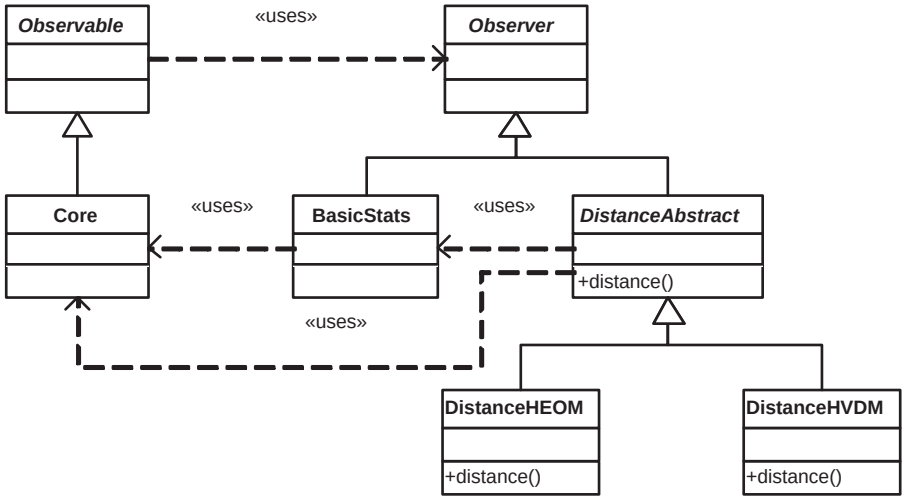

Figura 5: Diagrama de classes em UML do projeto dos módulos DistanceHEOM e DistanceHVDM.

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`.

2.3.4 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 através de um mapeamento linear simples, na qual os valores extremos do atributo a, denotados por max_a e min_a são mapeados em dois valores definidos pelo usuário max'_a e min'_a, sendo $max'_a \geq min'_a$.

Dado um valor v do atributo a, seu valor normalizado v' é dado pela Equação 2.

$$v' = r(v - min_a) + min'_a \tag{2}$$

na qual

$$r = \frac{max'_a - min'_a}{max_a - min_a} \tag{3}$$

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 $\hat{\mu}_a$ e $\hat{\sigma}_a$ a média e o desvio padrão estimados do atributo a, respectivamente. A normalização realizada pelo módulo NormalizaSimpleSD para um valor v do atributo a é definida pela Equação 4.

$$v' = \frac{v - \hat{\mu}_a}{\hat{\sigma}_a} \tag{4}$$

Se um atributo normalmente distribuído é normalizado conforme a Equação 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 2 e 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 5.

$$v' = r\left(\frac{v - \hat{\mu}_a}{\hat{\sigma}_a} - min_a\right) + min'_a \tag{5}$$

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 6 na página seguinte é ilustrada a interação entre as classes dos módulos de normalização.

2.3.5 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 `DistanceAbs-`

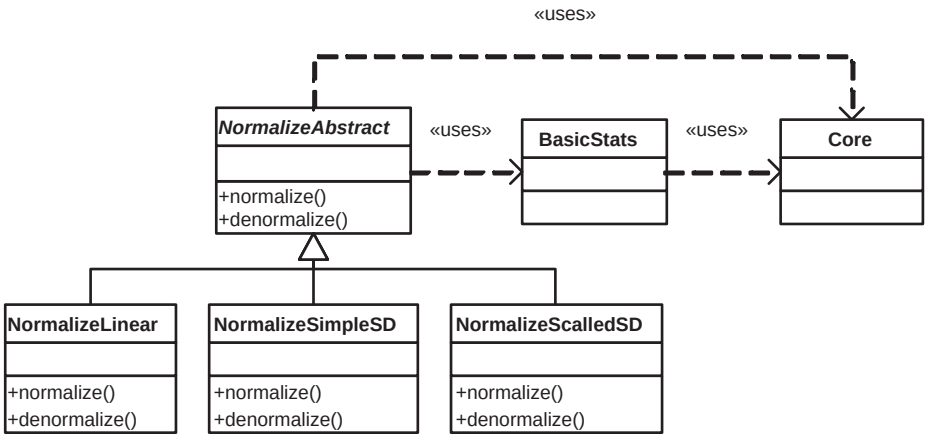

Figura 6: Diagrama de classes em UML do projeto dos módulos NormalizeLinear, NormalizeSimpleSD e NormalizeScalledSD.

`tract` 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-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.

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 o 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 7 na próxima página 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` 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` (esta relação não é mostrada no diagrama),

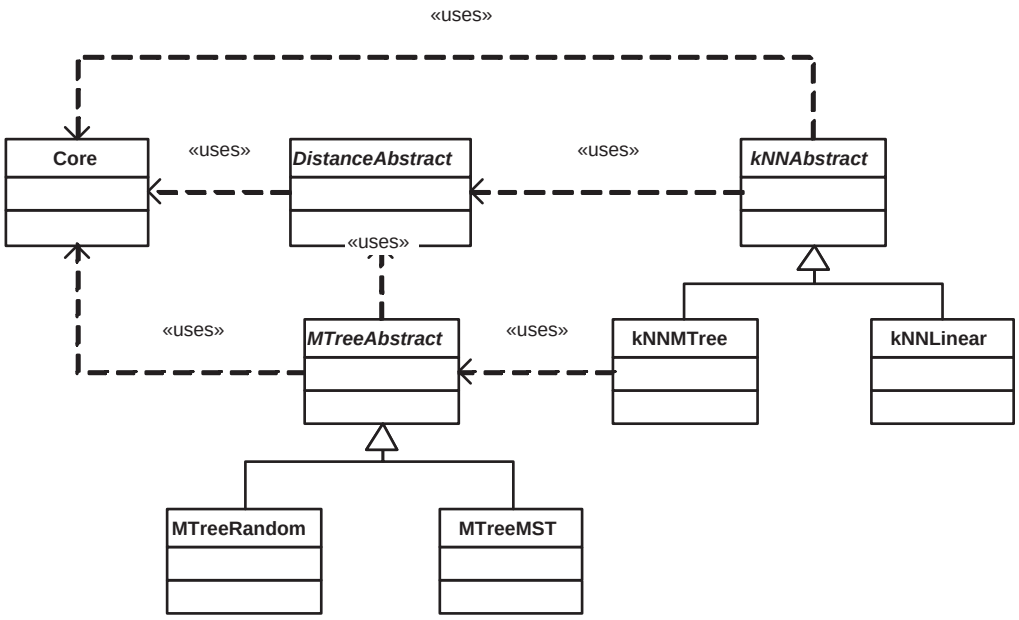

Figura 7: Diagrama de classes em UML do projeto dos módulos kNNMTree, kNNLinear, MTreeRandom e MTreeMST.

portanto essa classe utiliza o padrão de projeto **Observer** para observar as alterações no estado interno da classe `Core`.

2.4 Outras Soluções para a Implementação da Biblioteca DOL

Em Aprendizado de Máquina, a grande maioria dos sistemas de aprendizado é capaz de induzir padrões a partir de uma tabela atributo-valor. As tabelas atributo-valor são tabelas muito similares as 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 cuja performance é uma das preocupações centrais. Com a recente aplicação de sistemas de Aprendizado de Máquina 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 de Máquina. Nesse sentido, integrar métodos de pré-processamento de dados aos SGBDs pode obter proveito dessa tendência, uma vez que novos algoritmos de Aprendizado de Máquina poderão estar integrados aos SGBDs;

3. A grande maioria dos SGBDs relacionais utilizam a linguagem *SQL* (*Structured Query Language*) 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:

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 dos 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 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 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, realizado 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 de Máquina reside na integração com SGBDs. Esse é o principal fator que motiva a integração da biblioteca de classes DOL com SGBDs, como foi descrito nesta seção.

3 O Ambiente Computacional SNIFFER

O ambiente SNIFFER é um ambiente computacional para gerenciamento de avaliações e comparações experimentais de algoritmos de aprendizado. Análises experimentais são extremamente importantes na área de Aprendizado de Máquina porque os métodos empregados e os dados a serem analisados são, normalmente, muito completos 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 de Máquina tais como C4.5 (Quinlan 1988), C4.5 RULES (Quinlan 1987), ID3 (Quinlan 1986), CN2 (Clark & Boswell 1991), NEWID (Boswell 1990), C4.5 RULES (Quinlan 1988) e CN2 (Clark & Boswell 1991). 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, e permitem identificar 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, pois a repetição dos experimentos, como nos métodos de *resampling*, fornece ao usuário 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 o usuário tenha um compreensão melhor dos resultados, os métodos são executados em diversos conjuntos de exem-

plos. Dessa forma, é bastante comum que o usuário seja obrigado a executar um determinado método centenas de vezes.

Por exemplo, os experimentos realizados sobre tratamento de valores desconhecidos descritos em (Batista & Monard 2003b) 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 sistema 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, se torna 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 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 sistemas 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 métodos de *resampling* 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 *resampling* deve ser aplicado ao conjunto de dados;

Recuperação das taxas de erro
Os principais sistemas de aprendizado permitem que sejam fornecidos 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[11] 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 *resampling*, é necessário calcular estatísticas que fornecem um indicativo do desempenho do sistema de aprendizado no conjunto de dados analisado. O ambiente SNIFFER calcula medidas de desempenho para cada valor do atributo classe e, também, o desempenho geral levando em consideração todos os valores do atributo classe;

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 significativa 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

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

que os resultados publicados são fiéis aos resultados obtidos nos experimentos. Como, freqüentemente, os resultados dos experimentos envolvem uma grande quantidades de números, 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[12] (Crawford 1998), e tabelas no processador de textos LaTeX[13] (Kopla & Daly 1999).

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

3.1 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 8.

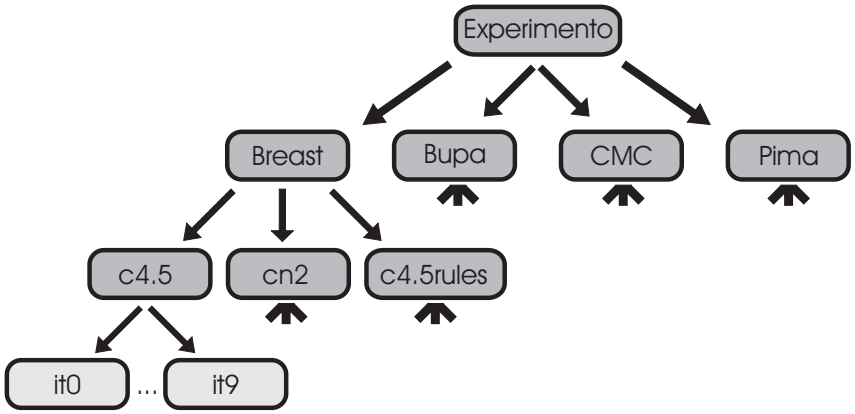

Figura 8: Exemplo de experimento organizado em diretórios para o ambiente SNIFFER.

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

[12] http://www.gnuplot.info.
[13] http://www.latex-project.org.

de dados foram criados outros três diretórios: `c4.5`, `cn2` e `c4.5rules` um para cada sistema de aprendizado a ser empregado. Na Figura 8 na página precedente 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 *resampling*, 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 1987)
`id3`	ID3 (Quinlan 1986)
`cn2`	CN2 (Clark & Boswell 1991)
`newid`	NEWID (Boswell 1990)

Tabela 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 2. Entretanto, o ambiente SNIFFER está preparado para acomodar outros sistemas de aprendizado como descrito na Seção 3.3 na página 28.

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 *resampling* 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 *resampling*, 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 *resampling* 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 *resampling*, 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 identifi-

cado pelo nome do diretório atual — Tabela 2 na página oposta. 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 *resampling*. 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 de Máquina 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; Prati, Baranauskas & Monard 2001a). 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 através de 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 1997).

Todos os resultados de desempenho obtidos com a aplicação de um método de *resampling* 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 8 na página 23.

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*[14] (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 através 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 *resampling*. Os erros médio e os desvios padrão são calculados para cada valor do atributo classe individualmente, e também para todos os valores do atributo classe em geral. Um exemplo de relatório gerado para o experimento ilustrado pela Figura 8 na página 23 é apresentado no Apêndice B, Seção B.1 na página 40;

`Detailed`
: Esse relatório é gerado para cada execução de um método de *resampling*. Ele apresenta, para cada iteração do método de *resampling*, 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 e erro média e do desvio padrão para todas as interações. No Apêndice B, Seção B.2 na página 43 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 *resampling* 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

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

significativo (com 95% de confiabilidade) ou altamente significativo (com 99% de confiabilidade). No Apêndice B, Seção B.3 na página 46 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).

3.2 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 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.

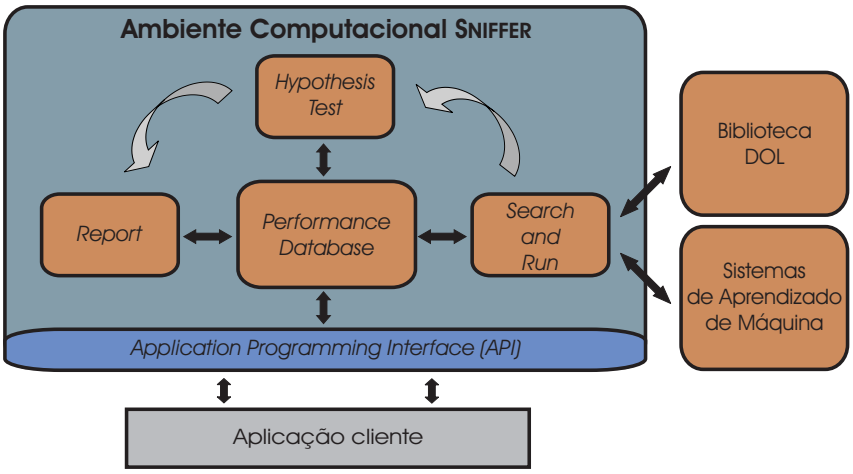

Figura 9: Arquitetura do ambiente computacional SNIFFER.

Na Figura 9 é apresentada uma representação gráfica da arquitetura do ambiente SNIFFER. Esse ambiente conta com quatro módulos principais, são eles:

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 de Máquina. 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 3.3. 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 de Máquina 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.

3.3 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 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 são:

Novos sistemas de aprendizado
: O usuário pode 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 sistema de aprendizado a ser adicionado;

Novos métodos de *resampling*
: Diferentes métodos de *resampling* 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 *resampling* nos conjuntos de dados. Para adicionar novos métodos de *resampling* ao ambiente SNIFFER basta que o método de *resampling* seja adicionado a biblioteca DOL;

Novos testes de hipótese

A comunidade de Aprendizado de Máquina 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 através da API que o ambiente proporciona aos usuários; ou através de criação de uma classe específica para esse fim.

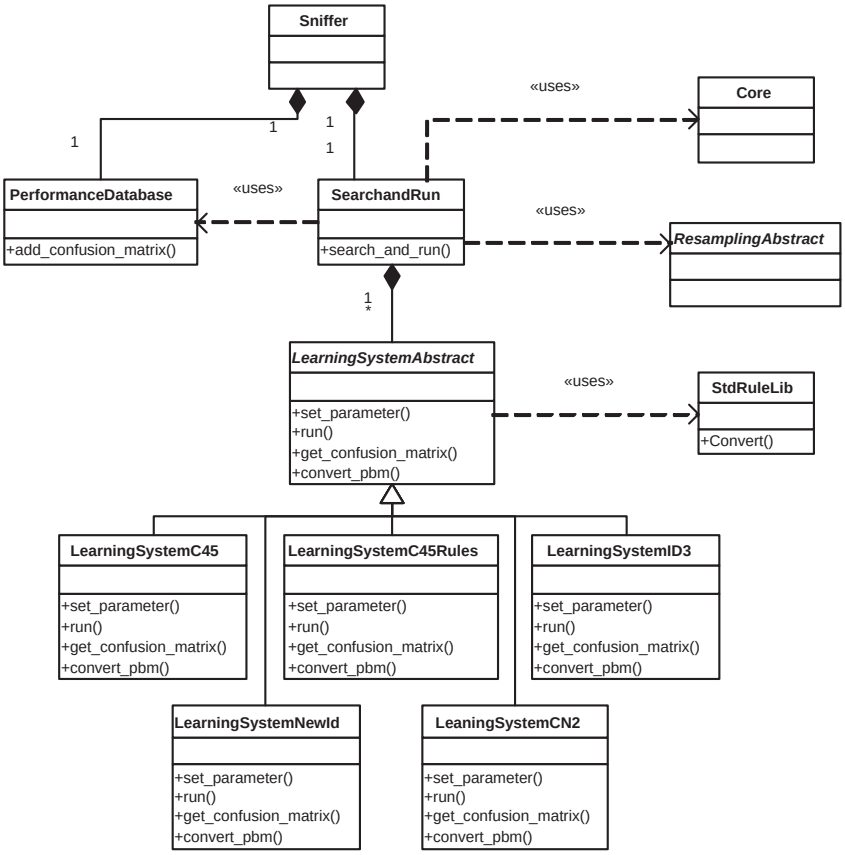

Figura 10: Projeto do módulo SearchandRun do ambiente computacional SNIFFER.

Na Figura 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 3.1 na página 23. 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 de Máquina. 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 (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.

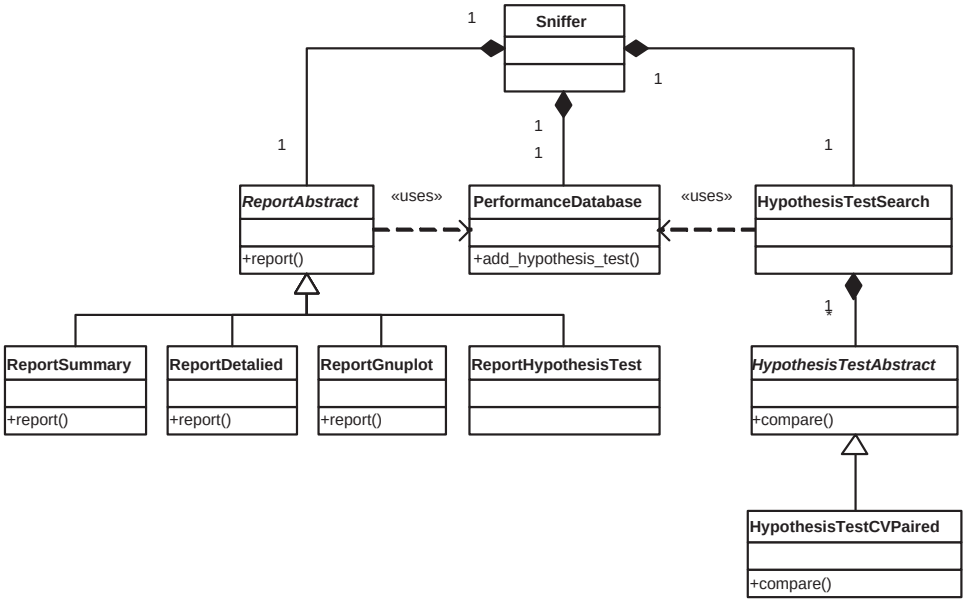

Figura 11: Projeto dos módulos Report e HypothesisTest do ambiente computacional SNIFFER.

Na Figura 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 de hipótese t pareado para *k-fold cross-validation*[15] 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**.

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

4 Conclusão

O objetivo deste trabalho 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 de Máquina SNIFFER.

A área de Aprendizado de Máquina 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. Essa sintaxe possui diversos recursos como o suporte à indução construtiva apoiada pelo usuário e o gerenciamento de diversos tipos de dados.

Finalmente, ressalva-se que o ambiente DLE foi utilizado para realizar diversos experimentos com algoritmos de Aprendizado de Máquina, as quais as medidas foram publicadas em diversos trabalhos (Batista & Monard 2002; Batista & Monard 2003b; Batista & Monard 2003a).

A A Sintaxe Padrão para Conjuntos de Dados

Graças a popularização da Mineração de Dados nos últimos anos, existem disponíveis diversos sistemas de aprendizado, tanto acadêmicos quanto comerciais. 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 outras tarefas, 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 1 na página 4.

Dessa forma, decidiu-se criar uma nova sintaxe para o ambiente DISCOVER, a qual foi dado o nome de DSX (*Discover Standard Sintax*), ao invés de utilizar alguma sintaxe já definida por outros pesquisadores, como por exemplo as sintaxes utilizadas nos projetos 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 dados tais como data e hora. Dessa forma, a nova sintaxe deve dar suporte aos tipos de dados mais atuais. Para a definição desses tipos, foram levantados os tipos de dados mais utilizados em sistemas gerenciadores de banco de dados, sendo que os tipos de dados 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 de 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 dados 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.1 Uma Visão Geral dos Arquivos da Sintaxe Padrão

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 3.

```
class.          | Class Attribute

| Attributes
outlook:       nominal (sunny, overcast, rain).
temperature:   integer.
humity:        integer.
windy:         nominal (yes, no).
class:         nominal (go, dont_go).
```

Tabela 3: 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 supervisionado, o atributo classe é mandatório, e esse atributo pode ser qualquer atributo presente no conjunto de dados. Neste 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. Desta 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 número de exemplos. A Tabela 4 na próxima página 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.

```
sunny, 25, 72, yes, go
sunny, 28, 91, yes, dont_go
overcast, 23, 90, yes, go
overcast, 29, 78, no, go
rain, 22, 95, no, go
rain, 19, 70, yes, dont_go
```

Tabela 4: Exemplo de arquivo de declaração de dados: `voyage.data`

A.2 Os Tipos de Dado da Sintaxe DSX

Como mencionado anteriormente, a sintaxe DSX deve suportar, além dos tipos de dados mais comuns (como os tipos numérico e nominal), tipos de dados 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 dados, como por exemplo os tipos data e hora. É importante que a sintaxe DSX ofereça suporte a diversos tipos de dados 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 oferece suporte aos tipos de dados mais sofisticados, é possível converter um conjunto de dados que utiliza esses tipos de dados mesmo para os sistemas de aprendizado que não suportam esses tipos. Isso pode ser feito através da definição de conversões padrão entre os tipos de dados mais complexos e os tipos de dados mais utilizados (numérico e nominal) pelos sistemas de aprendizado mais tradicionais.

Para definir quais tipos de dados devem ser suportados pela sintaxe DSX, foram analisados os principais tipos de dados suportados pelos sistemas gerenciadores de bancos de dados. Após essa análise foi decidido dar suporte, na sintaxe DSX, aos seguintes tipos de dados:

- `Nominal`;
- `Enumerated`;
- `Integer`;
- `Real`;
- `String`;
- `Date`;
- `Time`.

Nas próximas seções são descritos em mais detalhes cada um dos tipos de dados aceitos pela sintaxe DSX.

A.2.1 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. Neste 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.
AtribEx2: nominal (azul, amarelo, vermelho).
```

A.2.2 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.2.3 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.2.4 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.2.5 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 Dados sobre 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.2.6 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.4 na página 38) para informar datas em outros formatos. Um exemplo de declaração de um atributo do tipo de dado `date` é:

```
AtribEx7: date.
```

A.2.7 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.4) para informar horários em outros formatos. Um exemplo de declaração de um atributo do tipo de dado `time` é:

```
class.          | Class Attribute

| Attributes
outlook:          nominal (sunny, overcast, rain).
temperature:      integer.
humity:           integer.
windy:            nominal (yes, no).
temp_humity_rate: real    := $temperature/$humity.
class:            nominal (go, dont_go).
```

Tabela 5: Exemplo de arquivo de declaração de atributos, `voyage.names`, com declaração de atributo virtual.

```
AtribEx8: time.
```

A.3 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 5 mostra o arquivo de declaração de atributos, `voyage.names` (Tabela 3 na página 33) 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.

A.4 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 6 na próxima página. 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.

A.5 Gramática da Sintaxe DSX

A sintaxe DSX para arquivos de declaração de atributos `.names` pode ser mais formalmente definida pela seguinte gramática:

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 6: As línguas aceitas pela definição estendida `date_language`.

```
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 > := < expression > : < 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 (`< expression >`) pode ser qualquer expressão aritmética válida em Perl.

B Relatórios do Ambiente Sniffer

B.1 Exemplo de Relatório Resumido

```
==============================================================================
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: 15:03:10
==============================================================================
                              SUMMARY REPORT
==============================================================================

Base Directory: ./Experimento/breast/c4.5 Inducer:        C4.5

        benign  malignan   Overall

Mean      3.15%     6.70%     4.40%
SE        0.50%     1.42%     0.69%

------------------------------------------------------------------------------

Base Directory: ./Experimento/breast/c4.5rules Inducer:        C4.5Rules

        benign  malignan   Overall

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

Mean     38.46%    64.54%    50.45%    48.54%
SE        2.07%     2.98%     2.99%     1.61%

-----------------------------------------------------------------------------
```

```
Base Directory: ./Experimento/cmc/c4.5rules Inducer:        C4.5Rules

             1         2         3   Overall

Mean     42.29%    60.68%    38.93%    45.28%
SE        2.25%     3.40%     2.87%     1.89%

-----------------------------------------------------------------------------

Base Directory: ./Experimento/cmc/cn2 Inducer:        CN2

             1         2         3   Overall

Mean     23.85%    60.42%    74.37%    49.63%
SE        1.94%     4.21%     2.44%     1.60%

-----------------------------------------------------------------------------

Base Directory: ./Experimento/pima/c4.5 Inducer:        C4.5

             0         1   Overall

Mean     16.20%    43.29%    25.64%
SE        1.53%     3.04%     1.73%

-----------------------------------------------------------------------------

Base Directory: ./Experimento/pima/c4.5rules Inducer:        C4.5Rules

             0         1   Overall

Mean     17.00%    42.11%    25.78%
SE        1.72%     2.44%     1.10%

-----------------------------------------------------------------------------

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%

------------------------------------------------------------------------------

Fold: 1               File: ./Experimento/breast/c4.5/it1/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       44        1          2.22%
 malignan        0       24          0.00%

Error Rate for All Classes:           1.45%

------------------------------------------------------------------------------

Fold: 2               File: ./Experimento/breast/c4.5/it2/breast_C45.c45out

           PREDICTED
```

```
ACTUAL     benign malignan    Class Error
   benign       43        2          4.44%
 malignan        1       23          4.17%

Error Rate for All Classes:           4.35%

-----------------------------------------------------------------------------

Fold: 3               File: ./Experimento/breast/c4.5/it3/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       43        2          4.44%
 malignan        2       21          8.70%

Error Rate for All Classes:           5.88%

-----------------------------------------------------------------------------

Fold: 4               File: ./Experimento/breast/c4.5/it4/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       43        1          2.27%
 malignan        2       22          8.33%

Error Rate for All Classes:           4.41%

-----------------------------------------------------------------------------

Fold: 5               File: ./Experimento/breast/c4.5/it5/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       42        2          4.55%
 malignan        2       22          8.33%

Error Rate for All Classes:           5.88%

-----------------------------------------------------------------------------

Fold: 6               File: ./Experimento/breast/c4.5/it6/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       42        2          4.55%
```

```
 malignan        4       20         16.67%

Error Rate for All Classes:           8.82%

-----------------------------------------------------------------------------

Fold: 7               File: ./Experimento/breast/c4.5/it7/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       43        1          2.27%
 malignan        1       23          4.17%

Error Rate for All Classes:           2.94%

-----------------------------------------------------------------------------

Fold: 8               File: ./Experimento/breast/c4.5/it8/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       42        2          4.55%
 malignan        1       23          4.17%

Error Rate for All Classes:           4.41%

-----------------------------------------------------------------------------

Fold: 9               File: ./Experimento/breast/c4.5/it9/breast_C45.c45out

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign       44        0          0.00%
 malignan        1       23          4.17%

Error Rate for All Classes:           1.47%

-----------------------------------------------------------------------------

Fold: All (Total Confusion Matrix and Total Error Rates)

           PREDICTED
ACTUAL     benign malignan    Class Error
   benign      430       14          3.15%
 malignan       16      223          6.69%
```

```
Error Rate for All Classes:          4.39%

------------------------------------------------------------------------

[ Sumary ]

Fold    benign  malignan   Overall

   0      2.22%     8.33%     4.35%
   1      2.22%     0.00%     1.45%
   2      4.44%     4.17%     4.35%
   3      4.44%     8.70%     5.88%
   4      2.27%     8.33%     4.41%
   5      4.55%     8.33%     5.88%
   6      4.55%    16.67%     8.82%
   7      2.27%     4.17%     2.94%
   8      4.55%     4.17%     4.41%
   9      0.00%     4.17%     1.47%

Mean      3.15%     6.70%     4.40%
SE        0.50%     1.42%     0.69%
```

B.3 Exemplo de Relatório com Testes Hipótese

```
==========================================================================
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: 15:03:22
==========================================================================
                           HYPOTHESIS TEST REPORT
==========================================================================

*  Diference is statistically significant (95% confidence level)
** Diference is highly significant (99% confidence level)

Positive results mean "top" algorithm has a lower error rate
Negative results mean "right side" algorithm has a lower error rate

Root: ./Experimento/breast
Performing 10-fold paired t-test

Comparing: /c4.5
```

```
    benign  malignan   Overall
      0.00     -0.95     -0.57       => With: /c4.5rules
     -1.87      0.15     -0.59       => With: /cn2

Comparing: /c4.5rules

    benign  malignan   Overall
     -0.70      0.70      0.10       => With: /cn2

---------------------------------------------------------------------------

Root: ./Experimento/bupa
Performing 10-fold paired t-test

Comparing: /c4.5

    benign  malignan   Overall
     -0.21     -0.08     -0.22       => With: /c4.5rules
      1.19     -1.90     -0.15       => With: /cn2

Comparing: /c4.5rules

    benign  malignan   Overall
      1.83     -1.35      0.21       => With: /cn2

---------------------------------------------------------------------------

Root: ./Experimento/cmc
Performing 10-fold paired t-test

Comparing: /c4.5

    benign  malignan   Overall
      1.85     -1.10     -2.64*      => With: /c4.5rules
     -4.21**   -0.66      5.01**     => With: /cn2

Comparing: /c4.5rules

    benign  malignan   Overall
     -7.11**   -0.04     13.24**     => With: /cn2

---------------------------------------------------------------------------

Root: ./Experimento/pima
Performing 10-fold paired t-test
```

```
Comparing: /c4.5

    benign  malignan   Overall
      0.28     -0.38      0.06       => With: /c4.5rules
    -6.44**    3.61**    0.45       => With: /cn2

Comparing: /c4.5rules

    benign  malignan   Overall
    -3.85**    4.85**    0.49       => With: /cn2

-----------------------------------------------------------------------------
```

Referências

Axmark, D., Widenius, M. & DuBois, P. (2000). *MySQL 3.23 Reference Manual*. http://www.mysql.com/doc.html. 5, 20

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. 25

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. 5, 29

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. Best Paper Award Winner. 8

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. 8, 31

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). (in press). 31

Batista, G. E. A. P. A. & Monard, M. C. (2003b). 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. 8, 16, 22, 31

Batista, G. E. A. P. A. & Monard, M. C. (2003c). The Discover Object Library (DOL) User's Manual. Technical report, ICMC-USP. (in press). 10

Bernardini, F. C. (1997). Combinação de Classificadores Simbólicos para Melhorar o Poder Preditivo e Descritivo de *Ensembles*. Dissertação de Mestrado, ICMC-USP. 25

Blake, C. & Merz, C. (1998). UCI Repository of machine learning databases. 1

Booch, G., Jacobson, I., Rumbaugh, J. & Rumbaugh, J. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley. 10

Boswell, T. (1990). Manual for NewId version 4.1. Technical Report TI/P2154/RAB/4/2.3, The Turing Institute. 2, 21, 24

Bratko, I. (1990). *Prolog Programming for Artificial Intelligence*. Addison-Wesley. 38

Breiman, L. (1996). Bagging Predictors. *Machine Learning 24*, 123–140. 25

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. 8, 18

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. 2, 21, 24

Crawford, D. (1998). GnuPlot: An Interactive Plotting Program. `http://www.ucc.ie/gnuplot/gnuplot.html`. 23, 27

Descartes, A. & Bunce, T. (2000). *Programmming the Perl DBI*. O'Reilly & Associates. 5

Dietterich, T. G. (1997a). Limitations on Inductive Learning (Extended Abstract). `ftp://ftp.cs.orst.edu/pub/tgd/papers`. 21, 26

Dietterich, T. G. (1997b). Statistical Tests for Comparing Supervised Classification Learning Algorithms. `ftp://ftp.cs.orst.edu/pub/tgd/papers/stats.ps.gz`. 2, 27, 29

Djeraba, C. (2003). *Multimedia Mining: A Highway to Intelligent Multimedia Documents*, Volume 22 of *Multimedia Systems and Applications*. Kluwer Academic Publishers. 11

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. 11, 36

Famili, A., Shen, W.-M., Weber, R. & Simoudis, E. (1997). Data Preprocessing and Intelligent Data Analysis. *Intelligent Data Analysis 1*(1). 3

Flannery, R. M. (2000). *Informix Handbook*. Prentice Hall. 5

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns: Elements of Resusable Object-Oriented Software*. Addison Wesley. 1, 6, 9, 10, 11

Horstmann, C. S. & Cornell, G. (1997). *Core Java*, Volume 1. Prentice Hall. 20

Jr., C. T., Traina, A., Seeger, B. & Faloutsos, C. (2000). Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In *EDBT'2000*, pp. 51–65. 18

Kibler, D. & Langley, P. (1988). Machine Learning as an Experimental Science. *Machine Learning 3*(1), 5–8. 21

Koch, G. & Loney, K. (1997). *Oracle 8: The Complete Reference*. Oracle Press. 5

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. 1

Kohavi, R., Sommerfield, D. & Dougherty, J. (1997). Data Mining Using MLC++: A Machine Learning Library in C++. *International Journal on Artificial Intelligence Tools 6*(4), 537–566. 31

Kopla, H. & Daly, P. (1999). *A Guide to LaTeX: Document Preparation for Beginners and Advanced Users* (3 ed.). Addison-Wesley. 23, 27

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. 8

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. 32

Lidie, S. O. & Walsh, N. (2002). *Mastering Perl/Tk*. O'Reilly & Associates. 8

Masters, T. (1993). *Practical Neural Network Recipes in C++*. Morgan Kaufmann. 7

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.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip. 25

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.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_137.ps.zip. 25, 30

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. 2, 27

Provost, F. J. & Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine Learning 42*(3), 203–231. 2, 27

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning 1*, 81–106. Reprinted in Shavlik and Dieterich (eds.) Readings in Machine Learning. 2, 21, 24

Quinlan, J. R. (1987). Generating Production Rules from Decision Trees. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Italy, pp. 304–307. 2, 21, 24

Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. CA: Morgan Kaufmann. 2, 21, 24, 33

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. 25

Rankins, R., Garbus, J. R., Solomon, D. & McEwan, B. W. (1996). *Sybase SQL Server 11 Unleashed*. Sams. 5

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. 11

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. 1

Salzberg, S. L. (1997). On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. http://www.cs.jhu.edu/~salzberg/critique.ps. 2, 26, 27, 29

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. 21

Shalloway, A. & Trott, J. R. (2002). *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley. 1, 6, 9, 10, 11

Silberschatz, A., Korth, H. F. & Sudarchan, S. (1997). *Database System Concepts*. McGraw-Hill. 20

Snyder, A. (1986). Encapsulation and Inheritance in Object-Oriented Languages. In *Object-Oriented Programming Systems, Languages, and Applications Conference*, pp. 38–45. ACM Press. 6

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`. 1

Stanfill, C. & Waltz, D. (1986). Instance-based Learning Algorithms. *Communications of the ACM 12*, 1213–1228. 16

Stroustrup, B. (1997). *The C++ Programming Language*. Addison Wesley. 20

Tomek, I. (1976). Two Modifications of CNN. *IEEE Transactions on Systems Man and Communications SMC-6*, 769–772. 8

Wall, L., Christiansen, T. & Schwartz, R. L. (1996). *Programming Perl* (2 ed.). O'Reilly & Associates. 20, 37

Weiss, S. M. & Kulikowski, C. A. (1991). *Computer Systems that Learn*. San Mateo, CA: Morgan Kaufmann. 5, 29

Wilson, D. R. & Martinez, T. R. (2000). Reduction Techniques for Exemplar-Based Learning Algorithms. *Machine Learning 38*(3), 257–286. 8, 16

Witten, I. H. & Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann. 31

Wolpert, D. H. (1992). Stacked Generalization. *Neural Networks 5*, 241–259. 25