UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

Projeto e Implementação do Sistema DiscoverGraphics para a Geração de Gráficos de Dados

Richardson Floriani Voltolini
Maria Carolina Monard

Nº 280

RELATÓRIOS TÉCNICOS

São Carlos – SP
Set./2006

Projeto e Implementação do Sistema DiscoverGraphics para a Geração de Gráficos de Dados*

Richardson Floriani Voltolini
Maria Carolina Monard

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
Laboratório de Inteligência Computacional
Caixa Postal 668, 13560-970 - São Carlos, SP, Brasil
e-mail: {rfv, mcmonard}@icmc.usp.br

Resumo

O pré-processamento de dados é uma das primeiras fases do processo de Mineração de Dados – MD – e busca, principalmente, proporcionar ao usuário do processo uma maior compreensão dos dados sendo utilizados e a preparação desses dados para as fases seguintes. Sendo o pré-processamento um passo inicial da MD, os resultados obtidos nele influenciam as demais fases, tendo um impacto significativo no resultado final da mineração. Além disso, uma vez que o processo de MD é interativo e, em cada uma de suas fases, decisões de quais tarefas e procedimentos devem ser considerados e essas decisões muitas vezes são relativas aos dados utilizados, a compreensão dos mesmos é de essencial importância para a obtenção de bons resultados. Uma das principais maneiras de auxiliar o usuário do processo de MD a aumentar sua compreensão dos dados é por meio da análise de diferentes tipos de gráficos que representam esses dados. Este trabalho busca contribuir com essa tarefa, por meio de uma proposta, projeto e implementação de um sistema computacional para geração de gráficos de dados, denominado DISCOVERGRAPHICS. Esse sistema apresenta facilidades para a utilização de métodos de geração de gráficos para dados que se encontram no formato atributo-valor, muito comum nessa área de MD. Além disso, outras ferramentas simples que auxiliam a realização de algumas outras tarefas de pré-processamento foram implementadas. Neste trabalho são apresentados detalhes do sistema DISCOVERGRAPHICS, uma descrição das ferramentas utilizadas para a sua construção, bem como as diferentes interfaces que o sistema proporciona aos usuários. Essas interfaces permitem a utilização do sistema por: usuários avançados, utilizando principalmente a interface de linha de comando que permite acesso a um maior número de recursos do sistema; usuários iniciantes, utilizando principalmente a interface WEB que permite acesso as funcionalidades de geração de gráfico por usuários com pouco ou nenhum conhecimento da área de computação; e, por outros sistemas, que podem compartilhar dados e requisitar a execução de funcionalidades do sistema DISCOVERGRAPHICS por meio de arquivos XML. O sistema DISCOVERGRAPHICS desenvolvido apresenta facilidades de expansão para a inserção de novos métodos de geração de gráficos e novas ferramentas e funcionalidades. Ele está inserido em um sistema computacional de maior porte, que vem sendo desenvolvido em nosso Laboratório de Inteligência Computacional –LABIC –, para realizar as tarefas envolvidas nas diferentes fases do processo de Mineração de Dados.

Palavras-Chave: Análise Gráfica de Dados, Mineração de Dados, Aprendizado de Máquina, Extração de Padrões.

*Trabalho realizado com auxílio da FAPESP – Processo n° 04/04885-8.

Sumário

Lista de Figuras

Lista de Tabelas

[1]Este documento foi elaborado com o LaTeXe a sua bibliografia gerada automaticamente com o BibTeX

1 Introdução

A área de extração de conhecimento em bases de dados é relativamente nova e busca suprir a incapacidade humana de gerar conhecimento a partir da grande quantidade de dados adquirida e armazenada com a utilização de ferramentas computacionais. Para a extração desse conhecimento pode ser utilizado o processo de Mineração de Dados – MD – o qual pode ser dividido basicamente em três fases, ilustradas na Figura 1:

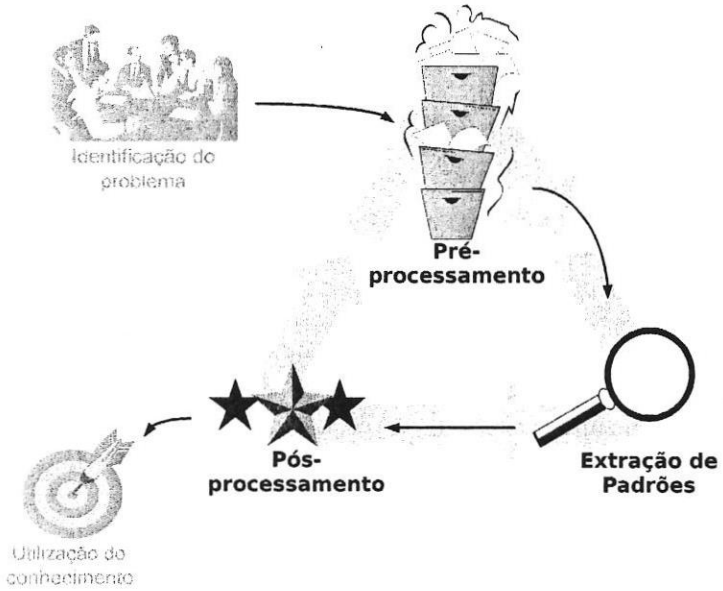

Figura 1: O processo de Mineração de Dados

1. **Pré-processamento:** essa fase possui dois objetivos principais, a compreensão dos dados a serem utilizados e a preparação dos mesmos para a fase seguinte – Extração de Padrões;
2. **Extração de Padrões:** nessa fase são extraídos os padrões implícitos nos dados (Weiss e Indurkhya, 1998). Para isso, devem ser escolhidos, entre outros, a linguagem de representação dos padrões e os algoritmos de extração de padrões. O resultado principal dessa fase é o conhecimento extraído dos dados;
3. **Pós-processamento:** nessa fase o conhecimento extraído na fase anterior deve ser avaliado e disponibilizado para que seja aplicado na resolução de problemas reais (Rezende et al., 2003).

A fase de pré-processamento de dados é de fundamental importância no processo de MD, e seu foco está relacionado à compreensão dos dados utilizados. Com o objetivo de auxiliar nessa tarefa, neste trabalho propomos e desenvolvemos o projeto e a implementação de um sistema computacional, denominado DISCOVERGRAPHICS, que implementa diversar representações gráficas de dados, tanto supervisionados quanto não-supervisionados, os quais encontram-se armazenados no formato atributo-valor. O sistema proposto será integrado futuramente a um sistema computacional de grande porte para extração de conhecimento em bases de dados, em desenvolvimento em nosso Laboratório de Inteligência Computacional – LABIC.

Este trabalho está organizado da seguinte maneira: na Seção 2 são apresentados maiores detalhes das funcionalidades do ambiente computacional DISCOVER e da DSX – *Discover Standard Syntax* – a sintaxe padrão para representação de base de dados utilizadas no ambiente DISCOVER e no DISCOVERGRAPHICS. Na Seção 3 são apresentadas algumas características gerais dos métodos de geração de gráficos, bem como uma breve descrição dos métodos implementados no DISCOVERGRAPHICS. Na Seção 4 são descritas as ferramentas e linguagens utilizadas na implementação do sistema DISCOVERGRAPHICS. Na Seção 5 são apresentados os detalhes da implementação realizada, maneiras de utilização do sistema e a interface WEB elaborada. E, por fim, na Seção 6 são apresentadas as conclusões e trabalhos futuros.

2 O Ambiente Discover

Na maioria dos projetos desenvolvidos pelos pesquisadores do Laboratório de Inteligência Computacional — LABIC — relacionados com aquisição automática de conhecimento e avaliação de conhecimento, inclusive no projeto no qual está inserido este trabalho, diversas tarefas como transformação de dados e formatos, execução de algoritmos de aprendizado, medições, entre outras, devem ser executadas diversas vezes. Muitas dessas funcionalidades podem ser realizadas de maneira semi-automática por sistemas comerciais. Porém, além da aquisição desses sistemas demandar um alto custo, esses sistemas fazem uso de algoritmos e ferramentas proprietários, o que dificulta sua utilização por pesquisadores que pretendem analisar e desenvolver novos algoritmos e ferramentas (Batista, 2003).

Assim, muitas vezes, alguns algoritmos e ferramentas são re-implementados por pesquisadores do próprio LABIC, ou por outros pesquisadores, que disponibilizam ferramentas de domínio público, tais como $\mathcal{MLC}$++ (Kohavi et al., 1994), WEKA (Witten e Frank, 2005) e YALE (Fischer et al., 2002). Entretanto, eventuais problemas decorrentes da utilização de algoritmos re-implementados, tais como um funcionamento diferente do proposto originalmente, fazem com que os algoritmos implementados pelos próprios idealizadores sejam preferidos para a realização de experimentos com o objetivo de avaliar diferentes algoritmos. Além disso, muitas ferramentas são implementadas para automatizar parcial ou integralmente algumas tarefas. Desse modo, não é incomum a ocorrência de sobreposições de implementações pela falta de comunicação e documentação do que já foi implementado pelos pesquisadores do laboratório. Ainda, também não é incomum a perda de implementações quando seus autores se desligam do laboratório ao final de seu projeto de pós-graduação.

Buscando resolver tais problemas, foi inicialmente proposto por Baranauskas e Batista (2000) um projeto denominado Projeto DISCOVER, que tem como principal objetivo integrar e padronizar os diversos projetos e ferramentas desenvolvidos pelos pesquisadores do LABIC relacionados com aquisição automática de conhecimento e avaliação de conhecimento. No início, o projeto DISCOVER consistiria apenas de um repositório de *scripts* para facilitar a configuração e execução de experimentos. Porém, essa solução mostrou-se pouco satisfatória. Assim, foi decidido criar um ambiente integrado no qual os *scripts* foram substituídos por bibliotecas de classes desenvolvidas em Perl (PERL, 1999). Deve ser ressaltado que todo o processo de implementação do projeto foi estudado segundo os fundamentos da Engenharia de Software (Rozante, 2003). Além disso, quanto a arquitetura do ambiente DISCOVER, foi proposto em Prati (2003) um *framework* para a integração dos componentes do ambiente DISCOVER utilizando *software patterns*.

O ambiente DISCOVER oferece vantagens em relação a outros sistemas com objetivos seme-

lhantes, pois permite a visão unificada que os formatos baseados em padrões proporcionam ao pesquisador (desenvolvedor) de novos componentes. Os padrões de representação foram sendo definidos por área. Em Prati et al. (2001a) é proposta uma sintaxe padrão para representação de conhecimento de diversos indutores simbólicos denominada $\mathcal{PBM}$ (Prati et al., 2002, 2001b). Para a representação de dados foi proposta uma sintaxe padrão (Batista, 2001), denominada DSX — DISCOVER *Standard Sintax*, a qual permite a utilização da biblioteca de classes DOL (Batista e Monard, 2006), para converter os arquivos de dados para a sintaxe utilizada por diversos sistemas de aprendizado simbólico, tais como C4.5, C4.5 Rules e CN2 entre outros. Mais especificamente, o ambiente DISCOVER Learning Environment — DLE — é composto pela biblioteca de classes DISCOVER Object Library — DOL — e pelo ambiente para gerenciamento de experimentos *Sniffer*, os quais foram desenvolvidos para diferentes tarefas de pré-processamento em mineração de dados (Batista, 2003; Batista e Monard, 2003, 2006). Novas funcionalidades foram e estão sendo especificadas, principalmente, para regras de regressão (Dosualdo, 2003; Pugliesi, 2004) e regras de associação (Melanda, 2004). Além disso, outros trabalhos foram ou estão sendo desenvolvidos na área de aprendizado de máquina relacional e proposicional (Prati, 2006; Ferro, 2006; Lee, 2005; Ferro, 2004; Milaré, 2003; Bernardini, 2006), aprendizado de máquina semi-supervisionado (Sanches, 2003; Matsubara, 2004), aprendizado não-supervisionado (Metz, 2006), mineração de dados (Paula, 2003; Pila, 2003; Chiara, 2003; Baranauskas, 2001) e mineração de textos (Martins, 2003; Matsubara, 2004; Matsubara et al., 2003).

A Sintaxe *DSX*

Neste trabalho é considerado que os dados encontram-se armazenados em uma tabela atributo-valor. Uma tabela atributo-valor apresenta em suas linhas os exemplos e em cada uma de suas colunas um atributo da base de dados. Na Tabela 1 é apresentado o formato geral de uma tabela atributo-valor com N exemplos E_i com $i = 1, ..., N$, na forma $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ para alguma função desconhecida $y = f(\mathbf{x})$. Os $\mathbf{x}_i$ são vetores da forma $(x_{i1}, x_{i2}, ..., x_{iM})$, com valores discretos ou contínuos. Assim, x_{ij} refere-se ao valor do atributo (ou *feature*) j, denominado $\mathbf{X}_j$, do exemplo E_i. Os valores y_i referem-se ao valor do atributo $\mathbf{Y}$, que representa a classe ou conceito-meta que descreve o fenômeno de interesse. Quando o valor do atributo Y é conhecido, os dados são denominados supervisionados, caso contrário, não-supervisionados.

	X_1	X_2	...	X_M	Y
E_1	x_{11}	x_{12}	⋮	x_{1M}	y_1
E_2	x_{21}	x_{22}	⋮	x_{2M}	y_2
⋮	⋮	⋮	⋱	⋮	⋮
E_N	x_{N1}	x_{N2}	⋮	x_{NM}	y_N

Tabela 1: Exemplos no formato atributo-valor

Na Tabela 2 é apresentado um exemplo de base de dados no formato atributo-valor. A base utilizada é a base de dados artificial *Voyage* (Quinlan, 1988), que será utilizada em diversos exemplos ilustrativos. Essa base possui $N = 6$ exemplos e $M = 4$ atributos (*outlook*, *temperature*, *humity* e *windy*). A função de classificação $y = f(\mathbf{x})$ pode assumir os valores *go* e *dont_go*.

Como mencionado, o principal objetivo do sistema DISCOVERGRAPHICS descrito neste trabalho é representar bases de dados armazenadas no formato atributo-valor em objetos gráficos,

	outlook	temperature	humity	windy	class
E_1	sunny	25	72	yes	go
E_2	sunny	28	91	yes	dont_go
E_3	overcast	23	90	yes	go
E_4	overcast	29	78	no	go
E_5	rain	22	95	no	go
E_6	rain	19	70	yes	dont_go

Tabela 2: Base de dados *Voyage* no formato atributo-valor

de maneira a facilitar a compreensão dos mesmos. O sistema que propomos para realizar essas representações gráficas, utiliza diversas funcionalidades do sistema DISCOVER, ao qual será futuramente integrado.

Embora a grande maioria dos sistemas de aprendizado utilize bases de dados no formato atributo-valor, cada um desses sistemas define uma sintaxe específica para os arquivos que armazenam suas bases. Uma vez que o DISCOVER abrange diferentes etapas do processo de MD e agrega vários sistemas de aprendizado, foi definida uma sintaxe padrão para representação de conjuntos de dados no formato atributo-valor. Essa sintaxe foi denominada DSX – DISCOVER *Standard Syntax*(Batista e Monard, 2003) e foi definida de maneira a ser mais abrangente e completa do que as utilizadas pela maioria dos sistemas de aprendizado. Isso, devido a necessidade da DSX abranger os recursos já disponibilizados por diferentes algoritmos, além de permitir expansões futuras que poderão se fazer necessárias com a implementação de outros módulos no projeto DISCOVER. A seguir é descrita, de maneira resumida, a sintaxe DSX, sendo que sua especificação completa pode ser encontrada em Batista e Monard (2003).

As bases de dados na sintaxe DSX são definidas por meio de dois arquivos texto, o arquivo de declaração de atributos com extensão *.names* e o arquivo de declaração de dados com extensão *.data*. Para uma mesma base, os dois arquivos devem ter o mesmo nome, somente se diferenciando pela extensão. Na Figura 2 é apresentado o exemplo de um arquivo de declaração de atributos (*voyage.names*) e na Figura 2 o de declaração de dados (*voyage.data*), ambos se referindo à base *Voyage* descrita na Tabela 2.

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

Figura 2: Exemplo de arquivo de declaração de atributos (*voyage.names*) na sintaxe DSX

```
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
```

Figura 3: Exemplo de arquivo de declaração de dados (*voyage.data*) na sintaxe DSX

No arquivo de declaração de atributos, a primeira linha deve conter o nome do atributo classe caso o aprendizado seja supervisionado. Em caso de aprendizado não-supervisionado o nome da classe deve ser substituído por "`null`".

É importante observar que todas as declarações em um arquivo de declaração de atributos terminam com "`.`" e comentários podem ser colocados em cada linha após a inserção do caractere "`|`". Esses comentários não são considerados pelos sistemas que utilizam esse arquivo.

As demais declarações, após o nome do atributo classe, referem-se aos atributos na mesma ordem que aparecem no arquivo de declaração de dados. Essas declarações devem estar no formato `nome_atributo : tipo_atributo.` onde `nome_atributo` é uma seqüência de caracteres iniciada por uma letra e seguida por zero ou mais caracteres alfa-numéricos ou "`_`". O elemento `tipo_atributo` representa o tipo do atributo, que pode ser:

nominal: utilizado para atributos que podem assumir um número restrito de valores (por exemplo cores). Um tipo nominal pode ser declarado apenas utilizando a palavra `nominal` ou pode vir seguido de uma lista dos valores que este atributo pode assumir, por exemplo, `nominal (verde, amarelo, vermelho)`. A segunda maneira de se declarar o atributo nominal é amplamente recomendada uma vez que o DISCOVER pode realizar verificações quanto a presença de erros nos dados. A primeira maneira pode ser utilizada quando o número de valores que o atributo pode assumir é muito elevado de maneira que o usuário não deseje preencher a lista de possíveis valores. Porém, considerando que é mais conveniente realizar a declaração dos valores nominais, neste trabalho foram implementadas ferramentas simples para auxiliar ao usuário na criação do arquivo de declaração de atributos na sintaxe DSX – ver Seção 5.1 – as quais também criam, automaticamente, a lista de valores nominais de atributos a partir do arquivo de declaração de dados.

enumerated: similar ao tipo `nominal`, porém existe uma ordem entre os valores do atributo. Por exemplo *pequeno*, *médio* e *grande*. Somente existe uma maneira de declarar esse tipo, que é utilizando a palavra `enumerated` seguido de uma lista dos possíveis valores em ordem crescente, por exemplo, `enumerated (crianca, jovem, adulto, idoso)`. Essa lista é obrigatória no tipo `enumerated` pois não é possível definir a ordem dos valores automaticamente;

integer: atributos que podem assumir valores inteiros. Sua declaração é `integer`;

real: atributos que podem assumir valores reais. Sua declaração é `real`;

string: atributos que podem assumir como valor cadeias de caracteres. Sua declaração é `string`;

date: atributos cujos valores são datas. Sua declaração é `date`;

time: atributos cujos valores são horários. Sua declaração é `time`.

No arquivo de declaração de dados, cada linha representa um exemplo distinto. Os valores dos atributos de cada exemplo devem aparecer na ordem definida no arquivo de declaração de atributos e devem ser separados por vírgula. Os valores de atributos do tipo `date` têm o formato *aaaa/mm/dd*, onde *aaaa* representa os quatro dígitos do ano, *mm* os dois dígitos do mês e *dd* os dois dígitos do dia. Os valores do tipo *string* devem aparecer entre aspas.

Com a mesma sintaxe do arquivo de declaração de dados, pode existir, opcionalmente, um arquivo de dados de teste. Esse arquivo, com a extensão *.test*, é utilizado pelo DISCOVER para

validação de modelos induzidos, quando o usuário não desejar que os mesmos dados utilizados para treinamento sejam utilizados para teste.

Neste trabalho, uma vez que o mesmo é parte integrante do projeto DISCOVER, as bases de dados a serem utilizadas devem se encontrar no formato definido pela sintaxe DSX. Caso a base a ser utilizada não esteja nesse formato, o DISCOVER possui métodos implementados que permitem a conversão de bases de dados em diversas sintaxes para a sintaxe DSX e vice-versa.

3 Métodos de Geração de Gráficos para Exploração de Dados

Existem diversos métodos de geração de gráficos os quais apresentam os dados de maneira diferentes, permitindo diversos modos de interação com o usuário. Esses métodos podem ser classificadas de diversas maneiras, baseando-se em diferentes critérios, tais como o objetivo da tarefa a ser executada, a estrutura da base de dados observada, a dimensão dos dados a serem exibidos, entre outros. Considerando o objetivo da tarefa a ser executada, os métodos de geração de gráficos podem ser classificados em (Grinstein e Ward, 2002):

Gráficos de exploração: Nessa tarefa, o usuário tem pouco ou nenhum conhecimento sobre os dados sendo analisados e, muitas vezes, não sabe exatamente o que está procurando. Sendo assim, torna-se necessário um ambiente dinâmico, no qual deve ser permitido ao usuário interagir com os gráficos gerados e alterar suas características, de modo a tornar visível padrões ou tendências dos dados.

Gráficos de confirmação: Nessa tarefa, o usuário tem uma hipótese a ser testada e utiliza algum tipo de gráfico para confirmar (ou não) essa hipótese. Nesse caso, o ambiente de geração de gráficos é mais estável e previsível, uma vez que o usuário sabe o que procura.

Gráficos de produção: Nessa tarefa, o usuário tem uma hipótese validada, sabe exatamente o que deve ser apresentado e busca melhores maneiras de representá-la graficamente, por exemplo, em campanhas de marketing.

Os métodos mais freqüentemente utilizados no processo de Mineração de Dados são os métodos de exploração, os quais são considerados neste trabalho. A seguir são descritos os métodos de exploração implementados no sistema DISCOVERGRAPHICS.

Para decidir quais métodos implementar, buscamos dar maior ênfase aos métodos que produzem representações gráficas de fácil interpretação, de modo que possam ser facilmente utilizados como ferramentas de análise de dados por usuários que não estejam familiarizados com essas técnicas e que busquem uma maneira rápida, porém eficiente, de aumentar a sua compreensão da base de dados utilizada. Além disso, nos focamos em métodos freqüentemente utilizados na documentação de trabalhos científicos, tais como artigos, dissertações e teses, com o objetivo de facilitar a publicação de resultados.

3.1 Gráfico de Dispersão

Os gráficos de dispersão (*scatterplots*) são gerados marcando-se pontos no espaço 2D ou 3D, no qual cada eixo do espaço representa um atributo da base de dados. Os pontos podem ainda ser coloridos de maneira a indicar o valor de um outro atributo, tal como a classe do exemplo.

Uma maneira comum de se visualizar esse tipo de gráfico é por meio de uma matriz de gráficos de dispersão (*scatterplot matrix*), a qual é construída gerando-se, em uma matriz, todos os gráficos de dispersão possíveis, combinando os atributos da base de dados dois a dois (no caso de um gráfico 2D) ou três a três (no caso de um gráfico 3D). Assumindo gráficos 2D, seja S uma matriz $M \times M$ de gráficos de dispersão, com elementos S_{ij} para i e $j = 1, 2, \ldots, M$, onde M é o número de atributos da base de dados. Cada S_{ij} é um gráfico no qual o eixo das abscissas representa o i-ésimo atributo, e o eixo das coordenadas o j-ésimo atributo. A Figura 4 mostra um exemplo de matriz de gráficos de dispersão para $M = 9$.

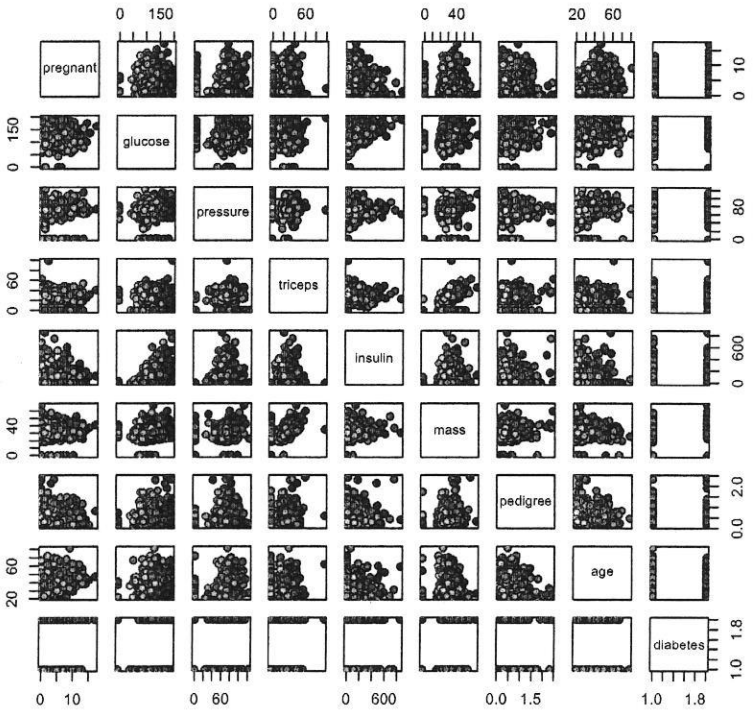

Figura 4: Exemplo de uma matriz de gráficos de dispersão para dados supervisionados

Um dos problemas de se criar uma matriz de gráficos de dispersão é que, em uma base de dados com M atributos, quando o espaço for 2D, serão gerados M^2 gráficos e quando for 3D, M^3. Isso torna inviável a visualização da matriz completa para bases de dados com elevado número de atributos, a qual necessitaria ser desmembrada em várias matrizes.

3.2 Histograma

Para a construção de um histograma, um atributo contínuo deve ser discretizado em uma determinada quantidade de intervalos do mesmo tamanho. Para cada um desses intervalos é realizado a contagem do número de exemplos que pertencem ao intervalo. Essa contagem é representada então por meio de barras. Se a base de dados utilizada para a construção do gráfico for supervisionada, o valor da classe pode ser representado por diferentes cores em cada

barra do histograma. A Figura 5 apresenta o histograma de um atributo de uma base que possui quatro valores para a classe.

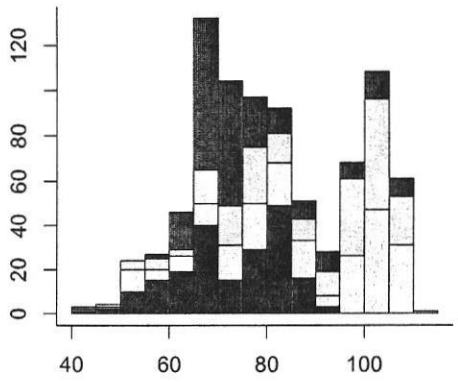

Figura 5: Exemplo de um histograma para dados supervisionados

3.3 Gráfico de Atributo Ordenado

Para a análise de um atributo individual, pode-se utilizar histogramas. Entretanto, para uma visão mais detalhada do mesmo, pode-se ordenar os exemplos da base com relação ao atributo a ser analisado e plotar um ponto para cada exemplo, *i.e.*, no eixo y o valor do referido atributo e no eixo x um número seqüencial que identifica cada exemplo. A Figura 6 mostra um exemplo deste tipo de gráfico para dados supervisionados, no qual cores diferentes identificam a classe do exemplo.

3.4 Gráfico de Coordenadas Paralelas

A representação de dados de alta dimensionalidade em um espaço bidimensional não é uma tarefa simples. Essa tarefa torna-se ainda mais complexa quando deseja-se tornar esses gráficos simples de serem interpretados pelo usuário. Um dos métodos de geração de gráficos que obtém relativo sucesso na resolução dessa tarefa é o gráfico de coordenadas paralelas. Esse gráfico é gerado criando-se barras paralelas eqüidistantes que representam eixos, um para cada atributo da base de dados. Os pontos extremos das barras indicam o valor máximo e mínimo daquele atributo na base. Os exemplos são representados por linhas que cruzam todas as barras no ponto respectivo ao valor do atributo representado pela barra. As linhas podem ainda ser coloridas no caso de exemplos supervisionados. Um exemplo de gráfico de coordenadas paralelas para uma base de dados que possui três valores para classe é mostrado na Figura 7 na página 10.

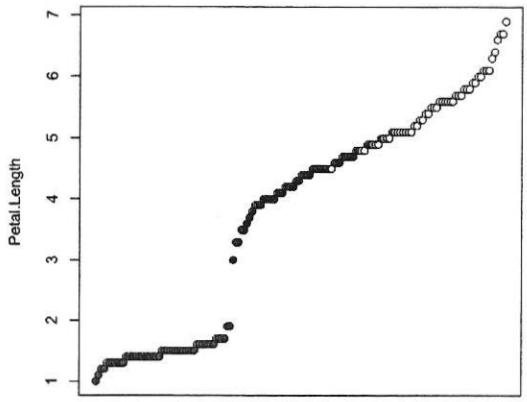

Figura 6: Exemplo de gráfico de atributo ordenado para dados supervisionados

4 Ferramentas Utilizadas

Para implementar o sistema DISCOVERGRAPHICS, foram utilizadas ferramentas computacionais de domínio público e linguagens para comunicação com outros sistemas. Cada uma dessas feramentas e linguagens são descritas brevemente a seguir, mostrando exemplos explicativos de sua utilização a fim de facilitar a sua compreensão.

4.1 A Ferramenta Computacional R

O R[1] (R Development Core Team, 2006) é um conjunto de ferramentas integradas, que provê um ambiente amplamente planejado e coerente para manipulação de dados, realização de cálculos e geração de gráficos. Esse ambiente possui: uma linguagem de programação (linguagem R) similar à linguagem S (Becker et al., 1988; Chambers e Hastie, 1992; Chambers, 1998); uma interface de linha de comando - CLI (*Command Line Interface*); um depurador; suporte a geração de gráficos em tela (no caso de estar sendo utilizado em um sistema operacional ou ambiente computacional que possua interface gráfica), entre outros. O R foi inicialmente desenvolvido por Ross Ihaka e Robert Gentleman no Departamento de Estatística da Universidade de Auckland, Nova Zelândia e, desde meados de 1997, possui um grupo de desenvolvedores que se dedicam a manutenção e aprimoramento do projeto. Algumas características do R incluem (Venables e Smith, 2004):

- facilidades para a manipulação e armazenamento de dados, tais como a realização de diferentes tipos de operações em matrizes e vetores;
- uma grande coleção de ferramentas para análise de dados, incluindo ferramentas gráficas para análise e exibição de dados;

[1] `http://www.r-project.org`

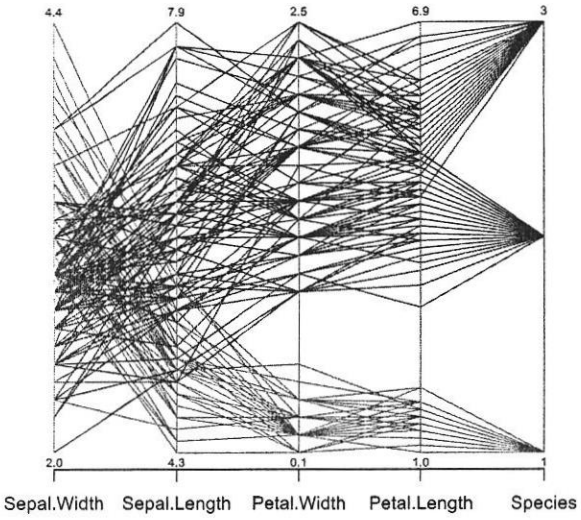

Figura 7: Exemplo de um gráfico de coordenadas paralelas para dados supervisionados

- linguagem de programação simples, efetiva e bem documentada;
- possui um grande número de pacotes (bibliotecas de funções, dados e documentação) implementados por terceiros e disponibilizado em um repositório bem organizado — CRAN[2] (*Comprehensive R Archive Network*).

A Figura 8 na próxima página apresenta o ambiente R e alguns exemplos de gráficos gerados por esse conjunto de ferramentas.

A seguir é apresentada uma breve introdução de algumas funcionalidades da linguagem R de maneira a facilitar a compreensão das próximas seções deste trabalho. Nos exemplos, para designar a linha de entrada de comandos na CLI do R utilizamos o símbolo ">". Também, é assumido que os exemplos representam uma única seção R, ou seja, são incrementais. Por exemplo, uma variável criada no primeiro exemplo poderá ser utilizada nos seguintes exemplos.

A linguagem R é sensível a caixa, ou seja, elementos com letra maiúscula e minúscula, tais como `var`, `Var`, `VAR` são diferentes. As variáveis não são tipadas e não é necessário que sejam declaradas explicitamente para serem utilizadas. Para atribuir um valor a uma variável, ou para alterar seu conteúdo, deve-se utilizar um sinal de menor e um sinal de negativo: "<-" ou um sinal de negativo e um de maior: "->", dependendo do lado que está a variável que recebe o valor. Por exemplo:

```
> A <- 5
> 1 -> a
> b <- a + A
```

[2] http://www.cran.r-project.org/

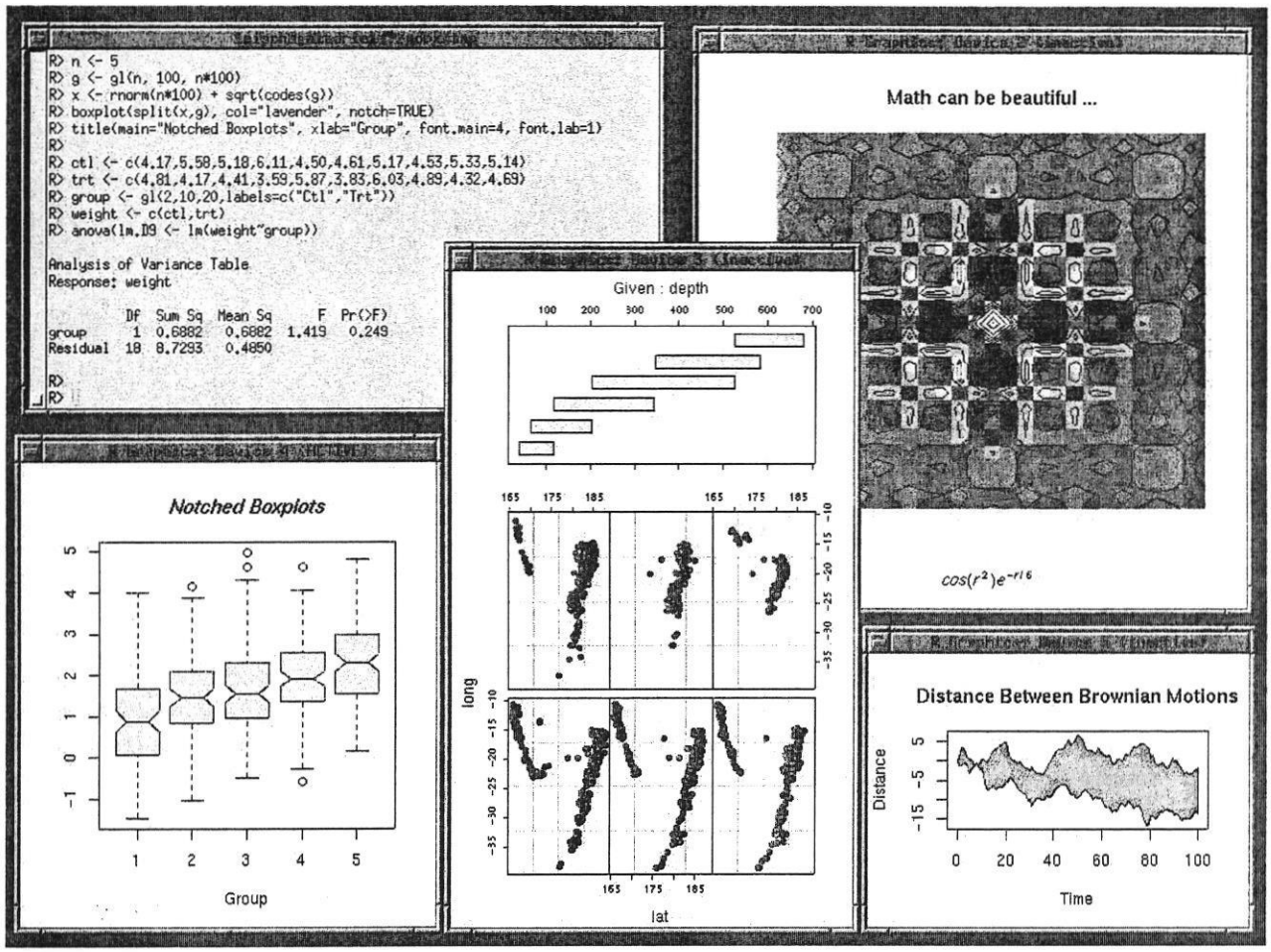

Figura 8: A CLI do R e exemplos de gráficos

Para listar as variáveis que já foram criadas, utiliza-se a função `ls`. Para executar qualquer função em R, mesmo que ela não exija o uso de argumentos, ela deve ser chamada seguida de "`()`". O valor de uma variável pode ser exibida em tela chamando-se o nome da variável na CLI do R. Por exemplo:

```
> ls()
[1] "a" "A" "b"
> a
[1] 1
> A
[1] 5
> b
[1] 6
```

As variáveis numéricas ou com texto no R são armazenadas como se fossem vetores. Por esse motivo são exibidos os indicadores "`[1]`" antes do valor da variável. Isso indica que o valor é o primeiro elemento do vetor. Vetores com mais de um elemento podem ser criados pela função de concatenação de valores "`c`". Quando um comando R é executado e não há uma atribuição do resultado a uma variável, o resultado da operação simplesmente é exibido na tela. Exemplo:

```
> v <- c(9, 5, 7, a, b)
```

```
> v
 [1] 9 5 7 1 6
> c(v,A,a)
 [1] 9 5 7 1 6 5 1
> v2 <- c(v,v)
> v2 * A
 [1] 45 25 35  5 30 45 25 35  5 30
> v2
 [1] 9 5 7 1 6 9 5 7 1 6
```

Funções em R são definidas pelo comando `function`. Por exemplo, uma função soma pode ser definida como:

```
> soma <- function (x, y) {
           x + y
         }
> ls()
[1] "a"    "A"    "b"    "soma" "v"    "v2"
> soma (2, 3)
[1] 5
```

Na chamada de função, os argumentos podem ser passados por nome ou por posição. No exemplo anterior os argumentos foram passados por posição. Um exemplo de passagem por nome é dado no próximo exemplo. As funções também podem possuir argumentos opcionais, ou seja, argumentos que assumem um valor padrão caso não forem passados na chamada da função. No exemplo a seguir é declarada e chamada uma função que tem o terceiro e quarto argumentos opcionais:

```
> conc1e2emult3e4 <- function (prim, seg, ter = 5, quar = 9) {
                      mult <- ter * quar
                      c(prim, seg, mult)
                     }
> conc1e2emult3e4 (seg = 2, prim = 1, quar=4)
 [1]  1  2 20
> conc1e2emult3e4 (1, 2, 3)
 [1]  1  2 27
> conc1e2emult3e4 (5, 6, quar = 3)
 [1]  5  6 15
```

Com esse exemplo é possível observar que uma vez que se alterou a ordem dos argumentos na chamada, por meio da utilização do nome, os demais atributos também devem ser passados por nome e, quando utiliza-se o valor padrão de um atributo, os posteriores devem ser passados à função por nome.

Um objeto R de interesse a esse trabalho é o *DataFrame*. Um *DataFrame* é similar a uma matriz, porém suas colunas podem ter tipos de dados diferentes. É a maneira utilizada neste trabalho para representar, no R, bases de dados no formato atributo-valor. É possível realizar várias operações e aplicar diversas funções aos *DataFrame*. No exemplo a seguir um pequeno subconjunto delas é utilizado. A explicação dessas funções e operações será dada após o exemplo e, para facilitar a explicação, nesse exemplo as linhas serão numeradas.

```
1 > voyage <- data.frame(
2              outlook = c("sunny", "sunny", "overcast", "overcast", "rain", "rain"),
3              temperature = c(25, 28, 23, 29, 22, 19),
4              humity = c(72, 91, 23, 29, 95, 70),
5              windy = c("yes", "yes", "yes", "no", "no", "yes"),
6              class = c("go", "dont_go", "go", "go", "go", "dont_go")
```

```
7                )

9  > voyage
10      outlook temperature humity windy   class
11 1     sunny           25      72   yes      go
12 2     sunny           28      91   yes dont_go
13 3  overcast           23      23   yes      go
14 4  overcast           29      29    no      go
15 5      rain           22      95    no      go
16 6      rain           19      70   yes dont_go

18 > names(voyage)
19 [1] "outlook"     "temperature" "humity"      "windy"       "class"

21 > voyage["humity"]
22   humity
23 1     72
24 2     91
25 3     23
26 4     29
27 5     95
28 6     70

30 > voyage[c(-1,-3)]
31   temperature windy   class
32 1          25   yes      go
33 2          28   yes dont_go
34 3          23   yes      go
35 4          29    no      go
36 5          22    no      go
37 6          19   yes dont_go

39 > subset(voyage, class == "go")
40      outlook temperature humity windy class
41 1     sunny           25      72   yes    go
42 3  overcast           23      23   yes    go
43 4  overcast           29      29    no    go
44 5      rain           22      95    no    go

46 > voyage[3,]
47      outlook temperature humity windy class
48 3  overcast           23      23   yes    go

50 > voyage [c(-1,-3,-6), c("temperature", "windy", "class")]
51   temperature windy   class
52 2          28   yes dont_go
53 4          29    no      go
54 5          22    no      go
```

Nesse exemplo, na Linha 1, é definido um *DataFrame* com os dados da base *Voyage*, e salvo na variável `voyage`. Na Linha 9, os dados contidos na variável `voyage` são exibidos na tela. É possível observar que cada coluna representa um atributo, cujo nome é apresentado no topo da mesma. Se desejarmos um vetor com esses nomes de atributos, podemos utilizar a função `names`, como mostrado na Linha 18. Na Linha 21, um único atributo da base é selecionado. Poderia ser selecionado um sub-conjunto de atributos por meio de um vetor de nomes de atributos. Essa seleção pode ser realizada também substituindo o nome do atributo pelo número inteiro que indica sua posição na base. É possível, também, selecionar um sub-conjunto de atributos

passando-se números negativos ou um sinal de negativo em frente do nome do atributo a ser omitido. Isso pode ser observado na Linha 30, onde são omitidos o primeiro e terceiro atributos. Uma maneira mais elaborada de selecionar um sub-conjunto de atributos ou exemplos de um *DataFrame* é por meio da utilização da função `subset`. Na Linha 39 são selecionados somente os exemplos cujo valor do atributo `class` é igual a `go`. Na Linha 46 e mostrado a seleção de um único exemplo, o terceiro. De maneira análoga, é possível passar um vetor de exemplos, tanto a serem exibidos, quanto a serem omitidos (números negativos). É possível selecionar um sub-conjunto de exemplos e de atributos simultaneamente. Isso pode ser observado na Linha 50, onde foram omitidos os exemplos `1`, `3` e `6` e selecionados os atributos `temperature`, `windy` e `class`.

Os conceitos da ferramenta R expostos nesta seção são necessários para a utilização, por meio da CLI do R, do sistema desenvolvido. Essa maneira de utilização é descrita nas Seções 5.1 e 5.2.

4.2 A Linguagem XML

A XML[3] – Linguagem de Marcação Extensível (*Extensible Markup Language*) – é uma linguagem derivada da SGML (ISO 8879[4]) e, desde fevereiro de 1998, é uma recomendação da W3C – Consórcio para a rede mundial de computadores (*World Wide Web Consortium*), uma organização onde funcionários, empresas associadas e o público em geral, contribuem para o desenvolvimento de padrões para a Internet. Desde sua recomendação pela W3C, a linguagem XML tem apresentado uma sensível popularização e é utilizada como padrão de troca de dados em grande quantidade dos sistemas atuais e em desenvolvimento.

Diferente de outras linguagens de marcação tais como HTML, que foram concebidas com o intuito de exibir dados com a preocupação de como eles são apresentados, a linguagem XML foi concebida com o intuito de descrever dados de acordo com o seu significado. Esta seção não tem como objetivo abordar todos os aspectos relacionados à XML, mas apenas uma breve introdução, possibilitando a compreensão dos arquivos XML utilizados no DiscoverGraphics.

Um documento XML, ou seja, um documento escrito na linguagem XML, utiliza marcações (*tags*) para descrever os seus dados. Um elemento XML é definido sempre por uma marcação de início e uma de fim, delimitando a informação descrita. As marcações sempre são indicadas entre um sinal de menor "<" e de maior ">", sendo que, para indicar a marcação final, uma barra "/" é inserida antes do nome do elemento. Por exemplo, um trecho de documento que descreve o título de um gráfico, pode descrever essa informação da seguinte maneira:

```
<grahicTitle>Gráfico de Coordenadas Paralelas</graphicTitle>
```

Nesse trecho é possível observar que o nome do elemento é `graphicTitle`; a marcação de início é dada por `<graphicTitle>`; a de fim por `</graphicTitle>`; e seu conteúdo é *Gráfico de Coordenadas Paralelas*. Em XML, nomes de elementos não são pré-definidos, sendo que o criador do documento é livre para definí-los, de preferência buscando tornar o documento auto-explicativo. Os elementos são sensíveis a caixa, ou seja, a utilização de letras maiúsculas e minúsculas nas marcações resultam em elementos distintos, por exemplo, `graphicTitle` é diferente de `graphictitle`.

[3] `http://www.w3.org/XML/`

[4] `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387`

Um elemento vazio pode substituir a marcação de início e de fim por apenas uma única marcação. Essa marcação inicia com "<" e termina com " / >", por exemplo:

```
<emptyElement />
```

Embora um elemento como esse possa parecer sem utilidade, ele pode conter atributos, que serão vistos ainda nessa seção.

O conteúdo dos elementos XML pode possuir novos elementos, criando assim uma relação de hierarquia, tal como elementos "pai", "filho" e "irmão". Por exemplo, no seguinte trecho de documento:

```
<graphic>
  <graphicTitle>Histograma</graphicTitle>
  <color>Blue</color>
</graphic>
```

os elementos `graphicTitle` e `color` são irmãos entre sí e são filhos de `graphic`, que por sua vez é pai dos dois primeiros. A identação e espaços apresentados nesse trecho de documento, embora sejam altamente recomendáveis, servem somente para torná-lo mais legível. Dessa maneira, o exemplo anterior pode ser escrito da seguinte maneira:

```
<graphic><graphicTitle>Histograma</graphicTitle><color>Blue</color></graphic>
```

Os elementos XML também podem possuir atributos. Esses atributos e seus valores devem ser inseridos na marcação de início de cada elemento. Em XML, todos os valores de atributos obrigatóriamente devem ser inseridos entre aspas. Por exemplo, inserindo um atributo `fontType` com valor *bold* no elemento `graphicTitle`:

```
<graphic>
  <graphicTitle fontType="bold">Histograma</graphicTitle>
  <color>Blue</color>
</graphic>
```

É importante observar que a marcação de início e fim de cada elemento deve ser disposta dentro de seu elemento pai. Por exemplo, o seguinte trecho **não** é um XML válido:

```
<graphic><graphicTitle>Histograma</graphic></graphicTitle>
```

Até aqui, somente foram mostrados exemplos com trechos de documentos XML, porém, não um documento completo. Um documento XML completo deve possuir inicialmente uma *declaração* XML. Essa declaração se inicia com "`<?xml`", termina com "`?>`" e deve ter, pelo menos, os atributos `version` e `encoding` que indicam, respectivamente, a versão da linguagem XML e a codificação de caractere utilizadas. Outra restrição dos documentos XML é que somente pode existir um único elemento raiz, ou seja, existe apenas um elemento sem predecessor (pai). A seguir é apresentado um exemplo de documento XML completo:

```
<?xml version="1.0" encoding="UTF-8"?>
<attribute>
  <name>humity</name>
  <type>real</type>
  <minValue>70</minValue>
  <maxValue>95</maxValue>
</attribute>
```

A seguir é apresentado um documento XML **inválido** pois viola a restrição de que deve haver apenas um elemento raiz:

```
<?xml version="1.0" encoding="UTF-8"?>
<attribute>
  <name>humity</name>
  <type>real</type>
  <minValue>70</minValue>
  <maxValue>95</maxValue>
</attribute>
<attribute>
  <name>temperature</name>
  <type>real</type>
  <minValue>19</minValue>
  <maxValue>29</maxValue>
</attribute>
```

Uma solução para o documento anterior é criar um elemento **database** e torná-lo pai dos elementos **attribute**. Sendo assim, o exemplo anterior corrigido fica:

```
<?xml version="1.0" encoding="UTF-8"?>
<database>
  <attribute>
    <name>humity</name>
    <type>real</type>
    <minValue>70</minValue>
    <maxValue>95</maxValue>
  </attribute>
  <attribute>
    <name>temperature</name>
    <type>real</type>
    <minValue>19</minValue>
    <maxValue>29</maxValue>
  </attribute>
</database>
```

Uma das grandes vantagens da linguagem XML é a sua expansibilidade, ou seja, tornar possível a inserção de novos elementos de maneira simples e com pouco ou nenhum impacto aos sistemas que utilizam o documento. Se desejarmos inserir novos elementos no exemplo anterior, isso pode ser realizado de maneira simples, por exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<database>
  <name>Voyage</name>
  <classes>
    <class>go</class>
    <class>dont_go</class>
  </classes>
  <attribute>
    <name>humity</name>
    <type>real</type>
    <minValue>70</minValue>
    <maxValue>95</maxValue>
  </attribute>
  <attribute>
    <name>temperature</name>
    <type>real</type>
    <minValue>19</minValue>
    <maxValue>29</maxValue>
  </attribute>
</database>
```

Nesse documento expandido, foram inseridos os elementos `name` para descrever o nome da base, `classes` com filhos `class` para definir as classes dos exemplos da base. Em geral, uma mudança estrutural como essa iria exigir uma alteração nos sistemas que fazem uso desse documento. Porém, é usual que sistemas que utilizam documentos XML façam uso dos elementos que necessitam e ignorem os elementos extranhos. Sendo assim, é possível expandir o documento para que atenda a novas necessidades sem que seja realizada uma alteração na implementação dos sistemas que já o utilizavam.

Essa breve introdução sobre XML apresentou apenas um sub-conjunto das propriedades dessa linguagem, as quais são necessárias para entender os documentos XML utilizados neste trabalho.

4.3 A Linguagem XSD

A grande flexibilidade dos documentos XML traz benefícios no seu uso para a troca de informações e por sistemas em constante crescimento. Porém, essa flexibilidade pode levar a acreditar que é impossível realizar qualquer validação mais criteriosa do que uma simples verificação se os documentos se adequam a sintaxe XML. Por exemplo, uma verificação quanto à adequação semântica desses documentos parece até o momento impossível. Sendo assim, como um sistema pode definir se, para ele, um determinado documento XML faz ou não sentido?

Para resolver essas e outras questões foi criada a XSD – Definição de esquemas XML (*XML Schema Definition*) – uma linguagem que define um conjunto de regras que um documento XML deve seguir para ser validado e, desde maio de 2001, é uma recomendação da W3C.

Semelhante à seção anterior, segue uma breve descrição da linguagem XSD, necessária para a compreensão de seu uso neste trabalho.

Um esquema XSD apresenta regras para disposição e conteúdo dos elementos e atributos em um documento XML. Um esquema XSD é um documento XML válido, isso apresenta vantagens tais como: não torna necessário que o usuário aprenda uma nova linguagem para definir os esquemas de documentos XML e permite que os mesmos procedimentos implementados para interpretar documentos XML também interpretem seus esquemas.

Um esquema XSD inicia com uma declaração XML, uma vez que é um documento XML, e tem como raiz um elemento `schema` definido no espaço de nome (*namespace*) *http://www.w3.org/2001/XMLSchema*. A seguir é apresentado um exemplo desse esquema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" />
```

Em XSD os elementos XML são classificados como *elementos simples* aqueles que possuem somente um conteúdo texto e como *complexos* os que possuem elementos filhos e/ou atributos. Para definir um elemento válido utiliza-se o elemento XSD `xs:element`, que pode possuir, dentre outros atributos, o atributo `name` que define o nome do elemento e `type` que define o tipo do conteúdo. A seguir é apresentado, como exemplo, um esquema XSD que define um elemento com nome `raiz` e tipo `xs:string`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="raiz" type="xs:string" />
</xs:schema>
```

Um documento XML que é validado por esse esquema pode ser observado a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<raiz>Teste</raiz>
```

Vários tipos de dados pré-definidos podem ser utilizados para restringir o conteúdo de elementos e atributos. Dentre eles podemos citar: `xs:string` para cadeia de caracteres, `xs:decimal` para números reais, `xs:integer` para números inteiros, `xs:boolean` para valores booleanos e `xs:date` para datas.

Além dos tipos básicos, outras restrições podem ser aplicadas ao conteúdo de elementos e atributos. Para isso devemos criar um novo tipo simples com o elemento `xs:simpleType` e utilizamos o elemento `xs:restriction` para fazer restrições tais como qual o menor e o maior valor de um conteúdo numérico, obrigar que o conteúdo esteja contido em uma lista de valores ou até mesmo escrever uma expressão regular que valida o conteúdo. Para evitar que as linhas iniciais de declaração do XML e de definição do elemento `xs:schema` sejam repetidas a cada exemplo, as mesmas serão omitidas nos próximos exemplos.

No exemplo a seguir, é apresentado um trecho de esquema XSD no qual são definidos três atributos simples `age`, `sex` e `citationName`. Observe que o exemplo é apenas um trecho de XSD, portanto não estamos definindo três elementos raiz, o que seria incorreto.

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0" />
      <xs:maxInclusive value="120" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="sex">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="female" />
      <xs:enumeration value="male" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="citationName">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][a-z])+, ([A-Z]\.( ){0,1})+" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Nesse exemplo, é possivel observar o elemento `xs:simpleType`. Ele é utilizado para que seja definido um novo tipo de elemento simples. No primeiro elemento (`age`), o tipo definido tem uma restrição (definida pelo elemento `xs:restriction`) que tem como base o tipo pré-definido `xs:integer`. Essa base indica que o conteúdo do elemento deve ser do tipo inteiro e que serão definidas outras restrições a ele. Essas novas restrições, definidas pelos elementos `xs:minInclusive` e `xs:maxInclusive` cujos valores do atributo `value` são, respectivamente, 0 e 120, indicam que o conteúdo do elemento `age` deve se encontrar no intervalo $[0, 120]$. De maneira análoga, foi definido o elemento `sex`, porém com restrição base `xs:string`. Nessa

restrição, foi utilizado o elemento `xs:enumeration` que define um possível valor que o conteúdo do elemento pode assumir, no caso desse exemplo, *female* ou *male*. Na restrição do último elemento (`citationName`), foi utilizado o elemento `xs:pattern` o qual permite a utilização de uma expressão regular para validar o conteúdo do elemento sendo definido. A expressão regular desse exemplo indica que o nome deve iniciar com uma letra maiúscula entre A e Z (*[A-Z]*), possuir uma ou mais letras minúsculas entre a e z (*([a-z])+*), uma vírgula e um espaço e, por fim, um ou mais conjuntos de letras maiúsculas entre A e Z seguidas por ponto e por nenhum ou um espaço (*([A-Z]\.( )0,1)+*). Essa expressão regular valida cadeias de caracteres tal como "*Lattes, C. M. G.*". As expressões regulares permitem ser combinadas por meio do caractere "|" que indica uma condição *ou*. Sendo assim, a lista de valores da restrição do elemento `sex` poderia ser substituída pela expressão regular "*female|male*", porém, quando o número de valores torna-se elevados, o uso de uma expressão regular longa pode reduzir a legibilidade. Um exemplo de trecho de documento XML que é validado pelo esquema anterior é mostrado a seguir:

```
<age>80</age>
<sex>male</sex>
<citationName>Lattes, C. M. G.</citationName>
```

enquanto que um exemplo de trecho de documento XML que não é validado pelo esquema anterior seria:

```
<age>128</age>
<sex>feminino</sex>
<citationName>McDouglas, F.</citationName>
```

As razões para esse trecho de XML não ser validado pelo esquema são: o conteúdo do elemento `age` excede o máximo permitido que é 120; o conteúdo do elemento `sex` somente pode ser *male* ou *female* e não *feminino* como consta no exemplo; e o primeiro trecho de caracteres (antes da vírgula) no elemento `citationName` permite apenas a primeira letra maiúscula e a letra *d* de *McDouglas* está em maiúscula. Embora nesse exemplo todas as restrições tenham sido violadas, apenas uma violação é suficiente para o documento XML não ser validado pelo esquema.

Até o momento só foram vistos elementos simples, ou seja, que não possuem atributos ou elementos filhos. Os elementos que possuem filhos ou atributos são considerados elementos complexos e são definidos em XSD utilizando-se o elemento `xs:complexType`. Os filhos de um elemento complexo são organizados por meio de um indicador de ordem, que pode ser um dos três elementos XSL `xs:all`, `xs:choice` e `xs:sequence`. Existem também indicadores de ocorrência, que definem quantas vezes o elemento filho pode aparecer. Os indicadores de ocorrência são: o atributo `maxOccurs` que define o número máximo de ocorrências e o atributo `minOccurs` que define o número mínimo de ocorrências de um elemento filho.

O indicador de ordem `xs:all` define que os elementos filhos declarados podem aparecer em qualquer ordem, no máximo uma única vez; o indicador de ordem `xs:choice` permite que apenas um dos elementos filhos declarados seja utilizado no documento XML e o indicador de ordem `xs:sequence` define que os elementos que ocorrem no documento XML devem aparecer na mesma sequência em que são declarados no esquema.

Os indicadores de ocorrência `xs:maxOccurs` e `minOccurs` têm valor padrão "1" para todos os indicadores de ordem, ou seja, caso sejam omitidos, os elementos filhos devem aparecer obrigatóriamente no documento XML apenas uma vez. Esses atributos podem ser declarados nos elementos de ordem ou no elemento `xs:element`. Quando são declarados no elemento de ordem, os valores se aplicam a todos os elementos filhos; quando são declarados no elemento

`xs:element`, os valores se aplicam somente ao elemento filho que recebe a declaração. Os indicadores de ocorrência podem receber qualquer valor inteiro positivo ou zero. O indicador `maxOccurs` pode receber ainda o valor *"unbound"*, indicando que o elemento não tem limite máximo de vezes que pode ocorrer. Um caso particular é no indicador de ocorrência `xs:all` para o qual o valor de `maxOccurs` não pode ser diferente de "1".

Um trecho de esquema que define elementos complexos é mostrado a seguir:

```
<xs:element name="database">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
      <xs:element name="classes" minOccurs="0">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="class" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Nesse exemplo, é definido um elemento raiz `database` que possui os filhos `name` e `classes`. O atributo `name` é obrigatório, somente pode ocorrer uma vez, e seu conteúdo deve ser do tipo `xs:string`. O elemento `classes` é opcional e, caso ocorra, deve possuir um ou mais elementos filhos `class` com conteúdo do tipo `xs:string`. Nesse exemplo, os elementos `database` e `classes` são complexos, os demais são elementos simples.

Os atributos de um elemento também são declarados por meio do elemento `xs:attribute`. A declaração de um elemento que possui os atributos `color` e `size`, sendo o primeiro opcional e do tipo `xs:string` e o segundo obrigatório e do tipo `xs:decimal`, pode ser observado no exemplo a seguir:

```
<xs:element name="car">
  <xs:complexType>
    <xs:attribute name="color" type="xs:string" />
    <xs:attribute name="size" type="xs:decimal" use="required" />
  </xs:complexType>
</xs:element>
```

Um trecho de XML que é validado com o esquema anterior é apresentado a seguir:

```
<car color="yellow with black stripes" size="5.14">Dodge Charger RT 1968</car>
```

A definição de esquemas XSL que validem os documentos utilizados por um determinado sistema é importante para garantir que, no mínimo, a informação essencial esteja presente e que não haja inconsistência semântica na mesma. Embora isso foi obtido com os exemplos vistos até o momento, foi perdida a vantagem de expansibilidade dos documentos XML. Para criarmos um esquema que permita que o documento seja expandido, é necessário utilizar o elemento `xs:any`. Com isso, indicamos que podem existir quaisquer outros elementos naquela posição. Esse elemento respeita os mesmos indicadores de ordem e ocorrência apresentados anteriormente. De maneira análoga, existe o elemento `xs:anyAttribute` que permite a expansibilidade dos atributos. Ambos `xs:any` e `xs:anyAtribute` possuem o atributo `processContents` que pode assumir os valores `strict` indicando que para que esses elementos/atributos expandidos devem obrigatóriamente possuir um esquema XSD que os valide; `lax` indicando que se houver um esquema

XSD esse esquema será utilizado para validação dos elementos/atributos expandidos, porém, se o mesmo não existir, nenhum erro ocorrerá; e `skip` indicando que os elementos/atributos expandidos não sofrerão qualquer tipo de validação. O valor padrão caso esse atributo seja omitido é `strict`.

Expandindo o exemplo anterior para permitir que outros atributos e elementos filhos sejam adicionados à `car`, obtem-se o seguinte esquema:

```
<xs:element name="car">
  <xs:complexType>
    <xs:attribute name="color" type="xs:string" />
    <xs:attribute name="size" type="xs:decimal" use="required" />
    <xs:anyAttribute />
    <xs:sequence>
      <xs:any maxOccurs="unbounded" processContents="lax"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Embora esta seja uma breve introdução à linguagem XSD, os conceitos apresentados nesta seção, permitem analisar e entender os esquemas XSD criados para os documentos XML utilizados pelo sistema DISCOVERGRAPHICS.

5 Descrição da Implementação do Sistema DiscoverGraphics

Nesta seção são abordados alguns detalhes da integração dos métodos de geração de gráficos ao projeto DISCOVER, a interface Web desenvolvida e a implementação de algumas ferramentas para auxíliar na descrição de bases de dados e na criação do arquivo de declaração de atributos.

Para a implementação do sistema de geração de gráficos, foram criados dois pacotes (*packages*) para a ferramenta R. Os pacotes R são como bibliotecas de funções que, após instalados no sistema, permitem que o usuário carregue-os para memória para fazer uso de suas funções. Os pacotes implementados foram `discGraphics` e `discUtils`:

- O pacote `discUtils` apresenta funções para a interpretação de bases de dados no formato DSX do DISCOVER, interpretação de arquivos XML, descrição de bases de dados e auxilio na criação de arquivos de descrição de dados.
- O pacote `discGraphics` apresenta funções com interfaces padrão, que utilizam primitivas gráficas e gráficos R, permitindo a visualização de bases de dados no formato DSX, previamente interpretadas pelo pacote `discUtils`.

Ambos pacotes podem ser utilizados iterativamente por meio de comandos fornecidos pelo usuário na interface de linha de comando (CLI – *Command Line Interface*) do R. Exemplos dessa utilização serão apresentados no decorrer das Seções 5.1 e 5.2. Embora essa maneira de utilização permita um maior controle sobre as funções utilizadas, ela exige um conhecimento básico da linguagem R. Sendo assim, foram também criadas funções que permitem a interpretação de arquivos XML contendo parâmetros para criação de gráficos. Isso permite que o usuário possa também utilizar o sistema de geração de gráficos de maneira rápida e sem prévio conhecimento da linguagem R. A interpretação dos arquivos XML simplificam também a comunicação entre o módulo de geração de gráficos e outras ferramentas computacionais, uma vez

que se estabelece um protocolo de comunicação padronizado. A especificação desses arquivos XML é apresentada na Seção 5.3. Na Seção 5.4 é apresentada a terceira maneira de utilização do sistema DISCOVERGRAPHICS, que é realizada por meio de uma interface WEB amigável, permitindo o uso do sistema por usuários leigos em computação. A interação do usuário com essas três interfaces: linha de comando, XML e WEB, é apresentada na Figura 9. Nessa figura é possível observar ainda a interação das interfaces entre sí, onde a interface XML acessa as funções por meio de *scripts* executados na linha de comando e a interface WEB se comunica com a interface XML para acessar as funções do R.

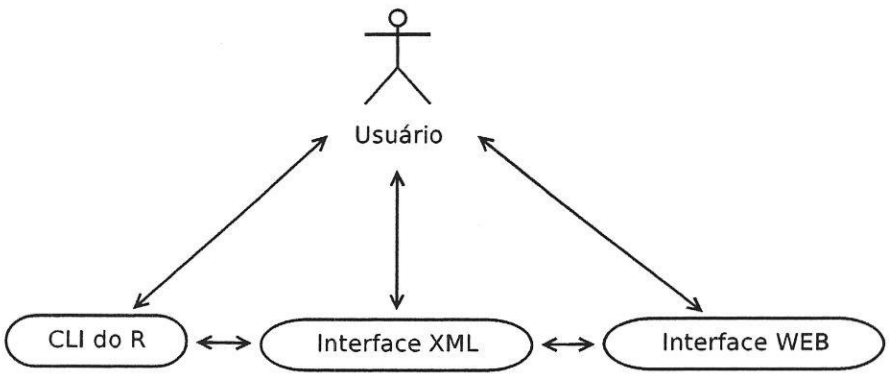

Figura 9: As três interfaces do sistema DISCOVERGRAPHICS

5.1 O Pacote `discUtils`

O pacote `discUtils` apresenta funções auxiliares para importar e exportar bases no formato DSX, interpretar arquivos XML, descrever bases de dados e gerar arquivos de descrição de dados. Essas funções são descritas a seguir, mostrando, para cada uma delas, uma breve descrição da ação da função, argumentos, valor retornado e exemplo de utilização.

`discReadDSX`: importa uma base de dados no formato DSX para um formato nativo do R (*Data Frame*). Essa conversão permite que as funções R acessem de maneira natural e mais eficiente a base de dados. Caso o arquivo de descrição de dados da DSX apresente informação de qual atributo deve ser considerado como rótulo dos exemplos – dados supervisionados –, esse atributo é colocado como último atributo do *Data Frame*;

Uso: `discReadDSX (dbname, isTest = FALSE)`

Argumentos:

dbname: cadeia de caracteres contendo o nome da base a ser lida;

isTest: booleano que indica se deve ser lida a base de teste (*.test*). Valor padrão: *falso*.

Retorno: *Data Frame* contendo os dados da base lida;

Exemplo de utilização:

Carrega a base *iris* para a variável *base*:

```
base <- discReadDSX ("iris")
```

Carrega a base de teste *iris* (iris.test) para a variável *baseteste*

```
baseteste <- discReadDSX ("iris", TRUE)
```

`discWriteDSX`: exporta uma base de dados no formato nativo do R (*Data Frame*) para o formato DSX do DISCOVER. Esta função é particularmente útil para a geração semi-automática de arquivos de descrição de dados no formato DSX. O R possui funções prontas para leitura de bases de dados no formato CSV – valores separados por vírgula (*Comma Separated Values*). Aplicativos populares de planilhas eletrônicas, tais como Microsoft Excel e OpenOffice Calc permitem salvar seus arquivos de dados nesse formato. O formato CVS geralmente não apresenta informações sobre o tipo de dados de cada atributo, nem dos valores que os atributos não-numéricos podem assumir. Sendo assim, a geração manual de um arquivo de descrição de dados pode ser uma tarefa muito trabalhosa, ou até mesmo inviável, dependendo no número de atributos em questão. Uma alternativa para a geração semi-automática do arquivo de descrição de dados na sintaxe DSX, é utilizar as funções R pré-definidas para importação dos dados no formato CSV e, por meio da função `discWriteDSX`, gerar o arquivo de dados (*.data*) e de descrição de atributos (*.names*) no formato DSX.

Uso: `discWriteDSX (ds, file = "ds", isTest = FALSE, forceNominal = c(ncol(ds)))`

Argumentos:

ds: *Data Frame* contendo os dados a serem exportados;

file: cadeia de caracteres contendo o nome da base a ser criada. Valor padrão: cadeia de caracteres *ds*;

isTest: booleano que indica se a base a ser exportada é a base de teste (*.test*). Valor padrão: *falso*;

forceNominal: vetor com as posições dos atributos que devem ser considerados nominais. Para a definição do tipo do atributo, é realizada uma análise para se descobrir se são inteiros, reais ou nominais. Entretanto há bases de dados com atributos nominais cujos valores são representados com números, por exemplo, para designar os valores de sexo "masculino"e "feminino"muitas vezes utiliza-se, respectivamente, os valores "0"e "1". Nesses casos, esse teste de tipo falha, sendo necessário que tais atributos sejam especificados manualmente pelo usuário. Valor padrão: último atributo da base, uma vez que, usualmente, a classe se encontra na posição do último atributo e os rótulos são sempre nominais;

Retorno: nenhum;

Exemplo de utilização:

Exporta a variável *base* para a base *iris2* no formato DSX, gerando os arquivos *iris2.data* e *iris2.names*:

```
discWriteDSX (base, "iris2")
```

Exporta a variável *baseteste* para a base de teste *iris2* no formato DSX, gerando o arquivo *iris2.test*:

```
discWriteDSX (baseteste, "iris2", TRUE)
```

Importa para a variável *estudoCaso* (*Data Frame*) os dados do arquivo *estudoCaso.csv*, na sintaxe CSV:

```
estudoCaso <- read.csv ("estudoCaso.csv")
```

Exporta a variável *estudoCaso* para a base de *estudoCaso* no formato DSX, gerando os arquivos *estudoCaso.data* e *estudoCaso.names*:

```
discWriteDSX (estudoCaso, "estudoCaso")
```

`numExamples`: calcula o número de exemplos de um *Data Frame*;

Uso: `numExamples (ds)`

Argumentos:

ds: *Data Frame* cujo número de exemplos será retornado;

Retorno: Número de exemplos da base passada como parâmetro;

Exemplo de utilização:

Armazena na variável *numExBase* o número de exemplos da variável *base*

```
numExBase <- numExamples(base)
```

`numAttributes`: calcula o número de atributos de uma base de dados;

Uso: `numAttributes (ds)`

Argumentos:

ds: *Data Frame* cujo número de atributos será retornado;

Retorno: Número de atributos da base passada como parâmetro, incluindo a classe;

Exemplo de utilização:

Armazena na variável *numAttBase* o número de atributos da variável *base*

```
numAttBase <- numAttributes(base)
```

`numClasses`: calcula o número de classes de um *Data Frame*, ou seja, o número de rótulos distintos que assumem os exemplos do *Data Frame*;

Uso: `numClasses (ds)`

Argumentos:

ds:*Data Frame* cujo número de classes será retornado;

Retorno: Número de classes da base passada como parâmetro;

Exemplo de utilização:

Armazena na variável *numClBase* o número de classes da variável *base*

```
numClBase <- numClasses (base)
```

`numDuplicate`: calcula o número de exemplos duplicados de um *Data Frame*. Exemplos duplicados são aqueles que possuem os mesmos valores para seus atributos e o mesmo rótulo (valor da classe) associado. Ou seja, dado dois exemplos $\mathbf{x}_1$ e $\mathbf{x}_2$, eles são considerados duplicados se $\mathbf{x}_1 = \mathbf{x}_2$ e $f(\mathbf{x}_1) = f(\mathbf{x}_2)$;

Uso: `numDuplicate (ds)`

Argumentos:

ds:*Data Frame* cujo número de exemplos duplicados será retornado;

Retorno: Número de exemplos duplicados da base passada como parâmetro;

Exemplo de utilização:

Armazena na variável *numDupBase* o número de exemplos duplicados da variável *base*

```
numDupBase <- numDuplicate (base)
```

`numConflicting`: calcula o número de exemplos conflitantes. Exemplos conflitantes são aqueles que tem os mesmos valores de atributos, porém a classe é diferente. Por exemplo, $\mathbf{x}_1 = \mathbf{x}_3$ e $f(\mathbf{x}_1) \neq f(\mathbf{x}_3)$.

Uso: `numConflicting (ds)`

Argumentos:

ds: *Data Frame* cujo número de exemplos conflitantes será retornado;

Retorno: Número de exemplos conflitantes da base passada como parâmetro;

Exemplo de utilização:

Armazena na variável *numConfBase* o número de exemplos conflitantes da variável *base*

```
numConfBase <- numConflicting (ds)
```

`distClasses`: calcula a distribuição das classes, ou seja, o número de exemplos rotulados com cada uma das classes C_j de uma base de dados.

Uso: `distClasses (ds)`

Argumentos:

ds: *Data Frame* cuja distribuição das classes será retornada;

Retorno: Vetor com j posições, sendo j o número de classes da base passada como parâmetro. Cada posição i do vetor é rotulada com o nome da classe C_i e seu valor é o número de exemplos da base rotulados com a classe C_i. Exemplo de retorno para base *iris*:

```
 setosa  versicolor  virginica
   50         50          50
```

Exemplo de utilização:

Armazena na variável *distClBase* a distribuição das classes da variável *base*

```
distClBase <- distClasses(base)
```

Armazena na variável *distPorcClBase* a porcentagem de exemplos rotulados com cada classe

```
distPorcClBase <- (distClasses(base) / numExamples(base)) * 100
```

Armazena na variável *nameMajClass* o nome da classe majoritária da variável *base*

```
nameMajClass <- names(which.max(distClasses(base)))
```

`genAttXMLDescription`: gera, em formato XML, informações sobre os atributos de uma base de dados.

Uso: `genAttXMLDescription (ds)`

Argumentos:

ds: *Data Frame* cuja descrição será retornada em formato XML;

Retorno: XML com as seguintes informações: nome do atributo, tipo de atributo (nominal/numérico) e, no caso de atributos nominais:

- os possíveis valores que o atributo pode assumir, e;
- para cada valor, a quantidade de exemplos que assumem aquele valor para o atributo em questão;

no caso de atributos numéricos:

- os valores mínimo e máximo do atributo;
- o valor do primeiro e do terceiro quartis do atributo;
- o valor da média e da mediana do atributo;
- o valor do desvio padrão do atributo.

Além disso, essas informações são obtidas para o subconjunto de exemplos de cada classe da base de dados. Isso permite que sejam observadas diferenças entre as distribuições dos atributos em diferentes classes.

Um trecho dessa descrição em formato XML pode ser observado na Figura 10. Nessa figura é considerada a base *Voyage* descrita na Seção 2. De maneira a tornar o exemplo mais conciso, são descritos apenas dois atributos, *outlook* e *temperature*, sendo o primeiro nominal e o segundo numérico. A base *Voyage* possui duas classes *go* e *dont_go*.

Exemplo de utilização:

Armazena na variável *descXMLBase* a descrição variável *base* em formato XML

```
descXMLBase <- genAttXMLDescription(base)
```

Salva a descrição da base contida na variável *descXMLBase* no arquivo *descBase.xml*. Observação: a função *saveXML* é implementada no pacote *XML* do R.

```
saveXML (descXMLBase, "descBase.xml")
```

`discExecXML`: abre um arquivo XML com comandos e executa-os. Maiores informações sobre os arquivos XML com comandos são apresentadas na Seção 5.3.

Uso: `discExecXML (filename)`

Argumentos:

filename: cadeia de caracteres que representa o nome do arquivo XML a ser interpretado e seus comandos executados.

Retorno: Um arquivo XML com o resultado do comando executado. Outros arquivos podem ser criados, de acordo com o comando executado. Usualmente são criados arquivos de imagem, quando os comandos para geração de gráficos são executados.

Exemplo de utilização:

Interpreta e executa os comandos contidos no arquivo *geraGrafHistograma.xml* e gera as respectivas saídas, de acordo com os comandos executados

```
discExecXML("geraGrafHistograma.xml")
```

O pacote `discUtils`, além de colocar a disposição do usuário final ferramentas para manipulação e extração de informações relevantes de bases de dados no formato DSX, serve de base para as funções implementadas no pacote `discGraphics` apresentado a seguir.

5.2 O Pacote `discGraphics`

Por meio da utilização de primitivas gráficas e funções gráficas de alto nível do R, neste trabalho foram implementadas no pacote `discGraphics` funções com uma interface padronizada para a geração de gráficos. Essa interface padronizada foi por nós adotada para permitir ao

```
<attributes>
 <attribute>
  <name>outlook</name>
  <type>nominal</type>
  <values>
   <value>
    <name>overcast</name>
    <quantity>2</quantity>
   </value>
   <value>
    <name>rain</name>
    <quantity>2</quantity>
   </value>
   <value>
    <name>sunny</name>
    <quantity>2</quantity>
   </value>
  </values>
  <distributions>
   <class name="dont_go">
    <value name="overcast">0</value>
    <value name="rain">1</value>
    <value name="sunny">1</value>
   </class>
   <class name="go">
    <value name="overcast">2</value>
    <value name="rain">1</value>
    <value name="sunny">1</value>
   </class>
  </distributions>
 </attribute>
```

```
 <attribute>
  <name>temperature</name>
  <type>numeric</type>
  <min>19</min>
  <firstquartil>22.25</firstquartil>
  <median>24</median>
  <mean>24.33</mean>
  <thirdquartil>27.25</thirdquartil>
  <max>29</max>
  <standarddev>3.777124</standarddev>
  <distributions>
   <class name="dont_go">
    <min>19</min>
    <firstquartil>21.25</firstquartil>
    <median>23.5</median>
    <mean>23.5</mean>
    <thirdquartil>25.75</thirdquartil>
    <max>28</max>
    <standarddev>6.363961</standarddev>
   </class>
   <class name="go">
    <min>22</min>
    <firstquartil>22.75</firstquartil>
    <median>24</median>
    <mean>24.75</mean>
    <thirdquartil>26</thirdquartil>
    <max>29</max>
    <standarddev>3.095696</standarddev>
   </class>
  </distributions>
 </attribute>
</attributes>
```

Figura 10: Exemplo de XML de descrição de atributos utilizando a base *Voyage*

usuário gerar diferentes tipos de gráfico, utilizando diferentes funções, por meio de chamadas bastante similares.

Todas as funções de geração de gráficos do pacote `discGraphics` recebem como parâmetro dois objetos do tipo lista, o objeto de dados e o objeto de configuração. O objeto de dados possui a base de dados e a relação dos atributos a serem utilizados. O objeto de configuração possui as configurações a serem utilizadas para a geração do gráfico. Cada um desses objetos é descrito brevemente a seguir:

Objeto de dados: o objeto de dados é uma lista com os ítens `ds` e `features`. o ítem `ds` é um *Data Frame*, que pode ser obtido por meio da importação de dados no formato DSX como apresentado na descrição da funcção `discReadDSX` do pacote `discUtils`. O ítem `features` é um vetor com o nome dos atributos a serem considerados na geração de gráficos.

Objeto de configuração: o objeto de configuração é uma lista com os ítens:

- `classcol`: vetor com nome das cores a serem utilizadas para diferenciação das classes dos exemplos nos gráficos. No caso do vetor possuir menos cores que o número de classes, as cores serão reutilizadas. Por exemplo, se o vetor possuir três cores e a base seis classes distintas, a primeira e quarta classes utilizarão a primeira cor do vetor, a segunda e quinta classes a segunda cor e a terceira e sexta classes a última cor. Dessa maneira, a não diferenciação de classes em um gráfico pode ser obtida atribuindo-se apenas uma cor ao vetor de cores. No caso de existirem mais cores no vetor do que classes distintas, as cores excedentes são ignoradas.
- `filename`: cadeia de caracteres indicando o nome do arquivo utilizado para armazenar o gráfico gerado. Essa opção não é utilizada caso o usuário opte por gerar o gráfico apenas para exibição na tela.
- `device`: cadeia de caracteres indicando o formato de arquivo a ser gerado ou se nenhum arquivo deve ser gerado e o gráfico deve somente ser exibido na tela. Os possíveis valores para `device` são:
  - *pdf*: arquivos de gráfico no formato PDF – *Portable Document Format*[5] – desenvolvido pela Adobe Systems. É recomendada a utilização desse formato por ser um formato que gera imagens vetoriais, ou seja, imagens que são geradas por meio de descrições geométricas de formas, não perdendo qualidade quando redimensionados;
  - *jpg*: arquivos de gráfico no formato JPEG – *Joint Photographic Experts Group*[6]. Nesse formato é realizada compactação com perda de dados, causando uma diminuição da qualidade em relação à imagem original. Por ser um formato no qual a imagem é armazenada como mapa de bits (*bitmap*), a ampliação dos gráficos pode ocasionar diminuição da qualidade;
  - *png*: arquivos de gráfico no formato PNG – *Portable Network Graphics*[7]. Esse formato é uma recomendação da W3C e permite a compactação de imagens sem perda de dados. Isso permite o armazenamento de imagens com melhor qualidade do que o formato JPEG, porém gerando, usualmente, arquivos maiores. Esse formato também armazena as imagens como mapa de bits, portanto sofrendo

[5] `http://www.adobe.com/products/acrobat/adobepdf.html`
[6] `http://www.jpeg.org/`
[7] `http://www.w3.org/TR/PNG`

as mesmas deficiências que o forato JPEG no caso de redimensionamento. O formato PNG é recomendado quando os gráficos forem inseridos em páginas web ou em documentos nos quais o formato PDF não possa ser utilizado.

- *bmp*: arquivos de gráfico no formato BMP desenvolvido pela Microsoft. Esse formato armazena as imagens como mapa de bits sem compactação. Os arquivos são, na grande maioria das vezes, consideravelmente maiores que os demais formatos descritos anteriormente e também podem sofrer perda de qualidade quando redimensionados.

No caso de um valor vazio ou diferente dos especificados anteriormente, nenhum arquivo é gerado e o gráfico é exibido na tela do computador;

- `main`: cadeia de caracteres contendo o título do gráfico. Quando deseja-se suprimir o título deve-se atribuir à `main` a cadeia de caracteres vazia;
- `legend`: valor booleando indicando se deve ser gerado, além do arquivo com o gráfico, um arquivo com a legenda do mesmo;
- `rotation`: lista com os ítens `x`, `y` e `z` indicando a rotação a ser realizada no caso de gráficos em três dimensões. Cada ítem dessa lista deve conter um número inteiro, positivo ou negativo, indicando o número de graus que cada eixo deve ser rotacionado.

Ambos objetos, de dados e de configuração, podem ser criados facilmente utilizando, respectivamente, as funções `dgBuildDataObj` e `dgBuildConfObj`. Essas e as demais funções do pacote `discGraphics` são descritas a seguir:

`dgBuildDataObj`: cria um objeto de dados;

Uso: `dgBuildDataObj (ds, features = names(ds))`

Argumentos:

ds: *Data Frame* que será utilizado para criação do objeto de dados;

features: vetor com o nome dos atributos a serem utilizados para geração de gráficos. Valor padrão: vetor com o nome de todos os atributos da base `ds`.

Retorno: Objeto de dados que consiste em uma lista contendo a base de dados e o vetor com o nome dos atributos;

Exemplo de utilização:
Cria um objeto de dados com a variável *base* e o armazena na variável *dataObjBase*

```
dataObjBase <- dgBuildDataObj(base)
```

`dgBuildConfObj`: cria um objeto de configuração;

Uso: `dgBuildConfObj (classcol <- rainbow (10), filename = "graphicOut", device = "pdf", main = "Title", legend = FALSE, rotation = list (x = 20, y = -20, z = 0))`

Argumentos: A descrição dos atributos dessa função é similar aos ítens do objeto de configuração apresentado anteriormente.

classcol: Valor padrão: vetor com dez cores definidas pela função do R `rainbow`;

filename: Valor padrão: cadeia de caracteres "*graphicOut*";

device: Valor padrão: cadeia de caracteres "*pdf*", ou seja, geração de gráficos no formato PDF;

main: Valor padrão: cadeia de caracteres "*Title*";

legend: Valor padrão: *falso*

rotation: Valor padrão: lista com os ítens x, y e z, com valores *20*, *-20* e *0*, respectivamente;

Retorno: Objeto de configuração, que consiste em uma lista contendo os ítens descritos previamente;

Exemplo de utilização:

Cria um objeto de configuração com título "*Gráfico Base*", com arquivo de saída "*graficoBase*"e os demais valores padrão e o armazena na variável *confObjBase*

```
confObjBase <- dgBuildConfObj(main = "Gráfico Base",
filename = "graficoBase")
```

`dgScatterPlot`: gera um gráfico de dispersão;

Uso: `dgScatterPlot (dataobj, paramobj)`

Argumentos:

dataobj: objeto de dados. Pelo fato do gráfico de dispersão ser bidimensional, somente são considerados os dois primeiros atributos especificados no vetor de atributos;

paramobj: objeto de configuração. Os valores de rotação não são considerados;

Retorno: Não há retorno para o valor da função. O gráfico gerado é armazenado no arquivo especificado ou é exibido em tela;

Exemplo de utilização:

Gera um gráfico de dispersão, utilizando as variáveis *dataObjBase* e *confObjBase*. O gráfico gerado pode ser observado na Figura 11

```
dgScatterPlot (dataObjBase, confObjBase)
```

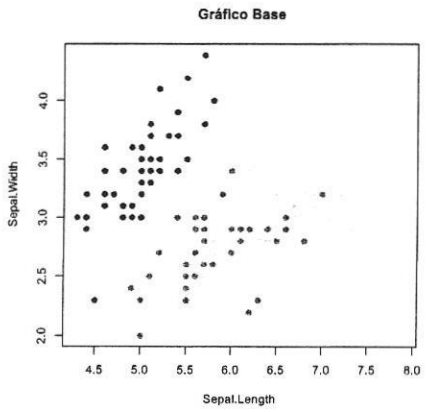

Figura 11: Exemplo de gráfico de dispersão

`dgScatterPlotMatrix`: gera uma matriz de gráficos de dispersão;

Uso: `dgScatterPlotMatrix (dataobj, paramobj)`

Argumentos:

dataobj: objeto de dados;

paramobj: objeto de configuração. Os valores de rotação não são considerados;

Retorno: Não há retorno para o valor da função. O gráfico gerado é armazenado no arquivo especificado ou é exibido em tela;

Exemplo de utilização:

Gera uma matriz de gráficos de dispersão, utilizando as variáveis *dataObjBase* e *confObjBase*. O gráfico gerado pode ser observado na Figura 12

```
dgScatterPlotMatrix (dataObjBase, confObjBase)
```

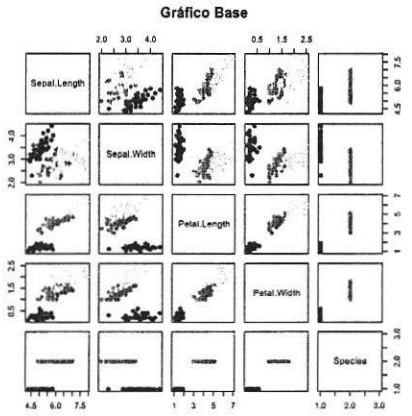

Figura 12: Exemplo de matriz de gráficos de dispersão

`dgOrderedAtt:` gera um gráfico de atributo ordenado;

Uso: `dgOrderedAtt (dataobj, paramobj)`

Argumentos:

dataobj: objeto de dados. Pelo fato do gráfico de dispersão ser unidimensional, somente é considerado o primeiro atributo especificado no vetor de atributos;

paramobj: objeto de configuração. Os valores de rotação não são considerados;

Retorno: Não há retorno para valor da função. O gráfico gerado é armazenado no arquivo especificado ou é exibido em tela;

Exemplo de utilização:

Gera gráfico de atributo ordenado, utilizando as variáveis *dataObjBase* e *confObjBase*. O gráfico gerado pode ser observado na Figura 13

```
dgOrderedAtt (dataObjBase, confObjBase)
```

`dgParallelCoord:` gera um gráfico de coordenadas paralelas;

Uso: `dgOrderedAtt (dataobj, paramobj)`

Argumentos:

dataobj: objeto de dados;

paramobj: objeto de configuração. Os valores de rotação não são considerados;

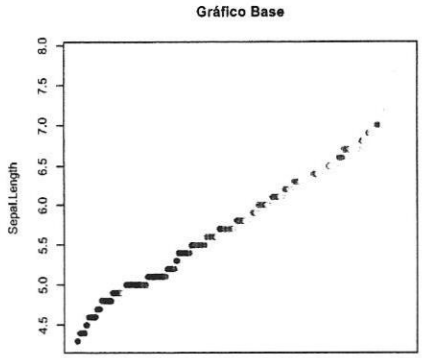

Figura 13: Exemplo de gráfico de atributo ordenado

Retorno: Não há retorno para o valor da função. O gráfico gerado é armazenado no arquivo especificado ou é exibido em tela;

Exemplo de utilização:

Gera um gráfico de coordenadas paralelas, utilizando as variáveis *dataObjBase* e *confObjBase*. O gráfico gerado pode ser observado na Figura 14

```
dgParallelCoord (dataObjBase, confObjBase)
```

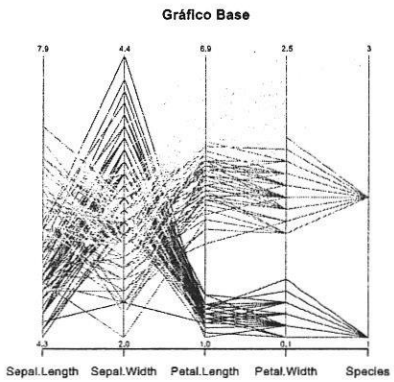

Figura 14: Exemplo de gráfico de cordenadas paralelas

`dgHistogram`: gera um histograma;

Uso: `dgHistogram (dataobj, paramobj)`

Argumentos:

dataobj: objeto de dados. Pelo fato do gráfico de dispersão ser unidimensional, somente é considerado o primeiro atributo especificado no vetor de atributos;

paramobj: objeto de configuração. Os valores de rotação não são considerados;

Retorno: Não há retorno para o valor da função. O gráfico gerado é armazenado no arquivo especificado ou é exibido em tela;

Exemplo de utilização:
Gera um histograma, utilizando as variáveis *dataObjBase* e *confObjBase*. O gráfico gerado pode ser observado na Figura 15

```
dgHistogram (dataObjBase, confObjBase)
```

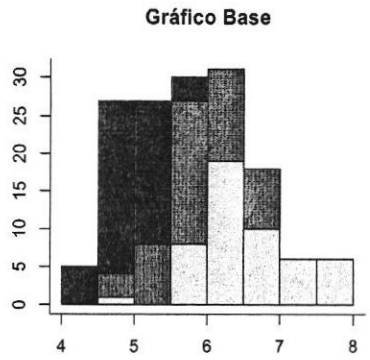

Figura 15: Exemplo de histograma

Nesta seção foram descritas as funções que permitem a geração de gráficos implementados no sistema DISCOVERGRAPHICS e foi monstrada a utilização das mesmas por meio de comandos na CLI do R. Essas funções também são utilizadas pela interface XML, descrita a seguir.

5.3 Utilização do Módulo de Geração de Gráficos por meio de Arquivos XML

Como mencionado, uma outra maneira de se utilizar o sistema implementado é por meio de arquivos XML, interpretados pela função `discExecXML` do pacote `discUtils`, descrita anteriormente. Sendo assim, o usuário não precisa conhecer a linguagem da ferramenta R, para a utilização da interface por linha de comando, mas apenas a sintaxe e semântica dos arquivos XML utilizados.

A interpretação de arquivos XML para a utilização do sistema é também muito útil pois cria uma interface padrão, permitindo que outros sistemas acessem funções de geração de gráficos de maneira simples e sem a necessidade de que sejam implementadas funções que realizem comunicação com a ferramenta R. Um exemplo de sistema que faz uso da interface por meio de arquivos XML é a interface WEB implementada, descrita na Seção 5.4.

O arquivo de XML de entrada, ou seja, que será interpretado pela função `discExecXML`, possui informações sobre a função a ser executada e os parâmetros dessa função. O esquema XSD desse arquivo é apresentado a seguir:

```
<xs:element name="webdiscoverexecute">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="procedure" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name">
```

```
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="genDbXMLDescription" />
                <xs:enumeration value="dgParallelCoord" />
                <xs:enumeration value="dgScatterPlot" />
                <xs:enumeration value="dgHistogram" />
                <xs:enumeration value="dgOrderedAtt" />
                <xs:enumeration value="dgScatterPlotMatrix" />
                <xs:enumeration value="dg3DScatterPlot" />
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="parameter" maxOccurs="unbounded">
            <xs:complexType>
              <xs:simpleContent>
                <xs:extension base="xs:string">
                  <xs:attribute name="name" use="required">
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="dbname" />
                        <xs:enumeration value="attributes" />
                        <xs:enumeration value="classcolors" />
                        <xs:enumeration value="output" />
                        <xs:enumeration value="device" />
                        <xs:enumeration value="title" />
                        <xs:enumeration value="legend" />
                        <xs:enumeration value="rotx" />
                        <xs:enumeration value="roty" />
                        <xs:enumeration value="rotz" />
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>
                </xs:extension>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

Nesse arquivo, o elemento `procedure` é instanciado uma vez para cada funcionalidade que deve ser executada pelo interpretador do XML. O elemento `name` deve conter o nome da função a ser executada e, atualmente, pode assumir somente os valores especificados no esquema. Esses valores se referem, respectivamente, às funções `genAttXMLDescription` do pacote `discUtils` e `dgParallelCoord`, `dgScatterPlot`, `dgHistogram`, `dgOrderedAtt`, `dgScatterPlotMatrix` e `dg3DScatterPlot` do pacote `discGraphics`. Para cada função, devem ser definidos seus parâmetros pelo elemento `parameter`, o qual é instanciado uma vez para cada parâmetro e possui o atributo `name` que indica o nome do parâmetro. Os nomes que esse parâmetro atuamente pode assumir, referem-se, respectivamente, ao nome da base de dados a ser considerada e aos ítens `features` do objeto de dados e `classcol`, `filename`, `device`, `main`, `legend`, `rotation.x`, `rotation.y` e `rotation.z` do objeto de configuração, ambos descritos na Seção 5.2.

Um exemplo de documento XML para geração de uma matriz de gráficos de dispersão para base *Voyage* é apresentado a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<webdiscoverexecute>
  <procedure>
    <name>dgScatterPlotMatrix</name>
    <parameter name="dbname">voyage</parameter>
    <parameter name="output">teste.png</parameter>
    <parameter name="title">Matriz de Dispersão: Base Voyage</parameter>
    <parameter name="classcolors">green,red</parameter>
    <parameter name="attributes">temperature,humity,windy</parameter>
    <parameter name="device">png</parameter>
    <parameter name="legend">true</parameter>
  </procedure>
</webdiscoverexecute>
```

Ao se chamar a função `discExecXML` para esse documento, um arquivo `teste.png` será criado, contendo gráficos de dispersão para os atributos `temperature`, `humity` e `windy`, com o título "*Matriz de Dispersão: Base Voyage*". Serão utilizadas as cores verde e vermelho para representar as classes e um arquivo `legend.teste.png` será criado contendo a legenda do gráfico.

De maneira análoga, é possível utilizar a interface XML para a utilização das demais funções de geração de gráfico implementadas, não exigindo que o usuário tenha conhecimento da linguagem R e permitindo que outros sistemas que tenham suporte a XML utilizem o sistema DiscoverGraphics de uma maneira simples e tranparente.

5.4 Interface WEB

A utilização do sistema de geração de gráficos por meio de comandos na CLI do R, ou por meio de arquivos XML, permite maior flexibilidade e um maior controle das funções utilizadas. Porém, exige do usuário alguns conhecimentos básicos da linguagem R ou da sintaxe de arquivos XML. Acredita-se que com um pequeno tempo de estudo, até mesmo usuários que não são da área de computação podem obter tais conhecimentos. Porém, muitos desses usuários podem não dispor desse tempo, podem acreditar que os resultados não compensam o tempo investido, podem se sentir desmotivados a utilizarem o módulo de geração de gráficos ou, até mesmo, se sentir intimidados com o obstáculo de adquirirem conhecimentos fora de sua área de atuação, no caso de usuários que não sejam da área de computação.

Visando minimizar tais problemas, e buscando difundir a utilização de métodos de geração de gráficos para um melhor entendimento dos dados sendo analisados, foi decidido desenvolver uma interface amigável e intuitiva, permitindo que usuários sem conhecimento de computação gerem gráficos e obtenham estatísticas de suas bases de dados em poucos minutos. A interface possui, ainda, ferramentas para transformação das tabelas de descrição das bases de dados em código de tabelas em LaTeX, poupando um tempo significativo dos pesquisadores que realizavam tal trabalho manualmente com o objetivo de publicar resultados experimentais.

Antes da elaboração da interface, foram analizados seus principais objetivos e realizado o levantamento dos requisitos do sistema. Alguns dos principais requisitos encontrados foram:

- Proporcionar ao usuário uma interface de fácil utilização;
- Permitir que multiplos usuários acessem o sistema ao mesmo tempo e que esse acesso concorrente seja transparente ao usuário;

- Disponibilizar ao usuário a possibilidade de enviar, de maneira simples, suas bases de dados ao computador no qual o sistema está sendo executado;
- Proteger os dados de cada usuário, não permitindo que terceiros os acessem;
- Permitir que sejam realizadas expansões do sistema, de maneira simples e com o menor número de alterações possível.

Considerando tais requisitos, optou-se por implementar um sistema WEB, uma vez que as plataformas de desenvolvimento WEB proporcionam funcionalidades que simplificam a resolução desses requisitos, e, com a grande popularização da internet, a grande maioria das pessoas está cada vez mais habituada a esse tipo de interface.

A arquitetura utilizada pelo sistema WEB desenvolvido foi do cliente web magro[8] (Conallen, 2000) no qual a grande maioria do processamento é realizado por um computador servidor e o computador cliente, que é a máquina utilizada pelo usuário, somente é encarregado de requisitar páginas ao servidor e exibir as páginas recebidas. A descrição dessa arquitetura, aplicada ao sistema WEB implementado, é mostrada de maneira simplificada a seguir, e é apresentada graficamente na Figura 16.

1. O usuário inicia um navegador WEB no computador cliente e acessa o endereço do servidor WEB;
2. O navegador faz uma requisição ao servidor, por meio do protocolo HTTP[9]. O servidor de aplicação, que é um software executando no computador servidor, recebe a requisição, a interpreta e busca a página correspondente à solicitação;
3. A página de resposta pode ser estática ou dinâmica. Em uma página estática, seu conteúdo não muda, e a resposta enviada ao cliente é a própria página. Uma página dinâmica possui código executável que gera o conteúdo da página no momento que a página é acessada.
4. O código da página, dependendo na necessidade, pode se comunicar com procedimentos R, por meio da interface XML disponibilizada no pacote `discUtils`;
5. A página de resposta é enviada ao cliente para ser exibida no navegador WEB.

O sistema WEB foi implementado utilizando a plataforma Java EE – *Java Enterprise Edition*[10] desenvolvida pela Sun Microsystems. A plataforma Java EE permite o desenvolvimento de sistemas WEB robustos e sua utilização é cada vez mais freqüente. Embora o sistema WEB desenvolvido neste trabalho seja relativamente simples e não necessite de uma plataforma tão elaborada quanto Java EE, essa tecnologia foi escolhida para suprir necessidades que possam surgir em futuras expansões da interface. Por exemplo, sua utilização por outros módulos do DISCOVER, reutilizando o código desenvolvido neste trabalho.

No computador servidor, o sistema implementado é composto de um *arquivo de configuração*, *arquivos de geração de interface para chamada de funções*, *arquivos de usuários* e *páginas WEB*. Cada um desses elementos é descrito a seguir:

[8] Tradução literal de *Thin Web Client*

[9] *Hypertext Transfer Protocol* – `http://tools.ietf.org/html/2616`

[10] `http://java.sun.com/javaee/` – a plataforma Java EE era conhecida como J2EE porém atualmente teve seu nome simplificado pela empresa que a desenvolve

[10] `http://www.sun.com/`

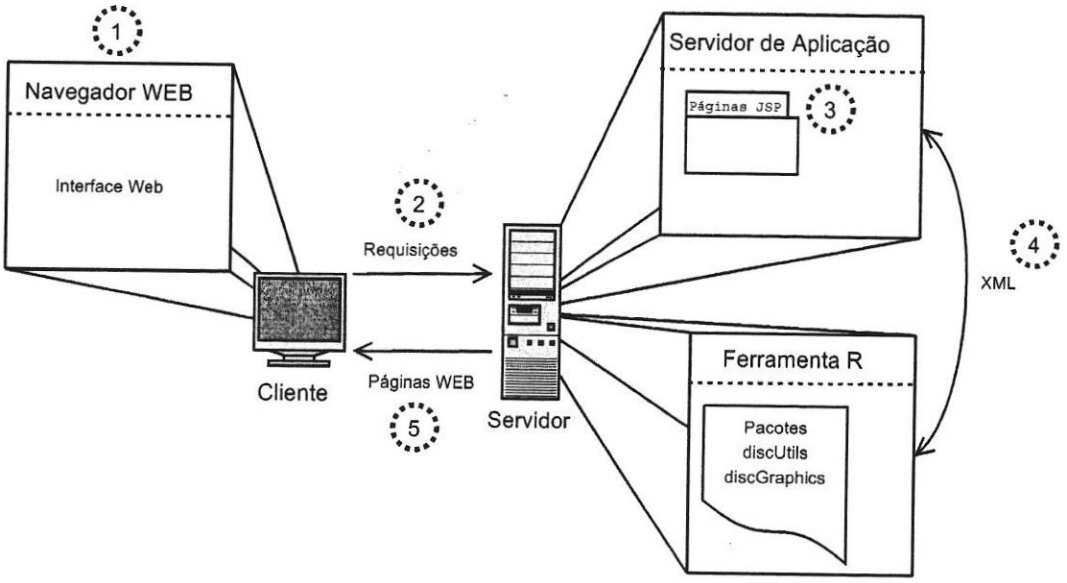

Figura 16: Figura esquemática da arquitetura do sistema WEB desenvolvido

Arquivo de configuração: É um arquivo XML, no qual são definidos os parâmetros necessários para o funcionamento do sistema. Esse arquivo tem como nome `appconfig.xml` e é armazenado no diretório base, `basedir`, definido pelo usuário administrador no momento de instalação do sistema. O esquema XSD desse arquivo é o seguinte:

```
<xs:element name="appconfig">
  <xs:complexType>
    <xs:all>
      <xs:element name="userbasedir" type="xs:string" />
      <xs:element name="rexecutable" type="xs:string" />
      <xs:element name="rbasefile" type="xs:string" />
      <xs:element name="inputxmldir" type="xs:string" />
      <xs:any minOccurs="0" maxOccurs="unbounded" processContents="lax" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

Nesse arquivo, o elemento `userbasedir` deve conter o caminho para o diretório no qual são armazenados os arquivos dos usuários; o elemento `rexecutable` deve conter o caminho e o nome do arquivo executável da ferramenta R; o elemento `rbasefile` deve conter o caminho e nome do arquivo com código R padrão que deve ser executado pela interface ao chamar a ferramenta R; e o elemento `inputxmldir` deve conter o diretório no qual estão os arquivos de geração de interface. O último elemento, definido por `xs:any`, é inserido de modo a permitir que outros elementos estejam presentes no XML. Embora esses elementos atualmente não são considerados pela interface, eles podem ser utilizados por extensões ou novos módulos futuramente adicionados à mesma.

Arquivos de geração de interface para chamada de funções: São arquivos em formato XML, que possuem os dados que o usuário deverá informar de maneira a executar cada uma das funções disponibilizadas pela interface. Por exemplo, para a geração de um histograma, o arquivo XML de geração de interface do histograma informa ao sistema WEB que deve ser solicitado ao usuário informações, tais como, a base de dados e o

atributo a serem considerados para a geração do gráfico e as cores utilizadas para cada uma das classes da base. Essa maneira de construção de interface foi escolhida por tornar mais simples a inserção de novas funcionalidades à interface. Por exemplo, para a inserção de um novo método de geração de gráficos, é necessário apenas que se crie um arquivo de geração de interface, e o próprio sistema WEB vai interpretá-lo, criar a interface e realizar a chamada da função do novo método. Os arquivos de geração de interface são armazenados em um diretório `interfacedir`, definido no arquivo de configuração do sistema. A seguir é apresentado o esquema XSD desses arquivos:

```
<xs:element name="procedurecall">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="language" type="xs:string" />
      <xs:element name="procedure" type="xs:string" />
      <xs:element name="input" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="type" use="required">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="dbname" />
                <xs:enumeration value="string" />
                <xs:enumeration value="boolean" />
                <xs:enumeration value="combobox" />
                <xs:enumeration value="classcolors" />
                <xs:enumeration value="attribute" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="name" type="xs:string" />
          <xs:sequence minOccurs="0">
            <xs:element name="label" type="xs:string" />
            <xs:element name="quantity">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:pattern value="([1-9])+([0-9])*|all"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="option">
              <xs:complexType>
                <xs:all>
                  <xs:attribute name="name" type="xs:string" />
                  <xs:attribute name="label" type="xs:string" />
                </xs:all>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Nesse arquivo, o elemento `language` define a maneira que a função será chamada. Atualmente, o único valor possível é "R", porém esse elemento foi criado para possibilitar futuras expansões. O elemento `procedure` deve conter o nome do procedimento a ser executado. O elemento `input` deve ocorrer uma vez para cada informação que deve ser fornecida pelo

usuário. Essa informação pode ser dos seguintes tipos:

dbname: Quando essa informação é solicitada ao usuário, é exibida a lista de bases que o usuário carregou para o servidor, para que o mesmo selecione uma delas.

string: Quando essa informação é solicitada, é exibida uma caixa de texto de uma linha na qual o usuário pode fornecer uma cadeia de caracteres.

boolean: Quando essa informação é solicitada, é exibida uma caixa de seleção, a qual pode ser alterada pelo usuário para apenas dois estados, selecionada e não selecionada.

combobox: Quando essa informação é solicitada, é exibida uma lista de ítens e o usuário pode selecionar apenas um deles. Essa lista é definida pelo elemento `option`. Esse elemento `option` deve ocorrer uma vez para cada ítem a ser inserido na lista e é composto de dois outros elementos: `label` que define o texto que será exibido ao usuário e `name` que será o valor retornado quando o ítem for selecionado.

classcolors: Quando essa informação é solicitada, é exibida uma caixa de seleção com várias cores, para cada classe da base selecionada, no caso de bases supervisionadas. As cores selecionadas pelo usuário serão utilizadas pelas funções gráficas para diferenciar os valores das classes da base.

attribute: Quando essa informação é solicitada, é exibida uma ou mais caixas de seleção. Os ítens dessas caixas são os nomes dos atributos da base selecionada, dos quais o usuário pode escolher um atributo por caixa. O número de caixas de seleção exibidas é definido pelo elemento `quantity`, que pode conter um valor inteiro positivo ou a cadeia de caracteres *all*, e nesse caso será exibida uma caixa para cada atributo da base.

O elemento `input` pode ainda conter um título informativo definido no elemento filho `label`, cujo conteúdo será exibido ao usuário na interface.

A fim de ilustrar essa interface, na Figura 17 é apresentada a interface gerada pelo arquivo de geração de interface para chamada de funções, descrito a seguir, considerando a base de dados *Voyage* – Tabela 2.

```
<procedurecall>
  <language>R</language>
  <procedure>dgHistogram</procedure>
  <input type="string" name="title">
    <label>Título do gráfico</label>
  </input>
  <input type="classcolors" name="classcolors">
    <label>Cores das classes</label>
  </input>
  <input type="attribute" name="attributes">
    <label>Atributos</label>
    <quantity>1</quantity>
  </input>
  <input type="combobox" name= "device">
    <label>Tipo de saída</label>
    <option>
      <name>jpg</name>
      <label>JPG</label></option>
    <option>
      <name>png</name>
      <label>PNG</label>
    </option>
```

```
    <option>
      <name>pdf</name>
      <label>PDF</label>
    </option>
    <option>
      <name>bmp</name>
      <label>BMP</label>
    </option>
  </input>
  <input type="boolean" name= "legend">
    <label>Gerar legenda</label>
  </input>
</procedurecall>
```

Figura 17: Exemplo de interface gerada pelo sistema WEB

Arquivos de usuários: Os arquivos de usuários são compostos de bases de dados no formato DSX que os usuários carregam para o servidor, gráficos gerados por meio da interface e outros arquivos temporários criados pelo sistema. Esses arquivos são armazenados em um diretório de usuários, `userdir`, definido no arquivo de configuração. Dentro de `userdir`, cada usuário do sistema possui um sub-diretório, cujo nome é igual ao *login* do usuário e é onde são armazenados seus arquivos pessoais. O conteúdo desse diretório não pode ser acessado diretamente pelo usuário, mas somente por meio das funcionalidades da interface;

Páginas WEB: São páginas dinâmicas implementadas com a tecnologia JSP – *Java Servlet Pages* – na qual é possível misturar trechos de código HTML, para a criação da parte estática da página, e trechos de código Java para geração de conteúdo dinâmico. São essas páginas que realizam a manipulação dos arquivos descritos anteriormente e promovem toda a dinâmica do sistema. A localização física das páginas WEB no disco depende do Servidor de Aplicação utilizado e é definido no momento de sua instalação. Na Figura 18 é apresentada, de maneira esquemática, a interação entre as páginas JSP implementadas. Nessa figura, uma seta saindo de uma página A e chegando a uma página B indica que é possível acessar a página B por meio de uma ligação (*link*) contida na página A e, quando existe envio de informações, uma breve descrição das mesmas é informada sobre a seta. A página *defaulterror.jsp* exibe erros que ocorrem em outras páginas do sistema, com exceção

das páginas *login.jsp* e *uploaddb.jsp* que possuem suas próprias páginas de erro. A seguir, uma breve descrição de cada uma das páginas é apresentada:

index.jsp: Primeira página exibida quando o usuário acessa o sistema. Possui uma breve informação do sistema.

stylemenu.jsp: Página que contém o menu do sistema. Não é exibida sozinha, mas é, incluída por todas as demais páginas.

about.jsp: Página com informações da instituição onde o sistema foi desenvolvido, bem como a versão do mesmo.

help.jsp: Página com informações de ajuda ao usuário na utilização do sistema.

login.jsp: Página que permite ao usuário efetuar seu *login* no sistema. Ela não é chamada diretamente por nenhuma das páginas, porém é exibida quando o usuário tenta acessar uma página protegida. Com excessão da página *errorlogin.jsp*, todas as páginas descritas a seguir estão protegidas.

errorlogin.jsp: Página que apresenta mensagem de erro caso o usuário informe *login* e/ou senha inválidos.

uploaddb.jsp: Página que permite ao usuário carregar (realizar *upload*) suas bases de dados no formato DSX para o servidor.

erroruploaddb.jsp: Página exibida caso haja erro ao carregar uma base de dados do usuário para o servidor. Nela é exibida a causa do erro.

selectdb.jsp: Página que permite que o usuário selecione com qual base deseja trabalhar. A base já deve ter sido carregada para o servidor.

dbinfo.jsp: Página que apresenta diversas informações sobre a base selecionada pelo usuário (ver *selectdb.jsp*). Atualmente, são exibidas todas as informações geradas pela função `genAttXMLDescription` do pacote `discUtils`, descrita anteriormente.

selectmultipledb.jsp: Página que permite ao usuário selecionar mais de uma base de dados. É utilizada quando o usuário desejar efetuar operações com várias bases de dados, por exemplo, criar uma tabela com a descrição de todas essas bases.

multipledbinfo.jsp: Página que apresenta, em forma de tabela, diversas informações referentes a uma ou mais bases de dados. Na versão atual do sistema, essas informações são: nome do conjunto de dados, número de exemplos, número de exemplos duplicados, número de atributos nominais e numéricos e, para bases de dados supervisionadas, número de exemplos conflitantes, número de classes, nome das classes, percentual de exemplos rotulados com cada classe e o erro da classe majoritária.

viewlatex.jsp: Página que exibe as tabelas de descrição de bases de dados apresentadas no sistema em tabelas no formato LaTeX, possibilitanto que o usuário simplesmente copie o código e cole em seu documento LaTeX.

selectgraphic.jsp: Página que permite ao usuário escolher um método de geração de gráficos para aplicar a uma base de dados. Essa página apresenta, para cada método, uma breve descrição e um gráfico demonstrativo que tem como objetivo auxiliar o usuário em sua escolha.

generateinput.jsp: Página que gera a interface a partir de um arquivo de geração de interface para chamada de função. A informação que indica qual arquivo deve ser considerado é recebida de outra página. Por exemplo, a página *selectgraphic.jsp* descrita anteriormente, onde o usuário seleciona o gráfico que deseja gerar; a partir dessa

informação, a página *generateinput.jsp* interpreta o respectivo arquivo de geração de interface e solicita ao usuário os dados necessários. Após o usuário fornecer os dados solicitados, a página realiza a chamada da função.

getoutput.jsp: Página que exibe os resultados de uma chamada de função realizada pela página *generateinput.jsp* descrita anteriormente. Para isso, é analisado o arquivo XML resultante da chamada da função e as informações são exibidas ao usuário de acordo com seu tipo de dado. Dados que podem ser interpretados diretamente pelo navegador WEB são exibidos na própria página, enquanto que para os dados que necessitem de uma aplicação externa para seu processamento, é realizada uma solicitação ao usuário para que os dados sejam armazenados no computador cliente. Por exemplo, gráficos no formato JPEG são exibidos diretamente no navegador, enquanto que gráficos no formato PDF devem ser armazenados no computador cliente.

defaulterror.jsp: Página que exibe erros genéricos que possam ocorrer no sistema. Ela recebe da página que gerou o erro e informações sobre o mesmo e as exibe ao usuário.

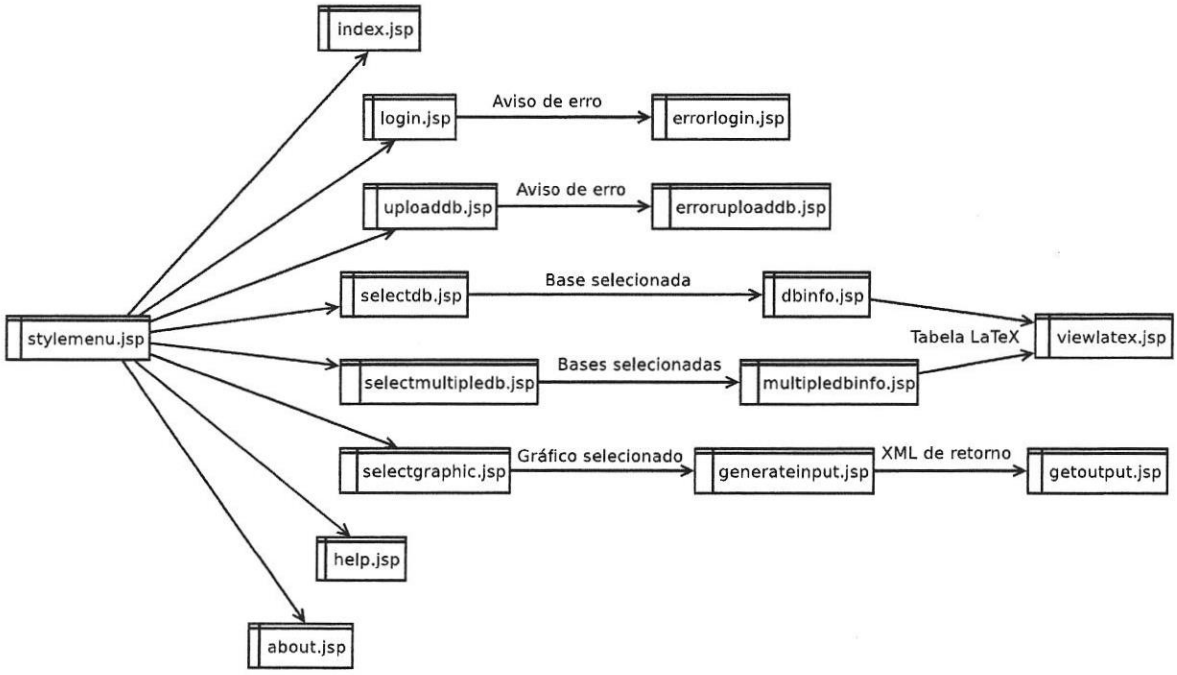

Figura 18: Esquema de interação entre as páginas JSP implementadas na interface WEB

A implementação da interface WEB do sistema DISCOVERGRAPHICS da maneira que foi realizada neste trabalho, disponibiliza tanto a usuários leigos quanto a futuros desenvolvedores de expansões para interface, um ambiente de fácil utilização, além de ser suficientemente flexível para permitir modificações e adição de novas funcionalidades de maneira simples.

6 Conclusão

A compreensão dos dados utilizados é de fundamental importância no processo de Mineração de Dados. Uma das maneiras de se obter tal compreensão é por meio da utilização de repre-

sentações gráficas desses dados. Essas representações gráficas permitem aos usuários analisarem o comportamento bem como alguns padrões nos dados, muito mais facilmente do que percorrendo centenas ou até milhares de dados numéricos e/ou categóricos. Tendo em conta a grande contribuição que métodos de geração de gráficos de dados podem trazer para o processo de MD, neste trabalho foi implementado o sistema DISCOVERGRAPHICS. Consideramos que a integração desse sistema ao projeto DISCOVER, em desenvolvimento no nosso laboratório de pesquisa LABIC, contribuirá com as pesquisas relacionadas à aquisição automática de conhecimento.

Os requisitos levantados para o sistema desenvolvido, tais como ser de fácil utilização, porém flexível, atendendo assim a diversos perfis de usuários além de apresentar possibilidade de expanção para a inserção de novas funcionalidades, foram atendidos. Para isso foram disponibilizadas três interfaces para a utilização do DISCOVERGRAPHICS: por meio de linha de comando, por meio de arquivos XML e por meio de uma interface WEB.

Além de atender os objetivos principais, consideramos que este trabalho contribui com o desenvolvimento de ferramentas úteis e que possibilitaram a automatização de tarefas tais como a geração de arquivos de descrição de dados na sintaxe DSX do DISCOVER e a criação de tabelas LaTeXpara a descrição de bases de dados, freqüentemente utilizadas nos documentos científicos elaborados nessa área de pesquisa.

Referências

Baranauskas, J. A. (2001). Extração automática de conhecimento por múltiplos intutores. Tese de Doutorado. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08102001-112806/restrito/tese.pdf.

Baranauskas, J. A. e Batista, G. E. A. P. A. (2000). O projeto DISCOVER: Idéias iniciais (comunicação pessoal).

Batista, G. E. A. P. A. (2001). Sintaxe padrão do arquivo de exemplos do projeto DISCOVER. http://www.icmc.sc.usp.br/~gbatista/SintaxePadraoFinal.htm.

Batista, G. E. A. P. A. (2003). Pré-processamento de dados em aprendizado de máquina supervisionado. Tese de Doutorado, ICMC-USP, http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06102003-160219/publico/TeseDoutorado.pdf.

Batista, G. E. A. P. A. e Monard, M. C. (2003). Descrição da arquitetura e do projeto do ambiente computacional DISCOVER LEARNING ENVIRONMENT — DLE. Technical Report 187, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_187.pdf.

Batista, G. E. A. P. A. e Monard, M. C. (2006). The Discover Object Library (dol) user's manual. Technical Report em preparação, ICMC-USP.

Becker, R. A., Chambers, J. M., e Wilks, A. R. (1988). *The New S Language*. Chapman & Hall, London.

Bernardini, F. C. (2006). Combinação de classificadores simbólicos utilizando medidas de regras de conhecimento e algoritmos genéticos. Tese de Doutorado, ICMC-USP.

Chambers, J. M. (1998). *Programming with Data*. Springer, New York. ISBN 0-387-98503-4.

Chambers, J. M. e Hastie, T. J. (1992). *Statistical Models in S*. Chapman & Hall, London.

Chiara, R. (2003). Aplicações de técnicas de *data mining* em *logs* de servidores web. Dissertação de Mestrado, ICMC-USP.

Conallen, J. (2000). *Building Web applications with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Dosualdo, D. G. (2003). Investigação de regressão no processo de mineração de dados. Dissertação de Mestrado, ICMC-USP.

Ferro, M. (2004). Aquisição de conhecimento de conjuntos de exemplos no formato atributo valor utilizando aprendizado de máquina relacional. Dissertação de Mestrado, ICMC-USP. `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-16112004-095938/publico/Dissertacao.pdf`.

Ferro, M. (2006). Aquisição de conhecimeno utilizando aprendizado de máquina relacional. Qualificação de Doutorado, ICMC-USP.

Fischer, S., Klinkenberg, R., Mierswa, I., e Ritthoff, O. (2002). Yale: Yet Another Learning Environment — Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund.

Grinstein, G. G. e Ward, M. O. (2002). *Introduction to Data Visualization*, volume 1 of *1*, chapter 1, pages 21–45. Morgan Kaufmann, San Francisco, CA, 1 edition. San Francisco, CA.

Kohavi, R., Sommerfield, D., e Dougherty, J. (1994). *$\mathcal{MLC}$++: A Machine Learning Library in C++*. IEEE Computer Society Press.

Lee, H. D. (2005). Seleção de atributos importantes para a extração de conhecimento de bases de dados. Tese de Doutorado, ICMC-USP. `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-22022006-172219/publico/tese_huei.pdf`.

Martins, C. A. (2003). Uma abordagem para pré-processamento de dados textuais em algoritmos de aprendizado. Tese de Doutorado, ICMC-USP. `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-08032004-164855/publico/tese-cam.pdf`.

Matsubara, E. T. (2004). O algoritmo de aprendizado semi-supervisionado co-training e sua aplicação na rotulação de documentos. Dissertação de Mestrado, ICMC-USP. `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19082004-092311/publico/edsontmDissertacao.pdf`.

Matsubara, E. T., Martins, C. A., e Monard, M. C. (2003). *PRETEX*: Uma ferramenta para pré-processamento de textos utilizando a abordagem *bag-of-words*. Technical Report 209, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_209.pdf`.

Melanda, E. A. (2004). Pós-processamento de regras de associação. Tese de Doutorado, ICMC-USP. `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-13012006-171753/publico/TeseEdsonMelandaVersaoCorrigida.pdf`.

Metz, J. (2006). Interpretação de clusters gerados por algoritmos de agrupamento hierárquico. Dissertação de Mestrado, ICMC-USP.

Milaré, C. R. (2003). Extração de conhecimento de redes neurais. Tese de Doutorado, ICMC-USP. `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-11082004-004358/publico/claudia_milare.pdf`.

Paula, M. F. (2003). Ambiente para disponibilização de conhecimento. Dissertaçao de Mestrado, ICMC-USP.

PERL (1999). *Programming in PERL*. Morgan Kaufmann Publishers, Inc.

Pila, A. D. (2003). Algoritmos genéticos para construção de regras de conhecimento com propriedades específicas. Exame de Qualificação de Doutorado, ICMC-USP.

Prati, R. C. (2003). O *framework* de integração do sistema DISCOVER. Dissertação de Mestrado, ICMC-USP, `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-20082003-152116/`.

Prati, R. C. (2006). Novas abordagens em aprendizado de máquina para a geração de regras, classes desbalanceadas e ordenação de casos. Tese de Doutorado, ICMC-USP.

Prati, R. C., Baranauskas, J. A., e Monard, M. C. (2001a). Extração de informações padronizadas para a avaliação de regras induzidas por algoritmos de aprendizado de máquina simbólico. Technical report, ICMC-USP. `ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/RT_145.ps.zip`.

Prati, R. C., Baranauskas, J. A., e 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`.

Prati, R. C., Baranauskas, J. A., e Monard, M. C. (2002). Padronização da sintaxe e informações sobre regras induzidas a partir de algoritmos de aprendizado de máquina simbólico. *Revista Eletrônica de Iniciação Científica*, 2(3). `http://www.sbc.org.br/reic/edicoes/2002e3`.

Pugliesi, J. B. (2004). Pós-processamento de regras de regressão. Tese de Doutorado, ICMC-USP.

Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann, CA.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Rezende, S. O., Pugliesi, J. B., Melanda, E. A., e Paula, M. F. (2003). *Mineração de Dados*, volume 1 of *1*, chapter 12, pages 307–336. Editora Manole, Barueri, SP, Brasil, 1 edition. ISBN 85-204-1683-7.

Rozante, T. A. A. (2003). Implantação do reuso de componentes no processo de desenvolvimento de software. Tese de Mestrado, ICMC-USP.

Sanches, M. K. (2003). Aprendizado de máquina semi-supervisionado: Proposta para rotular exemplos a partir de poucos exemplos rotulados. Dissertação de Mestrado, ICMC-USP.

Venables, W. N. e Smith, D. M. (2004). An introduction to R. Technical report, The R Project. `http://cran.r-project.org/doc/manuals/R-intro.pdf`.

Weiss, S. M. e Indurkhya, N. (1998). *Predictive Data Mining: A Practical Guide*. Morgan Kaufmann, San Francisco, CA.

Witten, I. H. e Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2. edition.